Arquitectura Avançada de Computadores

2005-2006

RELATÓRIO

Simulador de Cache

Dezembro de 2005

Autor:

Manuel Maia (ei00128@fe.up.pt)

Docente da disciplina:

João Canas Ferreira (jcf@fe.up.pt)

Índice

1 Introdução

Este trabalho foi realizado no âmbito da disciplina de Arquitectura Avançada de Computadores do 5º ano da LEIC, ano lectivo de 2005/2006. Consistiu em desenvolver um simulador de cache genérico. Entenda-se por cache como um nível de memória presente na arquitectura de qualquer computador moderno, parte-se do pressuposto que o leitor compreende minimamente o funcionamento de uma cache e qual o seu objectivo.

O principal objectivo deste trabalho é a partir de um conjunto de acessos a memória verificar a eficácia de uma determinada arquitectura de cache. A partir desse conjunto de acessos, que estão descritos num ficheiro, deve ser possível simular o funcionamento da cache sendo as características da mesma especificada pelo utilizador e obter estatísticas relevantes que demonstrem a eficácia da cache. A simulação tem como objectivo medir os seguintes parâmetros:

- Número de acessos a memória, mostrando quantos acessos de cada tipo (leitura, escrita, ou busca de instrução).
- Número de acertos (hits) e falhas (misses).
- Número de write-backs ou write-throughs.
- Acessos ao nível superior.

Outro objectivo é o de correr a simulação em modo gráfico iterativo. Isto significa que o utilizador pode correr a simulação passo-a-passo e ver o estado da cache mudar consoante se realizem acessos. Também pode aumentar e diminuir a velocidade de simulação durante o decorrer da simulação.

2 Descrição

Nesta secção descreve-se com algum detalhe algumas funcionalidades do programa, seguido por mais pormenores de algumas opções tomadas, como a arquitectura, algoritmos e detalhes da implementação.

2.1 Funcionalidades

Este programa tem como principal funcionalidade a visualização iterativa da simulação por meio de uma interface gráfica. Nesta interface o utilizador pode ver a simulação em modo play, que vai apresentado a sequência de acessos a memória com a cadência especificada por defeito, embora possa ser alterada usando a scroll-bar presente na interface principal que serve para definir a velocidade da execução da simulação. Em complemento com a funcionalidade play, é possível pausar a simulação usando o botão pause. Outra funcionalidade é o botão next-step, com esta opção o utilizador pode simular um acesso a memória de cada vez.

Como se pode ver na Figura 1 teve-se muito em conta a usabilidade e facilidade de utilização, as opções principais estão bem visíveis e as estatísticas também.

Outra funcionalidade será executar a simulação em modo silencioso, isto é, apresenta só as estatísticas durante o decorrer da simulação sem precisar de correr a simulação em modo gráfico completo, que é bastante lento e pode usar muitos recursos da máquina em que está a correr. Para simulações muito grandes, isto é, com vários milhões de acessos, é aconselhável correr em modo texto numa consola configurando a cache com passagem de argumentos. Nestes argumentos estão o tamanho da cache, o tamanho de bloco, associatividade, política de substituição, política de escrita e o nome do ficheiro de acessos.

Voltando à interface gráfica, o utilizador pode criar dois tipos de simulação, uma para split-cache e outra para unified-cache. Na área de desenho pode-se ver o esquema da cache, com os blocos representados por um rectângulo e algumas informações sobre qual a tag para cada bloco, se o bloco está limpo, se está dirty e informação da cache.

Por ultimo o utilizador pode abrir uma janela de diálogo em que pode ver os acessos ao nível de memória superior e gravar para um ficheiro de texto.

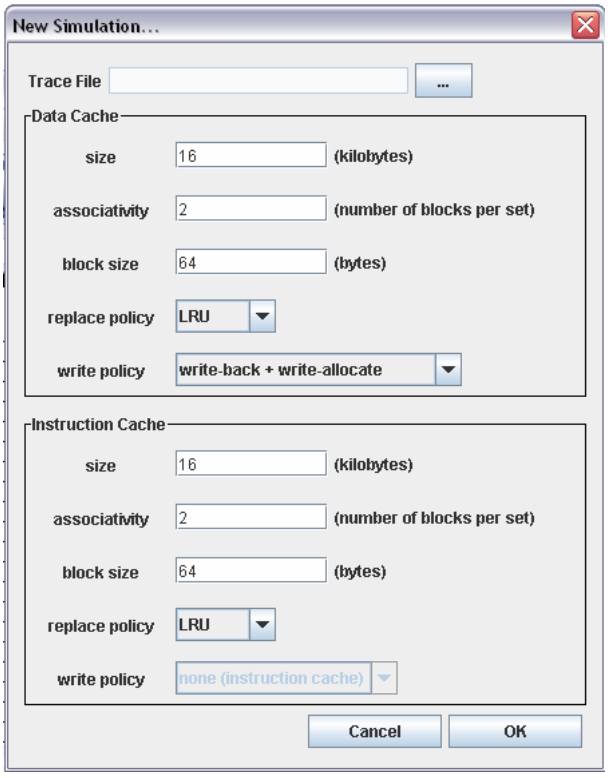


Figura 1: Criação de uma nova simulação para split-cache

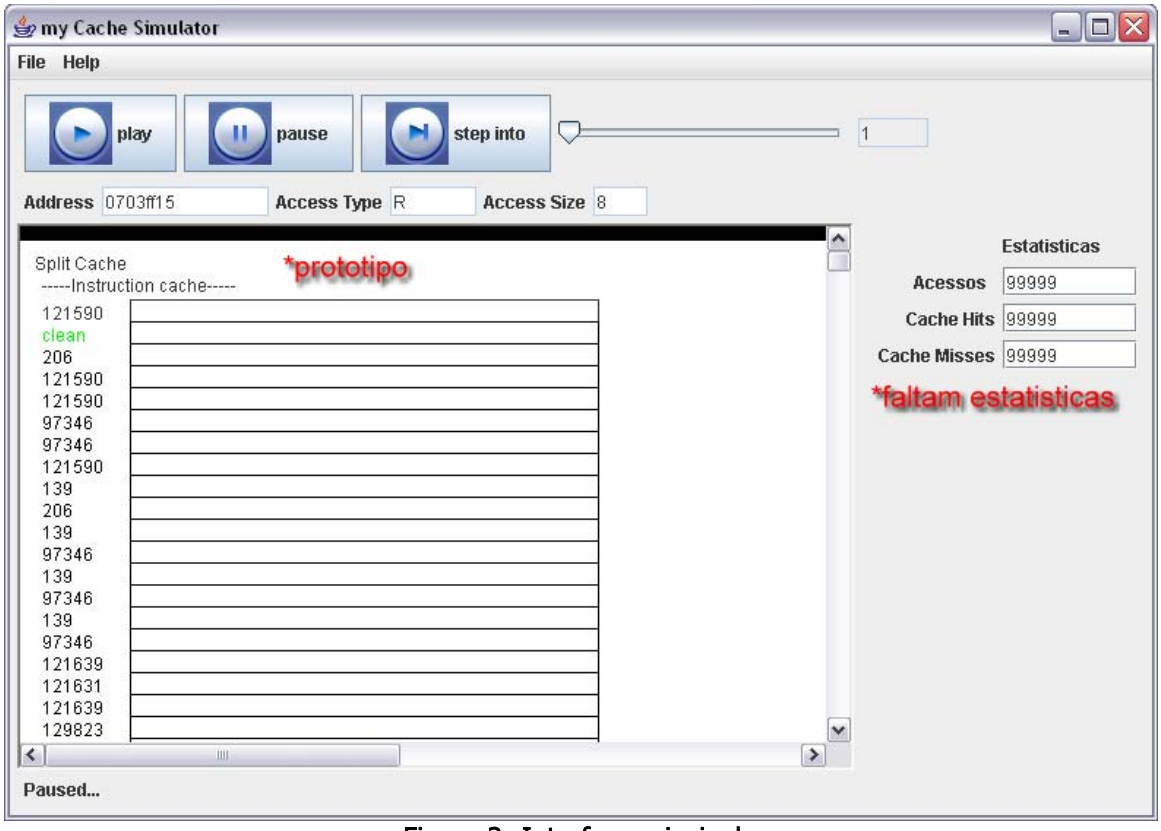


Figura 2: Interface principal

2.2 Arquitectura

A arquitectura deste programa é bastante simples, tem um objecto da classe RunProgram que representa a interface gráfica principal, esta fica encarregue de lançar uma thread que executa a simulação e que por sua vez desenha na interface principal.

A interface principal tem os controlos necessários para pausar, continuar e definir a velocidade da simulação. A thread de simulação simplesmente encontra-se num estado definido pela interface e age conforme o seu estado actual. Se estiver pausada suspende a simulação, se estiver em modo *play* continua a executar ininterruptamente a simulação ou se estiver em modo iterativo (next-step) reproduz um acesso a memória e espera um input da interface gráfica, que é dado pelo utilizador quando carrega no botão play/step-into.

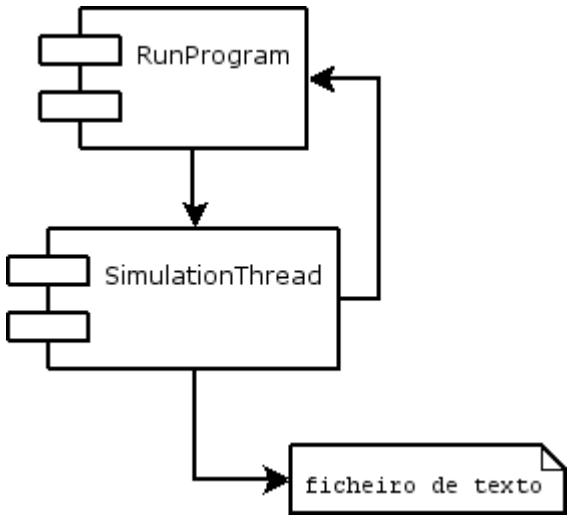


Figura 3: Arquitectura lógica

2.3 Algoritmos

O algoritmo mais complexo usado neste programa é a simulação do acesso a memória. Como se pode ver no diagrama de fluxo da Figura 4 trata-se de um processo complicado sobretudo na parte de substituição do bloco.

A thread de simulação tem um objecto do tipo SplitCache ou UnifiedCache que aceitam acessos a memória representados pela classe MemAccess (ver capitulo sobre detalhes da implementação). A partir da chamada à função doMemAccess o acesso é executado de acordo com o diagrama da Figura 4.

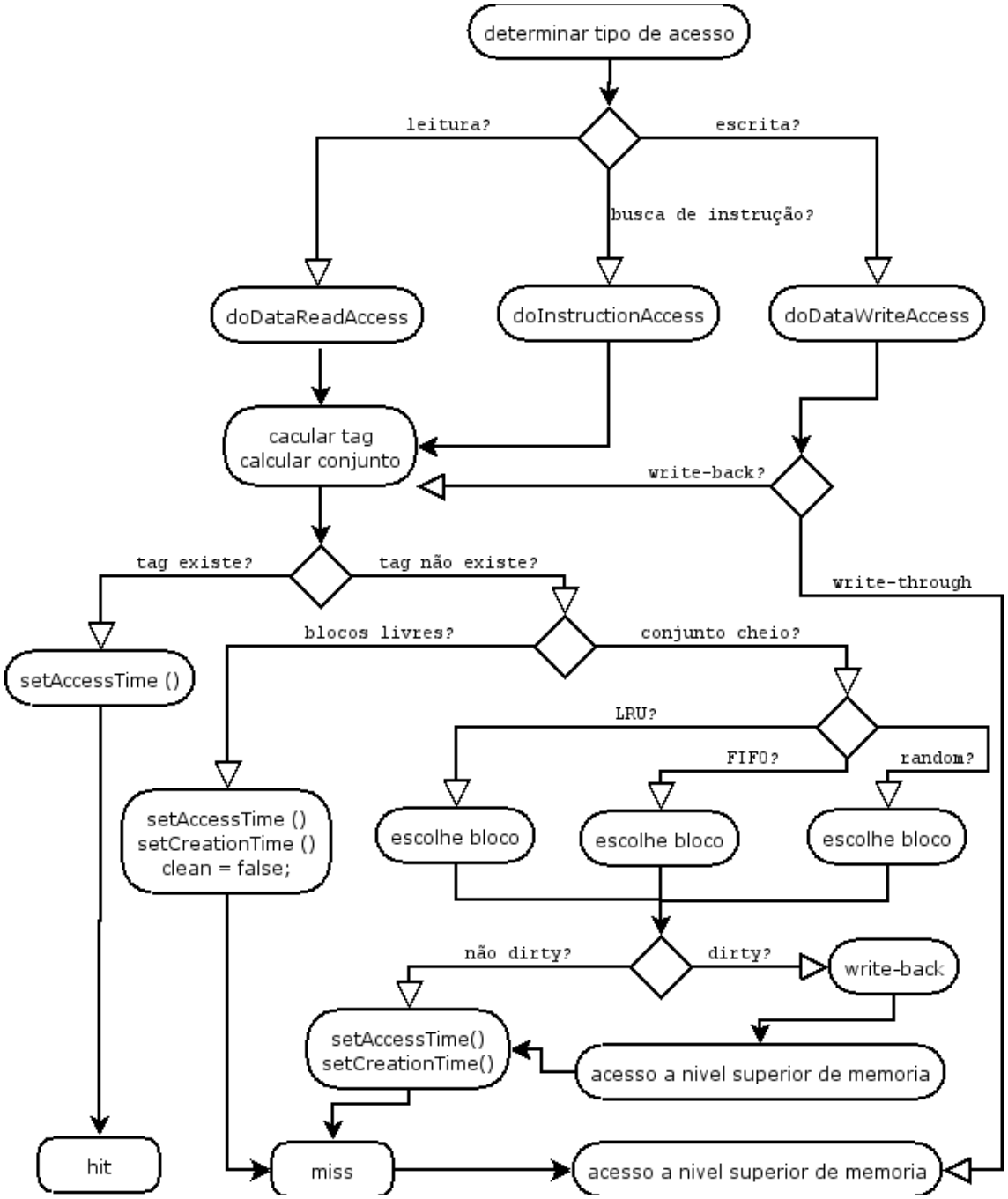


Figura 4: Algoritmo de simulação

O diagrama está bastante fácil de perceber, mas convém mencionar algumas funções utilizadas, setAccessTime e setCreationTime são funções que guardam o tempo corrente em nanosegundos numa variável do bloco.

É possível utilizar a política de substituição LRU para determinar o bloco à mais tempo inutilizado porque sempre que um bloco é utilizado chama-se a função setAccessTime, assim é só comparar e determinar quem tem o setAccessTime menor (mais antigo).

A função setCreationTime guarda o tempo corrente numa variável do bloco quando é usado pela primeira vez (miss), serve para determinar o bloco mais antigo em relação à data da

sua criação, assim é simples de implementar a política de substituição FIFO bastando verificar quem tem o setCreationTime menor (mais antigo).

Estas funções são sempre chamadas independentemente das políticas de substituição/escrita escolhidas, apenas actualizam as variáveis necessárias para depois se conseguir processar uma política de substituição/escrita.

Não está perceptível no diagrama da Figura 4 mas quando existe um acesso de escrita em que a política de escrita é write-back o dirty bit do bloco correspondente é colocado a *true*.

2.4 Detalhes de implementação

A figura seguinte mostra uma versão simplificada das classes, seus atributos e operações. Como se pode ver o código está bastante modular e facilmente compreensível. Existem outras classes de suporte, por exemplo janelas de diálogo, que apesar de terem alguma importância iriam dificultar a compreensão da organização global.

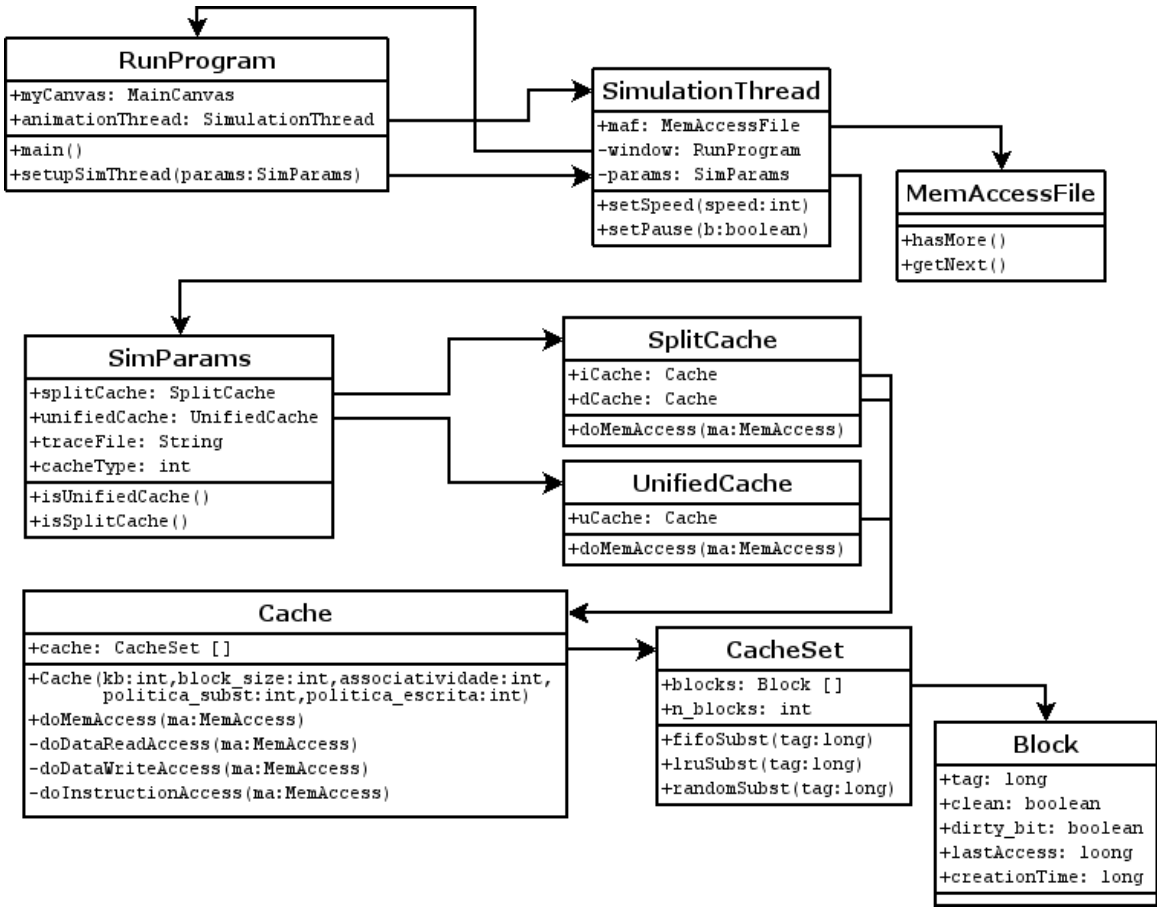


Figura 5: Organização do código em classes

A classe RunProgram utiliza e controla a SimulationThread e por sua vez a SimulationThread actualiza a componente gráfica do programa.

A SimulationThread corre a simulação chamando consecutivamente a função getNext da classe MemAccessFile que devolve o próximo acesso devidamente construído e representado pela classe MemAccess (não representada neste diagrama).

Para realizar o acesso a SimulationThread determina qual o tipo de simulação que está a correr a partir do SimParams e chama a função doMemAccess.

Como se pode ver a cache está organizada num array de CacheSet que representa conjuntos (grupos de blocos) e por sua vez a CacheSet contem um array de blocos que são o elemento base da cache.

Depois de correr o acesso, a SimulationThread chama a função repaint do objecto myCanvas (do tipo MainCanvas) localizado na classe RunProgram. Este objecto tem referência aos parâmetros da simulação (SimParams) e a partir disso consegue buscar os dados necessários à sua renderização.

2.5 Testes e validação

Nesta secção apresenta-se alguns resultados com o intuito de comprovar o funcionamento correcto do programa.

2.5.1 Coerência dos dados

Os primeiros testes realizados são da coerência dos dados, é de esperar que com o aumento da cache a taxa de falhas diminua. É preciso ter cuidado com estes testes porque com o aumento da cache ocorrem mais falhas compulsórias, como o ficheiro de acessos é relativamente pequeno convém evitar este erro utilizando-se valores pequenos para o tamanho da cache (1 a 8 KB são razoáveis).

Estatísticas para os primeiros 10000 acessos, tamanho do bloco 64bytes, associatividade 2, política de substituição LRU e política de escrita write-back:

	Tamanho (KB)	misses	next-level	hits
Split-Cache	1 * 2	4204	5236	5796
	2 * 2	3922	4879	6078
	4 * 2	3797	4719	6203
UnifiedCache	2	4171	5169	5829
	4	3932	4879	6068
	8	3760	4679	6240

Como se pode ver nesta tabela os resultados apresentados nos testes são coerentes. Conforme se vai aumentando a capacidade da cache a taxa de falhas reduz razoavelmente apesar de não se notar grandes melhorias. Este fenómeno é exlicado pelo facto que existe uma

atenuante, essa atenuante são as falhas compulsórias, quanto mais se aumenta a cache maiores são as falhas iniciais. Então será de esperar para testes relativamente mais longos que as diferenças aumentem significativamente.

A relação das taxas de falha entre a cache unificada e cache separada é muito próxima, o que seria de esperar visto que tem o mesmo tamanho. Outra evidencia que aumenta a confiança é o facto de que a soma entre o número de falhas e número de acertos resulta no número total de acessos.

2.5.2 Algoritmos de substituição

Outro teste que se pode realizar é o da verificação da eficácia dos algoritmos de substituição, para este teste realizam-se os mesmos 10000 acessos para uma cache unificada, tamanho 16KB, tamanho do bloco 64 bytes, associatividade 4 e política de escrita write-back:

	Política de substituição	Misses	next-level	Hits
Unified Cache	LRU	3370	4129	6630
	FIFO	3462	4276	6538
	Random	3757	4651	6243

Estes testes mostram o que era esperado, a política de substituição mais eficaz é a LRU, seguida da FIFO e por fim RANDOM. A maneira de funcionamento destas políticas é comprovada empiricamente com auxílio da interface gráfica, para cada acesso pode-se ver o estado das variáveis usadas no cálculo do bloco a substituir. Esses valores mostram a diferença em nanosegundos ao bloco que a política de substituição considera mais recente. Como se trata de um algoritmo simples e após verificação na interface gráfica pode-se concluir que os algoritmos de substituição estão correctos.

2.5.3 Resultados do acesso

Fizeram-se testes unitários para cada operação de acesso e para cada estado possível do bloco escolhido a substituir, os resultados apresentados a seguir mostram que todos os estados foram cobertos e que está de acordo com o funcionamento esperado de uma cache, sendo assim é fácil dizer com relativa confiança que os resultados obtidos serão correctos.

	empty	tag igual		tag diferente	
		dirty	não dirty	dirty	não dirty
I	1 miss 1 next-level	1 hit dirty	1hit	1 miss 1 write-back 2 next-level	1 miss 1 next-level
DR	1 miss 1 next-level	1 hit dirty	1 hit	1 miss 1 write-back 2 next-level	1 miss 1 next-level
DW	1 miss 1 next-level dirty	1 hit dirty	1 hit dirty	1 miss 1 write-back 2 next-level dirty	1 miss 1 next-level dirty

3 Conclusões

Este trabalho apesar de inicialmente parecer simples revelou ser de alguma complexidade devido à flexibilidade imposta na discriminação da arquitectura da cache. Fazer um simulador para um determinado tipo de cache, por exemplo, em que só variava o tamanho ou tamanho do bloco seria algo simples mas dar a hipótese ao utilizador de especificar políticas de substituição, políticas de escrita, associatividade, etc. torna o problema em algo muito mais complexo sobretudo porque é necessário renderizar o esquema da cache.

Todos os requisitos impostos foram cumpridos e ainda foi implementada a política de substituição LRU. A interface gráfica está bem desenhada sendo um programa de fácil utilização. Pode-se dizer que é uma ferramenta interessante como complemento no ensino desta matéria.

Em relação ao desenvolvimento pode-se dizer que o código está bastante modular e é facilmente expansível. A linguagem escolhida para desenvolver este produto foi Java ™ logo é um software que pode correr em qualquer sistema operativo que possua uma Java Virtual Machine.

4 Melhoramentos

Alguns melhoramentos que se podiam fazer em versões futuras podiam ser a inclusão de um menu de configuração que desse hipótese ao utilizador de escolher vários parâmetros da interface, por exemplo, se queria ver as tags em decimal, hexadecimal ou binários, etc. Seriam funcionalidades secundárias que adicionariam algumas opções interessantes mas não muito importantes.

Podia-se incluir mais algumas estatísticas, por exemplo, falhas compulsórias e outras estatísticas que podessem ser interessantes para o programa.

Outro melhoramento a fazer, talvez a inclusão de melhores animações.

5 Referências

5.1 Bibliografia

Acetatos das aulas teóricas de Arquitectura Avançada de Computadores, João Canas Ferreira. Ano lectivo 2005/2006.

5.2 Software

Plataforma de desenvolvimento Eclipse SDK v3.1.1, Sun Microsystems.

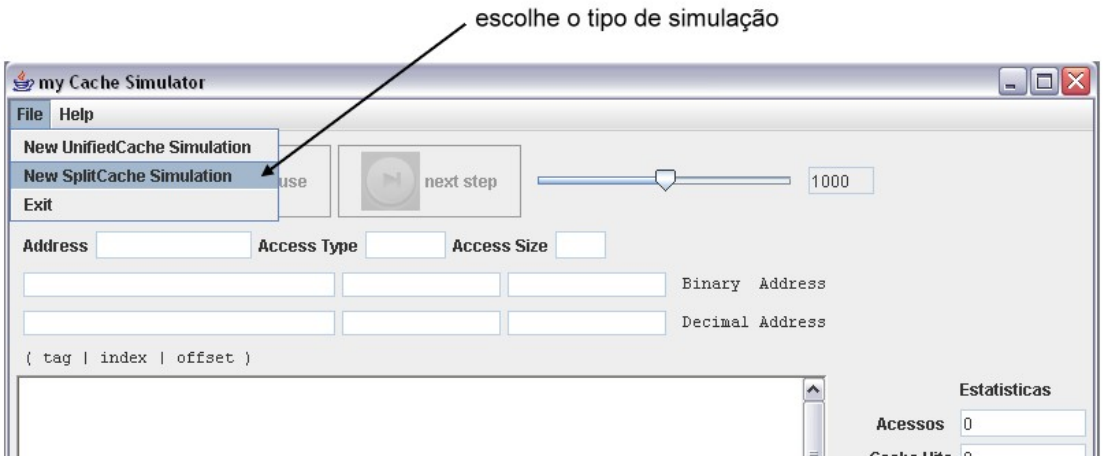
Java ™ Runtime Environment, Standard Edition (1.5.0-b64).

6 Apêndices

6.1 Manual do utilizador

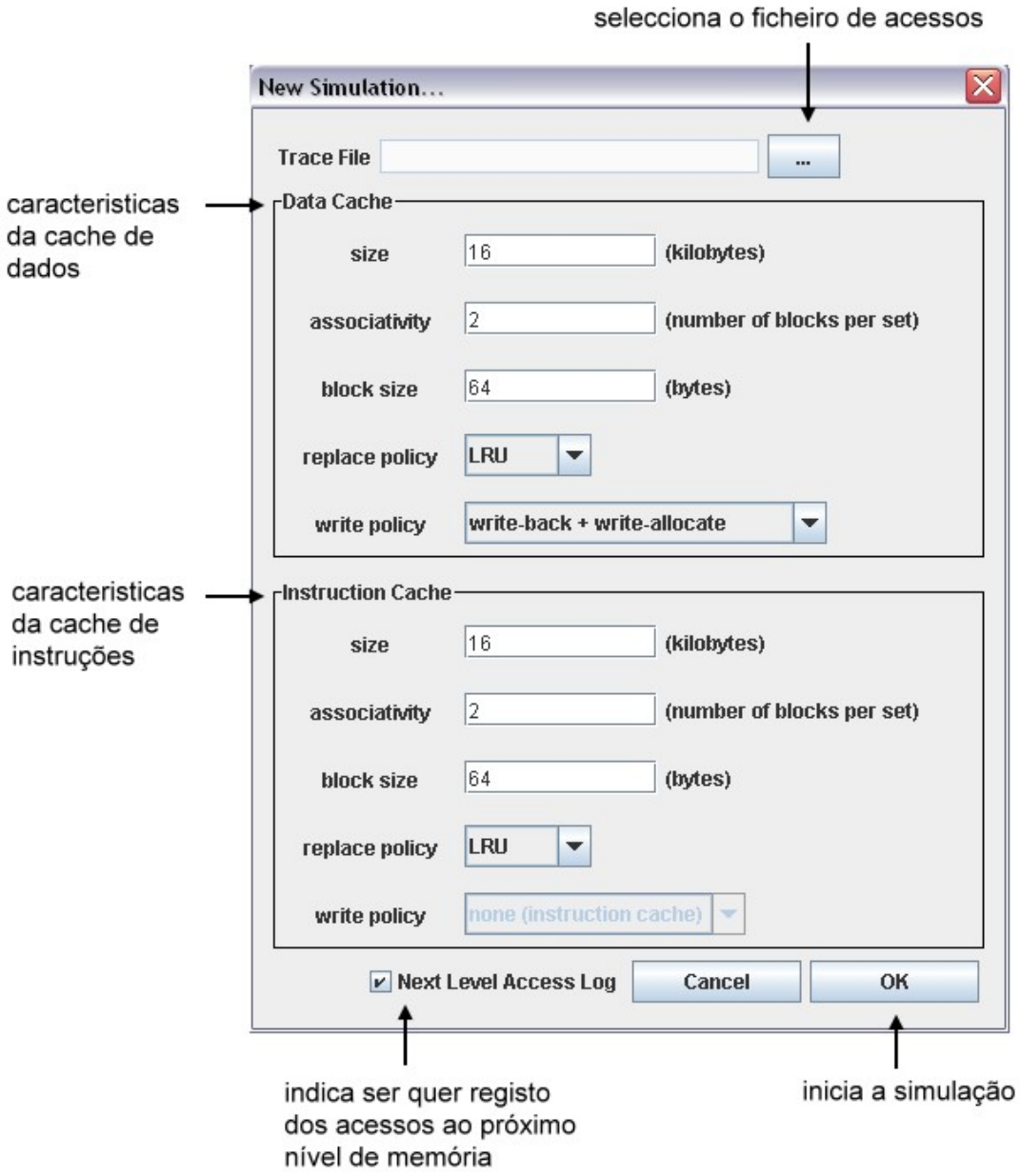
6.1.1 Escolher a simulação

Em primeiro lugar é necessário escolher a simulação que se pretende fazer, tanto pode ser uma simulação para um cache unificada como para uma cache separada. Antes de criar a simulação os controles estão desactivados.



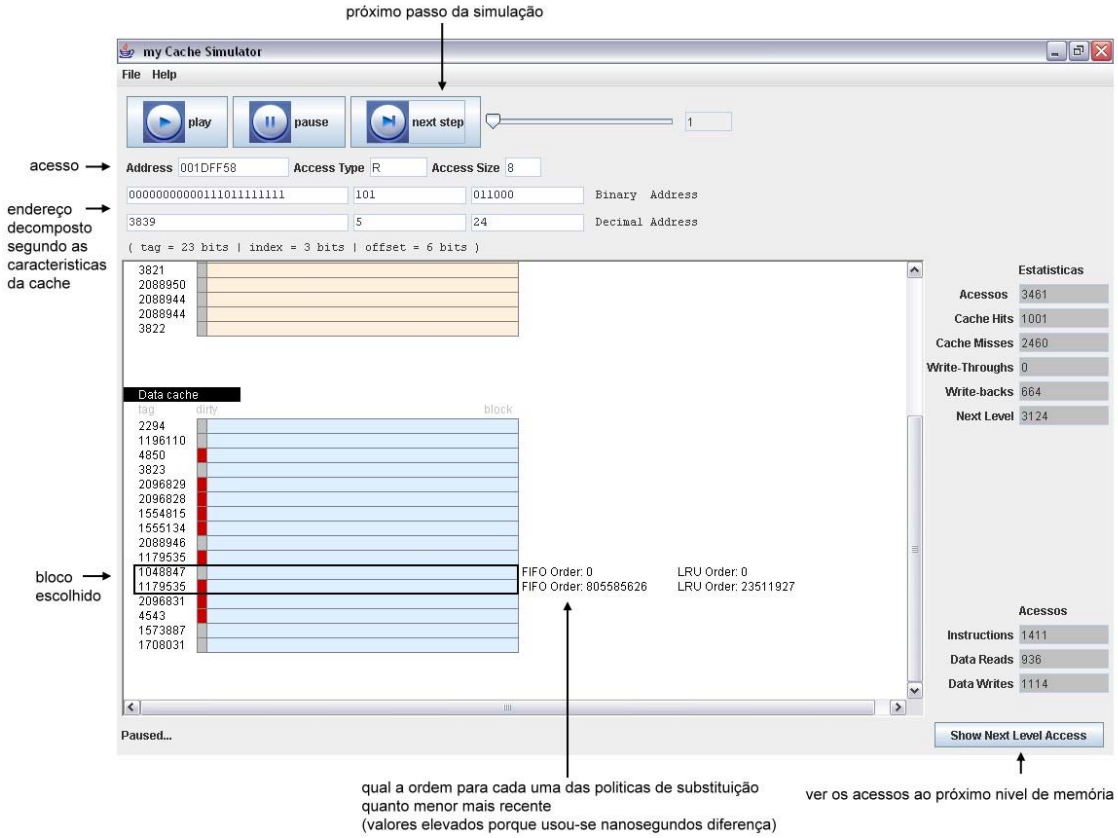
6.1.2 Especificar a cache

Despois de escolher o tipo de simulação o utilizador preenche as características da cache e diz qual o ficheiro de acessos necessário à simulação.



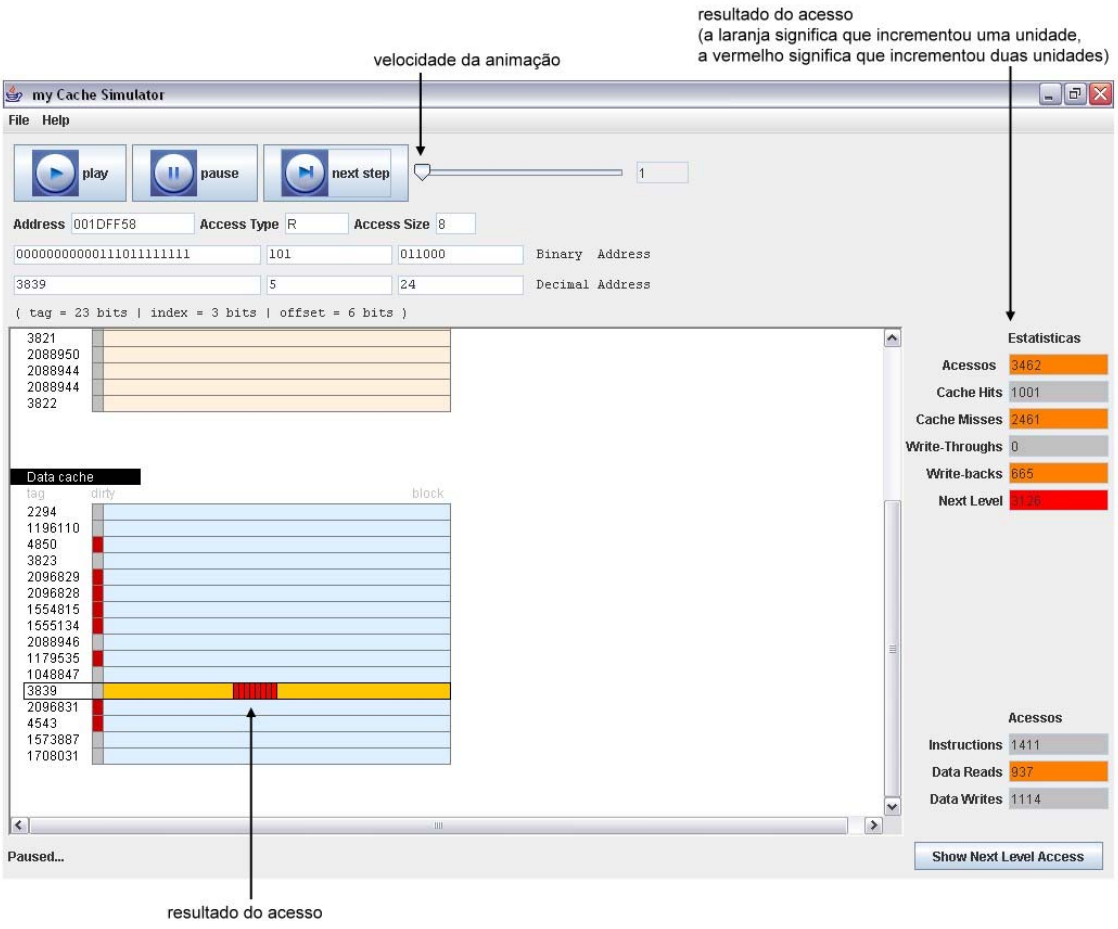
6.1.3 Correr a simulação

Depois de escolher as características da cache dá-se início à simulação. O utilizador tem três botões de controlo, play, pause e next-step. O normal será usar o next-step para ver qual o acesso e qual o conjunto escolhido para esse acesso, também pode ver o endereço decomposto segundo as características da cache tanto em binário como em decimal.



6.1.4 Resultado de um acesso

Depois de executar um acesso o utilizador pode ver o resultado desse acesso. Na parte das estatísticas pode ver a laranja os campos que foram incrementados uma unidade e os que estão a cinzento mantiveram-se iguais. Em algumas situações por exemplo, quando há um bloco que está dirty e faz-se um acesso de leitura que não tem a mesma tag acontece que existe dois acessos ao próximo nível, quando isso acontece o campo em vez de aparecer laranja aparece a vermelho.

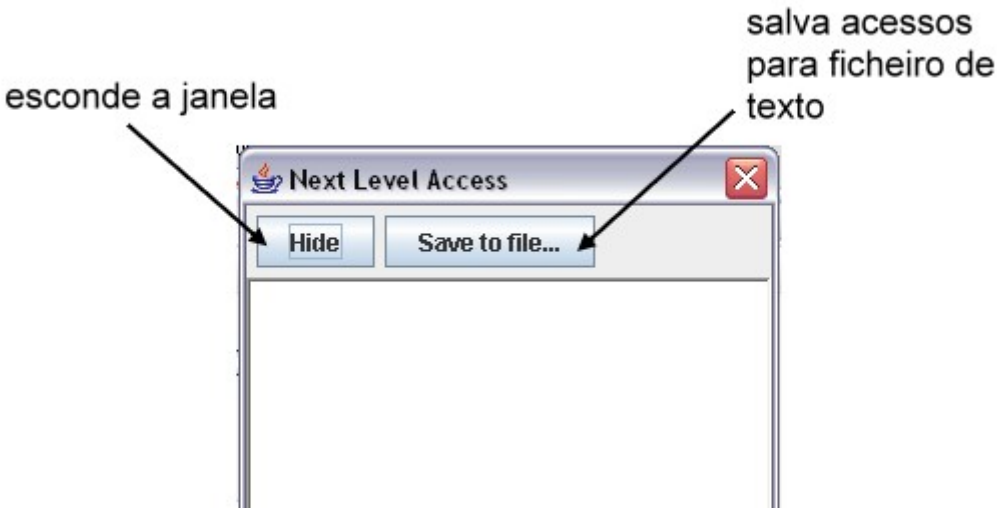


Também é possível ver em que parte do bloco o acesso diz respeito e qual o tipo de acesso feito.

Outra opção é a definição da velocidade de animação, esta velocidade só interessa se a simulação estiver em modo play.

6.1.5 Acessos ao próximo nível de memória

A interface dispõe de um botão situado no canto inferior direito que permite ver os acessos ao próximo nível de memória.



Esta janela só mostra os últimos 100 acessos por uma questão de eficiência, mas ao guardar para um ficheiro são escritos todos os acessos ao próximo nível de memória.

6.2 Código

As classes presentes neste apêndice são as seguintes:

- AboutDialog
- Block
- Cache
- CacheSet
- MainCanvas
- MemAccess
- MemAccessFile
- MyPoint
- NextLevelAccessDialog
- RunProgram
- SimParams
- SimulationThread
- SplitCache
- SplitCacheDialog
- UnifiedCache
- UnifiedCacheDialog
- WarningDialog

6.2.1 AboutDialog

Esta classe serve unicamente para mostrar informações sobre o trabalho realizado e sobre o autor. A maior parte do código foi gerado automaticamente pelo ambiente de desenvolvimento.

```
package cacheSim;

import java.awt.BorderLayout;


import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JDialog;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;

/**
 * Dialogo sobre o autor do programa
 * @author Manuel
 *
 */
public class AboutDialog extends JDialog {

	public static final long serialVersionUID = 0;
	private JPanel jContentPane = null;
	private JPanel jPanelCenter = null;
	private JButton jButtonOk = null;
	private JPanel jPanel1 = null;
	private JLabel jLabel = null;
	private JTextArea jTextArea = null;
	private JScrollPane jScrollPane = null;

	/**
	 * This is the default constructor
	 */
	public AboutDialog(JFrame frame) {
		super(frame, "Teste", true);
		initialize();
		this.setLocationRelativeTo(null);
	}

	/**
	 * This method initializes this
	 *
	 * @return void
	 */
	private void initialize() {
		this.setSize(356, 331);
		this.setContentPane(getJContentPane());
		this.setTitle("About...");
	}

	/**
	 * This method initializes jContentPane
	 *
	 * @return javax.swing.JPanel
	 */
	private JPanel getJContentPane() {
		if (jContentPane == null) {
			jContentPane = new JPanel();
			jContentPane.setLayout(new BorderLayout());
			jContentPane.add(getJPanelCenter(), java.awt.BorderLayout.SOUTH);
			jContentPane.add(getJPanel1(), java.awt.BorderLayout.NORTH);
			jContentPane.add(getJScrollPane(), java.awt.BorderLayout.CENTER);
		}
		return jContentPane;
	}
```

```
	/**
	 * This method initializes jPanelCenter
	 *
	 * @return javax.swing.JPanel
	 */
	private JPanel getJPanelCenter() {
		if (jPanelCenter == null) {
			jPanelCenter = new JPanel();
			jPanelCenter.setBackground(new java.awt.Color(204,51,0));
			jPanelCenter.add(getJButtonOk(), null);
		}
		return jPanelCenter;
	}

	/**
	 * This method initializes jButtonOk
	 *
	 * @return javax.swing.JButton
	 */
	private JButton getJButtonOk() {
		if (jButtonOk == null) {
			jButtonOk = new JButton();
			jButtonOk.setText("OK");
			jButtonOk.addActionListener(new java.awt.event.ActionListener() {
				public void actionPerformed(java.awt.event.ActionEvent e) {
					AboutDialog.this.setVisible(false);
				}
			});

		}
		return jButtonOk;
	}

	/**
	 * This method initializes jPanel1
	 *
	 * @return javax.swing.JPanel
	 */
	private JPanel getJPanel1() {
		if (jPanel1 == null) {
			jLabel = new JLabel();
			jLabel.setText("Arquitectura Avançada de Computadores 2005/2006");
			jLabel.setForeground(java.awt.Color.white);
			jPanel1 = new JPanel();
			jPanel1.setBackground(new java.awt.Color(204,51,0));
			jPanel1.setPreferredSize(new java.awt.Dimension(306,36));
			jPanel1.add(jLabel, null);
		}
		return jPanel1;
	}

	/**
	 * This method initializes jTextArea
	 *
	 * @return javax.swing.JTextArea
	 */
	private JTextArea getJTextArea() {
		if (jTextArea == null) {
			jTextArea = new JTextArea();


			jTextArea.setText(    "FEUP 2005\n" +
					"Este programa foi desenvolvido no âmbito da " +
					"cadeira de Arquitectura Avançada de Computadores,
ano lectivo 2005/2006.\n" +
					"\nDesenvolvido por: "+
					"\nManuel Faria de Azevedo Maia" +
					"\nmail: ei00128@fe.up.pt" +
					"\nhttp://www.fe.up.pt/~ei00128/index.html" +
					"\n\nDocente da disciplina: " +
					"\nJoão Canas Ferreira" +
					"\nmail: jcf@fe.up.pt"
			);
			jTextArea.setLineWrap(true);
			jTextArea.setWrapStyleWord(true);
			jTextArea.setBounds(new java.awt.Rectangle(5,5,300,96));
			jTextArea.setBackground(java.awt.SystemColor.text);
```

```
                }
                return jTextArea;
        }

        /**
         * This method initializes jScrollPane
         *
         * @return javax.swing.JScrollPane
         */
        private JScrollPane getJScrollPane() {
                if (jScrollPane == null) {
                        jScrollPane = new JScrollPane();

        jScrollPane.setBorder(javax.swing.BorderFactory.createEmptyBorder(5,5,5,5));
                        jScrollPane.setBackground(java.awt.SystemColor.text);
                        jScrollPane.setViewportView(getJTextArea());
                }
                return jScrollPane;
        }

}  //  @jve:decl-index=0:visual-constraint="10,10"
```

6.2.2 Block

Esta classe representa um bloco de cache, tem várias variáveis de estado como dirty, empty, tag, etc. Tem, também, variáveis de suporte aos algoritmos de substituição e escrita, como a ultima vez que foi usado (LRU) ou quando foi criado (FIFO). Na situação "foi criado" significa que existiu uma substituição ou escrita, enquanto que "foi usado" engloba as situações anteriores e essa variável também é actualizada quando existe um hit.

```
package cacheSim;

/**
 * Representa o bloco de uma cache.
 * Tem as variaveis sobre o estado do bloco (dirty, empty, tag, etc.)
 * e variáveis de suporte aos algoritmos de substituição e escrita
 * @author Manuel
 *
 */
public class Block {

	//dirty_bit se o bloco de cache foi alterado e não foi propragado para memória
	public boolean dirty_bit;
	//se o bloco está limpo, (ainda n foi usado)
	public boolean clean;

	public long tag;
	private int block_size;

	//representa o ultimo acesso feito que resultou num dirty state..
	public MemAccess lastMemAccess;
	public MemAccess lastDirtyMemAccess;

	private long lastAccess; //variavel para a política de susbstituição LRU

	private long creationTime; //variavel para a política de substituição FIFO

	public Block(int block_size){
		this.dirty_bit = false;
		this.clean = true;
		this.tag = 0;
		this.block_size = block_size;
		this.lastAccess = System.nanoTime();//System.currentTimeMillis();
		this.creationTime = System.nanoTime();
	}



	public boolean isDirty(){
		return dirty_bit;
	}

	public long getFIFOindex(){
		return this.creationTime;
	}

	public void setAccessTime(){
		this.lastAccess = System.nanoTime();
	}

	public void setCreationTime(){
		this.creationTime = System.nanoTime();
	}

	public long getLastAccessTime(){
		return this.lastAccess;
	}

	public boolean isClean(){
		return clean;
```

```
	}

	public int getBlockSize(){
		return this.block_size;
	}
	public long getTag(){
		return this.tag;
	}

	public boolean hasTag(long tag){
		if(clean)
			return false;
		if(this.tag == tag)
			return true;
		else
			return false;
	}


}
```

6.2.3 Cache

Esta classe é uma das mais importantes neste programa, representa uma cache genérica, tanto pode ser usada para simular uma cache unicamente de instruções, como uma cache de dados ou ambas.

Tem um array de conjuntos (um array de objectos do tipo CacheSet), esse array representa os vários conjuntos que existem na cache.

Tem funções para executar acessos a memória e outros para buscar informações relativas à cache.

```
package cacheSim;

import java.lang.Exception;

/**
 * Representa uma cache de leitura/escrita
 * Tem um um array de CacheSet que simula os vários conjuntos da cache.
 * Tem funções para fazer acessos a memória, e outros para buscar informação da cache
 * @author Manuel
 *
 */
public class Cache {


	/**
	 * Politica de substituição FIFO
	 */
	public static final int S_FIFO = 100;

	/**
	 * Politica de substituição RANDOM
	 */
	public static final int S_RANDOM = 101;

	/**
	 * Politica de substituição ASSOCIATIVIDADE_COMPLETA
	 */
	public static final int S_LRU = 102;

	/**
	 * Politica de escrita write-through + write-allocate
	 */
	public static final int WR_THROUGH_NO_ALLOCATE = 0;

	/**
	 * Politica de escrita write-back + write-no-allocate
	 */
	public static final int WR_BACK_ALLOCATE = 1;

	/**
	 * Não existe politica de escrita, cache de instruções
	 */
	public static final int WR_INSTR_CACHE = 2;

	public static final int DIRECT_MAPPING = 0;

	//propriedades da cache
	private int size;
	private int blocks;
	private int block_size;
	private int block_size_bits;
	private int index_size_bits;
	private int associatividade;
	private int substituicao;
	private int escrita; //politica de escrita
	private int sets;
```

```
	protected CacheSet [] cache;

	//estatisticas
	private long n_hits;
	private long n_misses;
	private long n_access;
	private long n_instr_access;
	private long n_dataread_access;
	private long n_datawrite_access;
	private long n_write_backs;
	private long n_write_throughs;
	private long n_access_to_next_level;

	private boolean falhas_compulsorias; //true conta com as falhas compulsorias,
false não contabiliza as falhas compulsorias

	private int lastUsedSetIndex;

	private MemAccess [] nextLevelAccessArray;

	/**
	 * Construtor mais usado para criar uma cache
	 * @param kb - tamanho da cache (em kilobytes)
	 * @param block_size - tamanho de bloco da cache (em bytes)
	 * @param associatividade - a associatividade da cache em multiplos de 2 (ou 1)
	 * @param substituicao - politica de substituição
	 * @param escrita - politica de escrita
	 * @throws CacheFormatException
	 */
	public Cache(int kb, int block_size, int associatividade, final int substituicao,
final int escrita)
	throws CacheFormatException
	{
		this.falhas_compulsorias = true;
		int size = (kb*1024);
		int blocks = size/block_size;
		makeCache(size, blocks, block_size, associatividade, substituicao,
escrita);

	}

	/**
	 * Pode contabilizar ou não falhas compulsórias
	 * @param kb
	 * @param block_size
	 * @param associatividade
	 * @param substituicao
	 * @param escrita
	 * @param falhas_compulsorias - true funciona como se não tivesse argumento
	 * @throws CacheFormatException
	 */
	public Cache(int kb, int block_size, int associatividade, final int substituicao,
final int escrita, boolean falhas_compulsorias)
	throws CacheFormatException
	{
		this.falhas_compulsorias = falhas_compulsorias;
		int size = (kb*1024);
		int blocks = size/block_size;
		makeCache(size, blocks, block_size, associatividade, substituicao,
escrita);
	}


	/*****
	 * Desaconselhado o uso deste construtor, existe para alguma compatibilidade com
versões anteriores
	 * @param size - tamanho da cache = blocks*block_size
	 * @param blocks - numero de blocos
	 * @param block_size tamanho do bloco em nº de bytes
	 * @param associatividade associatividade ou DIRECT_MAPPING
	 * @param substituicao politica de substituição (WR_...)
	 * @param escrita politica de escrita em memória (S_...)
	 * @throws CacheFormatException propriedades mal construidas
	 *
```

```
        public Cache(int size, int blocks , int block_size, int associatividade,int
substituicao, int escrita)
        throws CacheFormatException
        {
        makeCache(size, blocks, block_size, associatividade, substituicao, escrita );
        }*/
       private void makeCache(int size, int blocks , int block_size, int
associatividade,int substituicao, int escrita)
       throws CacheFormatException
       {

              int i;
              this.size = size;
              this.blocks = blocks;
              this.block_size = block_size;
              this.associatividade = associatividade;
              this.substituicao = substituicao;
              this.escrita = escrita;


              //verificações das propriedades da cache
              if(size != (blocks*block_size))
                     throw new Cache.CacheFormatException("Cache size different from
block_size*blocks");

              if( ( blocks % associatividade ) != 0)
                     throw new Cache.CacheFormatException("Number of sets incompatible
with number of blocks");

              if((block_size % 2) != 0)
                     throw new Cache.CacheFormatException("Block size must be
2,4,8,16,...");

              if(associatividade == DIRECT_MAPPING)
                     this.associatividade = blocks;

              if(size < 1 || blocks < 1 || block_size < 1 || associatividade < 0 ||
substituicao < 0 || escrita < 0)
                     throw new Cache.CacheFormatException("Wrong usage of arguments");
              //if(substituicao > 3 || escrita > 3)
              //     throw new Cache.CacheFormatException("Wrong usage of arguments");
              if(associatividade > blocks)
                     throw new Cache.CacheFormatException("Associatividade
impossivel");

              this.sets = (int)(blocks/associatividade);
              this.cache = new CacheSet[this.sets];

              for(i=0; i<sets; i++){
                     cache[i] = new CacheSet(associatividade,block_size);

              }


              //inicializando estatisticas
              this.n_hits = 0;
              this.n_misses = 0;
              this.n_access = 0;
              this.n_dataread_access = 0;
              this.n_datawrite_access = 0;
              this.n_instr_access = 0;
              this.n_access_to_next_level = 0;

              //calcular o numero de bits para o tamanho do bloco
              block_size_bits=0;
              i = block_size;
              while(i!= 1){
                     i = i >> 1;
              block_size_bits++;
              }

              //calcula o numero de bits necessários para o index
              i=sets;
              index_size_bits = 0;
              while(i!= 1){
                     i = i >> 1;
              index_size_bits++;
```

```
		}


		System.out.println("BlockSize(bits)= "+this.block_size_bits+"
IndexSize(bits)= "+index_size_bits);


	}

	/**
	 * executa um acesso a memória
	 * @param ma - Acessoa  memória do tipo MemAcess
	 * @throws CacheFormatException
	 */
	public void doMemAccess(MemAccess memAccess)
	throws CacheFormatException
	{
		int ret=0;
		MemAccess ma = memAccess.getCopy();


		//servia para verficar se o bloco saia
		//for(int i=0; i<ma.getSize(); i++){
		//ma.incAddress();
		//}
		//verifica se é uma cache de instruções, e se for só pode ser usada para
buscar instruções
		if(escrita == WR_INSTR_CACHE && (!ma.isInstructionAccess()))
			throw new CacheFormatException("Instruction Cache Only!");
		else{
			//CUIDADOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOOO
			if(ma.isDataReadAccess()){
				ret += doDataReadAccess(ma);

			}else if(ma.isDataWriteAccess()){
				ret += doDataWriteAccess(ma);

			}else if(ma.isInstructionAccess()){
				ret += doInstructionAccess(ma);

			}else{
				System.out.println("NÃO SEI QUAL É O
TIPO!!!!!!!!!!!!!!!!!!!!!!!!!!"+ma);
			}
		}



		//contabiliza o numero de acessos de cada tipo
		this.n_access++;
		if(ma.isDataReadAccess()){
			this.n_dataread_access++;
			if(ret==0){
				this.n_hits++;
			}
			if(ret == 1){
				this.n_access_to_next_level++;
				this.n_misses++;
			}
			if(ret == 2){
				this.n_write_backs++;
				this.n_misses++;
				this.n_access_to_next_level+=2; //1 pelo miss + 1 pelo
write-back

			}


		}else if(ma.isDataWriteAccess()){
			this.n_datawrite_access++;

			if(this.escrita == Cache.WR_BACK_ALLOCATE){
				if(ret == 2){
					this.n_misses++;
					this.n_write_backs++;
					this.n_access_to_next_level+=2;
```

```
                        }
                        if(ret == 1){
                                this.n_misses++;

                                this.n_access_to_next_level++;
                        }
                        if(ret == 0){
                                this.n_hits++; //verificar isto com o professor
                                //escritas na cache causam hits ou misses ou nenhum
dos dois?
                        }

                }else{
                        if(this.escrita == Cache.WR_THROUGH_NO_ALLOCATE){
                                this.n_write_throughs++;
                                this.n_access_to_next_level++;
                        }
                }


        }else if(ma.isInstructionAccess()){
                this.n_instr_access++;

                if(ret==0){
                        this.n_hits++;
                }
                if(ret == 1){
                        this.n_access_to_next_level++;
                        this.n_misses++;
                }
                if(ret == 2){
                        this.n_access_to_next_level += 2;
                        this.n_misses++; //verificar isto com o prof
                        this.n_write_backs++;

                }


        }


        //regista os ultimos acessos a memória
        //causou um miss e um write-back
        if(ret == 2){
                this.nextLevelAccessArray = new MemAccess [2];
                this.nextLevelAccessArray[0] =
cache[this.lastUsedSetIndex].getLastAccessedBlock().lastDirtyMemAccess.getCopy();
                this.nextLevelAccessArray[1] = ma.getCopy();


        }
        //causou só um miss
        if(ret == 1){
                this.nextLevelAccessArray = new MemAccess [1];
                this.nextLevelAccessArray[0] = ma.getCopy();
        }

        if(ret == 0)
                this.nextLevelAccessArray = null;

        //se o bloco ficou dirty (data-write access) entao memoriza o acesso que
fez-lo ficar dirty
        if(ma.isDataWriteAccess() &&
cache[this.lastUsedSetIndex].getLastAccessedBlock().isDirty())

    cache[this.lastUsedSetIndex].getLastAccessedBlock().lastDirtyMemAccess =
ma.getCopy();

    }



    /**
     * Não altera o conteudo da cache apenas determina a tag para um determinado
acesso
     * @param ma
     * @return
```

```
         */
        public long getTag(MemAccess ma){
                long tag = ma.getAddress() >>
(this.index_size_bits+this.block_size_bits);
                return tag;
        }

        public int getTagBits(){
                return 32 - this.block_size_bits - this.index_size_bits;
        }

        public int getIndexBits(){
                return this.index_size_bits;
        }

        public int getOffsetBits(){
                return this.block_size_bits;
        }

        public CacheSet getCacheSet(int index){
                return this.cache[index];
        }



        public long getOffset(MemAccess ma){
                long a = ma.getAddress();
                a = a >> this.block_size_bits;
                a = a << this.block_size_bits;

                long offset = ma.getAddress() - a;
                return offset;
        }
        public int getIndex(MemAccess ma){
                long a = ma.getAddress();
                a = a >> this.block_size_bits;
                int set =  (int)(a % this.sets);
                return set;
        }

        private int doDataReadAccess(MemAccess ma){
                long a = ma.getAddress();
                a = a >> this.block_size_bits;
                int set =  (int)(a % this.sets);
                this.lastUsedSetIndex = set;

                long tag = ma.getAddress() >>
(this.index_size_bits+this.block_size_bits);
                int acc = cache[set].doDataReadAccess(tag, this.substituicao);

                return acc;

        }

        /**
         * Um acesso de escrita nunca provoca um hit
         * Pode no máximo proovar um write-throug ou write-back e deve ser assim
         * que deve ser contabilizado para estatisticas
         * @param ma - acesso a memória do tipo write
         * @return
         */
        private int doDataWriteAccess(MemAccess ma){
                long a = ma.getAddress();
                a = a >> this.block_size_bits;
                int set =  (int)(a % this.sets);
                this.lastUsedSetIndex = set;

                long tag = ma.getAddress() >>
(this.index_size_bits+this.block_size_bits);
                int acc = cache[set].doDataWriteAccess(tag, this.substituicao,
this.escrita);

                return acc;

        }

        private int doInstructionAccess(MemAccess ma){
```

```
                long a = ma.getAddress();
                a = a >> this.block_size_bits;

                //calculo do conjunto
                int set =  (int)(a % this.sets);
                this.lastUsedSetIndex = set;
                /*calculo alternativo a ver se da os mesmos resultados~
                 long i = a; //32 bits a 1
                 i = i >> (this.index_size_bits);
                 i = i << this.index_size_bits;
                 i = a - i;
                 int set = (int)i;*/ //resulta perfeitamente ambos as maneiras dão o
mesmo resultado

                //fim do calculo alternativo

                long tag = ma.getAddress() >>
(this.index_size_bits+this.block_size_bits);
                int acc = cache[set].doInstructionAccess(tag, this.substituicao);

                return acc;

                /*
                 this.n_access++;
                 this.n_instr_access++;
                 if(acc==0){
                 this.n_hits++;
                 }else{
                 this.n_misses++;
                 }
                 */
                //System.out.println("Vai para o set "+set+"(n_sets="+this.sets+")");
                //System.out.println("Access="+n_access+"
,NInstrAccess="+n_instr_access+" ,NHits="+n_hits+" ,NMisses="+n_misses+", Falhas (por
milhar de instr.)="+((n_misses*1000/n_access)));
        }

        public void printStatistcs(){
                System.out.println("\n-- Cache Info --");
                System.out.println("Size = "+this.size/1024+"kB");
                System.out.println("Block Size = "+this.block_size+"bytes");
                System.out.println("Associatividade = "+this.associatividade+", Subst.
Policy = "+this.replacePolicyToString());
                if(this.falhas_compulsorias)
                        System.out.println("\n-- Cache Statistics (with start misses) --
");
                else
                        System.out.println("\n-- Cache Statistics (without start misses) -
-");
                System.out.println("Access = "+n_access);
                System.out.println("NInstrAccess = "+n_instr_access);
                System.out.println("NDataReadAccess = "+this.n_dataread_access);
                System.out.println("NDataWriteAccess = "+this.n_datawrite_access);
                System.out.println("NWriteBacks = "+this.n_write_backs);
                System.out.println("NWriteThroughs = "+this.n_write_throughs);
                System.out.println("NAccessToNextLevel = "+this.n_access_to_next_level);
                System.out.println("NHits = "+n_hits);
                System.out.println("NMisses = "+n_misses);
                System.out.println("Misses = "+((n_misses*1000/n_access))+"/1000
("+((n_misses*100/(double)n_access))+"%)");
        }

        public long getNumNextLevelAccess(){
                return this.n_access_to_next_level;
        }
        public long getNumWriteThroughs(){
                return this.n_write_throughs;
        }
        public long getNumWriteBacks(){
                return this.n_write_backs;
        }

        public long getNumIAccess(){
                return this.n_instr_access;
        }
        public long getNumDRAccess(){
                return this.n_dataread_access;
```

```
	}
	public long getNumDWAccess(){
		return this.n_datawrite_access;
	}

	public long getNumAccess(){
		return this.n_access;
	}
	public long getNumHits(){
		return this.n_hits;
	}
	public long getNumMisses(){
		return this.n_misses;
	}

	public String replacePolicyToString(){
		switch(this.substituicao){
		case Cache.S_FIFO:
			return "FIFO";
		case Cache.S_RANDOM:
			return "RANDOM";
		case Cache.S_LRU:
			return "LRU";
		default:
			return "direct-mapping";
		}
	}

	public String writePolicyToString(){
		switch(this.escrita){
		case Cache.WR_BACK_ALLOCATE:
			return "write-back + write-no-allocate";
		case Cache.WR_THROUGH_NO_ALLOCATE:
			return "write-through + write-allocate";
		default:
			return "no-write-policy";
		}
	}

	public int getNumBlocks(){
		return this.blocks;
	}
	public int getNumSets(){
		return this.sets;
	}

	public int getBlockSize(){
		return this.block_size;
	}

	public Block getBlock(int b){
		int index = b/this.associatividade;
		int bindex = b%this.associatividade;
		return cache[index].getBlock(bindex);
	}

	public boolean isInstrCache(){
		if(escrita == Cache.WR_INSTR_CACHE)
			return true;
		else
			return false;
	}

	public String associatividadeToString(){
		if(associatividade == blocks)
			return "direct-mapping";
		else
			return ""+this.associatividade;
	}

	public String toString(){
		return "Cache: size="+size+"(bytes),
associativity="+associatividadeToString()+"
("+(this.blocks/this.associatividade)+"sets), blocks="+this.blocks+", block
size="+this.block_size+"(bytes), write policy="+writePolicyToString()+", replace
policy="+replacePolicyToString();
	}
```

```
	//é preciso dizer qual o acesso a memória para determinar o conjunto que foi
acedido em ultimo lugar
	//so se pode chamar esta função depois de fazer o acesso a memória
	public int getLastAccessBlockIndex(){



		int index =  cache[this.lastUsedSetIndex].getLastAccessedBlockIndex();
		return (lastUsedSetIndex * this.associatividade)+index;

	}

	//determina o set a ser escolhido atraves do acesso a memória
	public int getSetIndex(MemAccess ma){
		long a = ma.getAddress();
		a = a >> this.block_size_bits;
		return (int)(a % this.sets);
	}

	public int getAssociatividade(){
		return this.associatividade;
	}


	//pode retornar null caso não haja acesso ao próximo nivel de memória
	public MemAccess [] getNextLevelAccess(){
		return this.nextLevelAccessArray;
	}




	public static class CacheFormatException
	extends Exception
	{
		static final long serialVersionUID = 0;

		public CacheFormatException(String s){
			super(s);
		}
	}

}
```

6.2.4 CacheSet

Esta classe representa um conjunto da cache, a cache é constituída por um array destes objectos. Por sua vez esta classe tem um array de blocos.

Nesta classe estão os algoritmos de substituição e escrita, esta era a opção mais lógica visto esses algoritmos escolhem o bloco para substituir/escrever dentro do conjunto que é representado por esta classe.

```
package cacheSim;

/**
 * Representa um conjunto da cache.
 * Aqui decide-se qual o bloco a substituir, etc...
 * Tem um array de Blocks que representa os blocos deste conjunto.
 * @author Manuel
 *
 */
public class CacheSet {
	private int n_blocks;
	private Block [] blocks;
	private int block_size;
	//podia contabilizar as falhas compulsorias
	//private boolean falhas_compulsorias;

	//flag que indica o index do ultimo bloco que foi acedido
	//serve pra efeitos de desenho, etc...
	private int lastAccessedBlockIndex;

	public CacheSet(int n_blocks, int block_size){
		this.lastAccessedBlockIndex = -1;

		this.n_blocks = n_blocks;
		this.block_size = block_size;
		blocks = new Block[n_blocks];
		for(int i=0; i<n_blocks; i++){
			blocks[i] = new Block(block_size);
		}
	}
	public int getNBlocks(){
		return n_blocks;
	}
	public int getBlockSize(){
		return block_size;
	}


	public Block getBlock(int index){
		return blocks[index];
	}

	/**
	 *
	 * @param tag
	 * @param substPolicy
	 * @return 0 - tag igual, 1 - tag diferente não dirty || empty, 2 - tag diferente
dirty
	 */
	public int doInstructionAccess(long tag, int substPolicy){
		//verifica se a instrução está num dos blocos
		for(int i=0; i<n_blocks; i++){
			if(blocks[i].hasTag(tag)){
				blocks[i].setAccessTime(); //hit, tempo para o LRU
				this.lastAccessedBlockIndex = i;
				return 0;
			}
		}
		//verifica se existem blocos livres, pode contar ou não falhas
compulsorias
```

```
		for(int i=0; i<n_blocks; i++){
			//verifica se existe algum bloco limpo
			if(blocks[i].isClean()){
				blocks[i].tag = tag;
				blocks[i].clean = false;
				blocks[i].dirty_bit = false;
				blocks[i].setAccessTime();
				blocks[i].setCreationTime(); //FIFO index
				this.lastAccessedBlockIndex = i;
				return 1;

				//return 0 se for para eliminar falhas compulsórias
				//return 1 se contar com tudo
			}

		}
		//politica de substituição

		switch(substPolicy){
		case Cache.S_RANDOM:
			return (1+randomSubst(tag));
		case Cache.S_LRU:
			return (1+lruSubst(tag));
		case Cache.S_FIFO:
			return (1+fifoSubst(tag));
		default:
			System.out.println("CacheSet.doInstructionAccess() - Não devia
acontecer isto!");
		return 9999999;
		}


	}


	/**
	 *
	 * @param tag
	 * @param substPolicy
	 * @param writePolicy
	 * @return 0 - tag igual, 1 - tag diferente não dirty || empty, 2 - tag diferente
dirty
	 */
	public int doDataWriteAccess(long tag, int substPolicy, int writePolicy){
		//se a politica de escrita for write-through+no-allocate não altera nada
		if(writePolicy == Cache.WR_THROUGH_NO_ALLOCATE){

			return 0; //não conta como uma cache miss
		}

		//o resto só é executado se for um write-back

		//verifica se os dados estão num dos blocos
		for(int i=0; i<n_blocks; i++){
			if(blocks[i].hasTag(tag)){
				//blocks[i].setCreationTime(); não! pk é no caso de hit
				blocks[i].setAccessTime(); //hit, tempo para o LRU
				blocks[i].dirty_bit = true;
				this.lastAccessedBlockIndex = i;
				return 0;
			}
		}
		//verifica se existem blocos livres, neste caso existem falhas porque tem
q ir buscar o resto do bloco
		for(int i=0; i<n_blocks; i++){
			//verifica se existe algum bloco limpo
			if(blocks[i].isClean()){
				blocks[i].tag = tag;
				blocks[i].clean = false;
				blocks[i].dirty_bit = true;
				blocks[i].setAccessTime();
				blocks[i].setCreationTime(); //FIFO index
				this.lastAccessedBlockIndex = i;

				return 1;

				//return 0 se for para eliminar falhas compulsórias
```

```
                        //return 1 se contar com tudo
                }

        }

        int ret;
        //CUIDADO!!! tem que meter o dirty_bit a true
        //politica de substituição
        if(writePolicy == Cache.WR_BACK_ALLOCATE){
                switch(substPolicy){
                case Cache.S_RANDOM:
                        ret = randomSubst(tag);
                        blocks[lastAccessedBlockIndex].dirty_bit = true;
                        return (1+ret);
                case Cache.S_LRU:
                        ret =  lruSubst(tag);
                        blocks[lastAccessedBlockIndex].dirty_bit = true;
                        return (1+ret);

                case Cache.S_FIFO:
                        ret = fifoSubst(tag);
                        blocks[lastAccessedBlockIndex].dirty_bit = true;
                        return (1+ret);
                default:
                        System.out.println("CacheSet.doDataReadAccess() - Não devia
acontecer isto!");
                return 9999999;
                }
        }else{
                System.out.println("CacheSet.doDataWriteAccess() - Não deveria
nunca chegar a isto!");
                return 999999;
        }



    }



    /**
     *
     * @param tag
     * @param substPolicy
     * @return 0 - tag igual, 1 - tag diferente não dirty || empty, 2 - tag diferente
dirty
     */
    public int doDataReadAccess(long tag, int substPolicy){
        //verifica se os dados estão num dos blocos
        for(int i=0; i<n_blocks; i++){
                if(blocks[i].hasTag(tag)){
                        blocks[i].setAccessTime(); //hit, tempo para o LRU
                        this.lastAccessedBlockIndex = i;
                        return 0;
                }
        }
        //verifica se existem blocos livres, pode contar ou não falhas
compulsorias
        for(int i=0; i<n_blocks; i++){
                //verifica se existe algum bloco limpo
                if(blocks[i].isClean()){
                        blocks[i].tag = tag;
                        blocks[i].clean = false;
                        blocks[i].dirty_bit = false;
                        blocks[i].setAccessTime();
                        blocks[i].setCreationTime(); //FIFO index
                        this.lastAccessedBlockIndex = i;
                        return 1;

                        //return 0 se for para eliminar falhas compulsórias
                        //return 1 se contar com tudo
                }

        }
        //politica de substituição

        switch(substPolicy){
```

```
		case Cache.S_RANDOM:
			//faz 2 acesso a memória um para escrever o bloco e outro para
escrever
			return (1 + randomSubst(tag));
		case Cache.S_LRU:
			return (1+lruSubst(tag));
		case Cache.S_FIFO:
			return (1+fifoSubst(tag));
		default:
			System.out.println("CacheSet.doDataReadAccess() - Não devia
acontecer isto!");
		return 9999999;
		}



	}

	public int getLastAccessedBlockIndex(){
		return lastAccessedBlockIndex;
	}

	public Block getLastAccessedBlock(){
		return this.blocks[this.lastAccessedBlockIndex];
	}


	//retorna 1 caso o bloco escolhido estiver "dirty" ou 0 se estiver "limpo"
	private int randomSubst(long tag){
		boolean dirty;
		int b = (int)(Math.random()*(n_blocks));

		//ultimo bloco a ser acedido
		lastAccessedBlockIndex = b;

		dirty = blocks[b].dirty_bit;
		//System.out.println("Random="+b);

		blocks[b].clean = false;


		//so coloca o dirty a false se a tag for igual
		if(tag != blocks[b].tag)
			blocks[b].dirty_bit = false;


		blocks[b].tag = tag;
		blocks[b].setAccessTime();
		blocks[b].setCreationTime();
		if(dirty)
			return 1;
		else
			return 0;


	}

	//retorna 1 caso o bloco escolhido estiver "dirty" ou 0 se estiver "limpo"
	private int lruSubst(long tag){
		boolean dirty;
		int index=0;
		//encontra o bloco que está hà mais tempo sem nenhum hit
		for(int i=0; i<this.n_blocks; i++){
			if(blocks[index].getLastAccessTime() >
blocks[i].getLastAccessTime()) //pk o i é mais antigo
				index = i;
		}

		//ultimo bloco a ser acedido
		lastAccessedBlockIndex = index;

		dirty = blocks[index].dirty_bit;


		blocks[index].clean = false;

		if(tag != blocks[index].tag)
```

```
                        blocks[index].dirty_bit = false;

                blocks[index].tag = tag;

                blocks[index].setAccessTime();
                blocks[index].setCreationTime();
                if(dirty)
                        return 1;
                else
                        return 0;
        }

        //retorna 1 caso o bloco escolhido estiver "dirty" ou 0 se estiver "limpo"
        private int fifoSubst(long tag){
                boolean dirty;
                int index=0;
                //encontra o bloco que está hà mais tempo sem nenhum hit
                for(int i=0; i<this.n_blocks; i++){
                        if(blocks[index].getFIFOindex() > blocks[i].getFIFOindex()) //pk o
i é mais antigo
                                index = i;
                }

                //ultimo bloco a ser acedido
                lastAccessedBlockIndex = index;


                dirty = blocks[index].dirty_bit;

                if(tag != blocks[index].tag)
                        blocks[index].dirty_bit = false;


                blocks[index].tag = tag;
                blocks[index].clean = false;

                blocks[index].setAccessTime();
                blocks[index].setCreationTime(); //set fifo index
                if(dirty)
                        return 1;
                else
                        return 0;
        }


}
```

6.2.5 MainCanvas

Esta classe representa a área de desenho que é visível na interface principal. Tem uma variável com os parâmetros da simulação e desenha no ecrã de acordo com o estado interno da classe. O estado interno é controlado pela thread de simulação representado pela classe SimulationThread.

```
package cacheSim;

import java.awt.Canvas;
import java.awt.Color;
import java.awt.Graphics;

/**
 * Esta classe representa a parte de Desenho da interface gráfica.
 * Tem os dados da simulação como a cache, aonde vai buscar a informação para
representa-la no ecra.
 * @author Manuel
 *
 */
public class MainCanvas extends Canvas {
	public static final long serialVersionUID = 0 ;

	public int my_x;
	public int my_y;

	public static final int X_IND_BLOCK = 70;

	//coordenadas iniciais dos vários elementos
	//possivel implementação do zoom
	//private float factor = 1;
	private MyPoint c1 = new MyPoint(15,65); //cache1
	private MyPoint c2 = new MyPoint(15,0); //cache 2
	private final MyPoint bb = new MyPoint(5,15); //tamanho minimo para um byte de um
bloco (é o quadrado mais pequeno)
	private MyPoint bsize1 = new MyPoint(0,0);
	private MyPoint bsize2 = new MyPoint(0,0);

	//block seleccionado
	private boolean isBlockSelected;
	private MyPoint blockSelected;
	private MyPoint blockSelectedSize;
	private int cacheSelected = 0;
	private int selected_set_index;

	private SimParams sim = null;


	public MainCanvas(){
		super();
		this.setSize(1024,1024);
		this.setBackground(Color.white);
		my_x = 10 ;
		my_y = 10;

		this.isBlockSelected = false;

	}

	public MyPoint getBlockByteDrawSize(){
		return bb;
	}

	public MyPoint getCache1Pos(){
		return c1;
	}

	public MyPoint getCache2Pos(){
		return c2;
```

```
	}

	public void setParams(SimParams sim){
		this.sim = sim;
		if(this.sim.isSplitCache()){
			int altura = sim.getSplitCache().getICache().getNumBlocks() +
sim.getSplitCache().getDCache().getNumBlocks();
			int largura = sim.getSplitCache().getICache().getBlockSize();
			int largura2 = sim.getSplitCache().getDCache().getBlockSize();
			if(largura2 > largura)
				largura = largura2;

			this.setSize(largura * bb.x + 500, altura * bb.y + 200);

		}else{

			int altura = sim.getUnifiedCache().getUCache().getNumBlocks();
			int largura = sim.getUnifiedCache().getUCache().getBlockSize();
			this.setSize(largura * bb.x + 500, altura * bb.y + 200);

		}
	}




	public void paint(Graphics g){
		if(sim != null){


			//desenho de uma cache de instruções
			if(sim.isSplitCache()){


	/*************************************************************/
				/*		DESENHO CACHE DE INSTRUÇÕES
*/

	/*************************************************************/


				g.fillRect(0,5,80,21);
				g.setColor(Color.WHITE);
				g.drawString("Split Cache",10,20);

				g.setColor(Color.BLACK);
				g.fillRect(0,c1.y-32,120,15);
				g.setColor(Color.WHITE);
				g.drawString("Instruction cache",c1.x, c1.y-20);

				g.setColor(Color.LIGHT_GRAY);
				g.drawString("tag		dirty",c1.x, c1.y-5);

				Cache i_cache = sim.getSplitCache().getICache();
				Cache d_cache = sim.getSplitCache().getDCache();

				//desenha a cache de instruções
				int blocks = i_cache.getNumBlocks();
				int block_size = i_cache.getBlockSize();

				g.setColor(Color.LIGHT_GRAY);
				g.drawString("block",c1.x + X_IND_BLOCK - 35 +
(block_size*bb.x), c1.y-5);

				bsize1.x = bb.x * block_size;
				bsize1.y = bb.y;

				for(int i = 0; i<blocks; i++){

					if(i_cache.getBlock(i).isClean()){
						g.setColor(Color.GREEN);
						g.drawString("empty",c1.x, c1.y + bb.y +
(bb.y * i) - 3 );

					}else{
						g.setColor(Color.BLACK);
```

```
	g.drawString(""+i_cache.getBlock(i).getTag(),c1.x, c1.y + bb.y + (bb.y * i) - 3
);
                                        }

                                        //print block status
                                        g.setColor(new Color(255,240,222));

                                        //g.setColor(new Color(150,150,150));
                                        g.fillRect(c1.x+X_IND_BLOCK, c1.y + (bb.y *
i),bsize1.x, bb.y);

                                        g.setColor(Color.GRAY);
                                        g.drawRect(c1.x+X_IND_BLOCK, c1.y + (bb.y *
i),bsize1.x, bb.y);



                                        //print dirty bit status
                                        if(i_cache.getBlock(i).isDirty())
                                                g.setColor(new Color(200,0,0));
                                        else
                                                g.setColor(Color.LIGHT_GRAY);
                                        g.fillRect(c1.x+X_IND_BLOCK-10, c1.y + (bb.y *
i),10, bb.y);
                                        g.setColor(Color.GRAY);
                                        g.drawRect(c1.x+X_IND_BLOCK-10, c1.y + (bb.y *
i),10, bb.y);







                                        //determina até onde vai a cache de instruções
                                        c2.y = c1.y + (bb.y * i);


                                }


        /*************************************************************/
                                /*                DESENHO CACHE DE DADOS
*/
        /*************************************************************/



                                c2.y += 100; //da-lhe um espaço extra pra separar as caches
                                blocks = d_cache.getNumBlocks();
                                block_size = d_cache.getBlockSize();

                                bsize2.x = bb.x * block_size;
                                bsize2.y = bb.y;

                                g.setColor(Color.BLACK);
                                g.fillRect(0,c2.y-32,120,15);
                                g.setColor(Color.WHITE);
                                g.drawString("Data cache",c2.x, c2.y-20);

                                g.setColor(Color.LIGHT_GRAY);
                                g.drawString("tag              dirty",c2.x, c2.y-5);
                                g.setColor(Color.LIGHT_GRAY);
                                g.drawString("block",c2.x + X_IND_BLOCK - 35 +
(block_size*bb.x), c2.y-5);

                                for(int i = 0; i<blocks; i++){

                                        if(d_cache.getBlock(i).isClean()){
                                                g.setColor(Color.GREEN);
                                                g.drawString("empty",c2.x, c2.y + bb.y +
(bb.y * i) - 3 );

                                        }else{
                                                g.setColor(Color.BLACK);
```

```
	g.drawString(""+d_cache.getBlock(i).getTag(),c2.x, c2.y + bb.y + (bb.y * i) - 3
);
                                        }

                                        g.setColor(new Color(222,240,255));

                                        g.fillRect(c2.x+X_IND_BLOCK, c2.y + (bb.y *
i),bsize2.x, bb.y);
                                        g.setColor(Color.GRAY);
                                        g.drawRect(c2.x+X_IND_BLOCK, c2.y + (bb.y *
i),bsize2.x, bb.y);


//                                      print dirty bit status
                                        if(d_cache.getBlock(i).isDirty())
                                                g.setColor(new Color(200,0,0));
                                        else
                                                g.setColor(Color.LIGHT_GRAY);
                                        g.fillRect(c2.x+X_IND_BLOCK-10, c2.y + (bb.y *
i),10, bb.y);
                                        g.setColor(Color.GRAY);
                                        g.drawRect(c2.x+X_IND_BLOCK-10, c2.y + (bb.y *
i),10, bb.y);

                                }




                        }else{
                                //desenho de uma cache unificada
                                if(sim.isUnifiedCache()){


        /*************************************************************/
                                        /*                DESENHO CACHE UNIFICADA
*/
        /*************************************************************/


                                        g.fillRect(0,5,90,21);
                                        g.setColor(Color.WHITE);
                                        g.drawString("Unified Cache",10,20);

                                        g.setColor(Color.BLACK);

                                        g.fillRect(0,c1.y-32,120,15);
                                        g.setColor(Color.WHITE);
                                        g.drawString("unified cache",c1.x, c1.y-20);

                                        g.setColor(Color.LIGHT_GRAY);
                                        g.drawString("tag              dirty",c1.x, c1.y-
5);

                                        Cache u_cache = sim.getUnifiedCache().getUCache();


                                        //desenha a cache de instruções
                                        int blocks = u_cache.getNumBlocks();
                                        int block_size = u_cache.getBlockSize();

                                        g.setColor(Color.LIGHT_GRAY);
                                        g.drawString("block",c1.x + X_IND_BLOCK - 35 +
(block_size*bb.x), c1.y-5);


                                        bsize1.x = bb.x * block_size;
                                        bsize1.y = bb.y;

                                        for(int i = 0; i<blocks; i++){

                                                if(u_cache.getBlock(i).isClean()){
                                                        g.setColor(Color.GREEN);
```

```
                                                g.drawString("empty",c1.x, c1.y +
bb.y + (bb.y * i) - 3 );

                                        }else{
                                                g.setColor(Color.BLACK);

        g.drawString(""+u_cache.getBlock(i).getTag(),c1.x, c1.y + bb.y + (bb.y * i) - 3
);
                                        }

                                        //print block status
                                        g.setColor(new Color(255,240,222));

                                        //g.setColor(new Color(150,150,150));
                                        g.fillRect(c1.x+X_IND_BLOCK, c1.y + (bb.y *
i),bsize1.x, bb.y);

                                        g.setColor(Color.GRAY);
                                        g.drawRect(c1.x+X_IND_BLOCK, c1.y + (bb.y *
i),bsize1.x, bb.y);



                                        //print dirty bit status
                                        if(u_cache.getBlock(i).isDirty())
                                                g.setColor(new Color(200,0,0));
                                        else
                                                g.setColor(Color.LIGHT_GRAY);
                                        g.fillRect(c1.x+X_IND_BLOCK-10, c1.y + (bb.y
* i),10, bb.y);
                                        g.setColor(Color.GRAY);
                                        g.drawRect(c1.x+X_IND_BLOCK-10, c1.y + (bb.y
* i),10, bb.y);




                                        //determina até onde vai a cache de
instruções
                                        c2.y = c1.y + (bb.y * i);


                                }



                        }
                }


                /*************************************************************/
                /*                DESENHO DO CONJUNTO                        */
                /*************************************************************/

                if(this.isBlockSelected){


                        if(sim.isSplitCache()){
                                int nsets;
                                if(this.cacheSelected == 0)
                                        nsets =
sim.getSplitCache().getICache().getAssociatividade();
                                else
                                        nsets =
sim.getSplitCache().getDCache().getAssociatividade();

                                g.setColor(Color.BLACK);
                                g.drawRect(this.blockSelected.x - X_IND_BLOCK - 5,
this.blockSelected.y, this.blockSelectedSize.x + X_IND_BLOCK + 5,
this.blockSelectedSize.y * nsets );
                                g.drawRect(this.blockSelected.x + 1 - X_IND_BLOCK -
5, this.blockSelected.y + 1, this.blockSelectedSize.x - 2 + X_IND_BLOCK + 5,
this.blockSelectedSize.y * nsets - 2);
```

```
                                        //tentar imprimir as flags de LRU ou FIFO
                                        if(this.cacheSelected == 0){

                                                CacheSet cs =
sim.getSplitCache().getICache().getCacheSet(this.selected_set_index);
                                                int n = cs.getNBlocks();
                                                long newest_fifo = 0;
                                                long newest_lru = 0;
                                                for(int i=0; i<n; i++){
                                                        long fifo =
cs.getBlock(i).getFIFOindex();
                                                        long lru =
cs.getBlock(i).getLastAccessTime();
                                                        if(fifo > newest_fifo)
                                                                newest_fifo = fifo;
                                                        if(lru > newest_lru)
                                                                newest_lru = lru;

                                                }
                                                for(int i=0; i<n; i++){
                                                        Block b = cs.getBlock(i);
                                                        if(!b.isClean()){
                                                                int block_size =
sim.getSplitCache().getICache().getBlockSize();
                                                                g.drawString("FIFO Order:
"+(newest_fifo - b.getFIFOindex()), this.blockSelected.x + 4 + (block_size * bb.x),
this.blockSelected.y + (bb.y*i) + bb.y - 3);
                                                                g.drawString("LRU Order:
"+(newest_lru - b.getLastAccessTime()), this.blockSelected.x + 4 + (32*bb.x) +
(block_size * bb.x), this.blockSelected.y + (bb.y*i) + bb.y - 3);
                                                        }
                                                }


                                        }else{

                                                CacheSet cs =
sim.getSplitCache().getDCache().getCacheSet(this.selected_set_index);
                                                int n = cs.getNBlocks();
                                                long newest_fifo = 0;
                                                long newest_lru = 0;
                                                for(int i=0; i<n; i++){
                                                        long fifo =
cs.getBlock(i).getFIFOindex();
                                                        long lru =
cs.getBlock(i).getLastAccessTime();
                                                        if(fifo > newest_fifo)
                                                                newest_fifo = fifo;
                                                        if(lru > newest_lru)
                                                                newest_lru = lru;

                                                }
                                                for(int i=0; i<n; i++){
                                                        Block b = cs.getBlock(i);
                                                        if(!b.isClean()){
                                                                int block_size =
sim.getSplitCache().getDCache().getBlockSize();
                                                                g.drawString("FIFO Order:
"+(newest_fifo - b.getFIFOindex()), this.blockSelected.x + 4 + (block_size * bb.x),
this.blockSelected.y + (bb.y*i) + bb.y - 3);
                                                                g.drawString("LRU Order:
"+(newest_lru - b.getLastAccessTime()), this.blockSelected.x + 4 + (32*bb.x) +
(block_size * bb.x), this.blockSelected.y + (bb.y*i) + bb.y - 3);
                                                        }
                                                }
                                        }

                                }else if(sim.isUnifiedCache()){
                                        int nsets;
                                        nsets =
sim.getUnifiedCache().getUCache().getAssociatividade();
                                        g.setColor(Color.BLACK);
                                        g.drawRect(this.blockSelected.x - X_IND_BLOCK - 5,
this.blockSelected.y, this.blockSelectedSize.x + X_IND_BLOCK + 5,
this.blockSelectedSize.y * nsets );
```

```
                                        g.drawRect(this.blockSelected.x + 1 - X_IND_BLOCK -
5, this.blockSelected.y + 1, this.blockSelectedSize.x - 2 + X_IND_BLOCK + 5,
this.blockSelectedSize.y * nsets - 2);

                                        //imprimi a ordem das flags para lru e fifo
                                        CacheSet cs =
sim.getUnifiedCache().getUCache().getCacheSet(this.selected_set_index);
                                        int n = cs.getNBlocks();
                                        long newest_fifo = 0;
                                        long newest_lru = 0;
                                        for(int i=0; i<n; i++){
                                                long fifo = cs.getBlock(i).getFIFOindex();
                                                long lru =
cs.getBlock(i).getLastAccessTime();
                                                if(fifo > newest_fifo)
                                                        newest_fifo = fifo;
                                                if(lru > newest_lru)
                                                        newest_lru = lru;

                                        }
                                        for(int i=0; i<n; i++){
                                                Block b = cs.getBlock(i);
                                                if(!b.isClean()){
                                                        int block_size =
sim.getUnifiedCache().getUCache().getBlockSize();
                                                        g.drawString("FIFO Order:
"+(newest_fifo - b.getFIFOindex()), this.blockSelected.x + 4 + (block_size * bb.x),
this.blockSelected.y + (bb.y*i) + bb.y - 3);
                                                        g.drawString("LRU Order:
"+(newest_lru - b.getLastAccessTime()), this.blockSelected.x + 4 + (32*bb.x) +
(block_size * bb.x), this.blockSelected.y + (bb.y*i) + bb.y - 3);
                                                }
                                        }

                                }
                                g.setColor(Color.BLACK);
                        }else{ //imprime o bloco realmente escolhido

                                if(sim.isSplitCache()){

                                        int i =
sim.getSplitCache().getLastAccessBlockIndex(); //pk se trata do index
                                        //nao pode ser inserido código aqui porque ainda
nada foi inicializado
                                        //i == -1

                                        if(i>=0){

                                                MemAccess lma =
sim.getSplitCache().getLastMemAccess();
                                                int offset;
                                                int size = lma.getSize();
                                                if(lma.isInstructionAccess())
                                                        offset = (int)
sim.getSplitCache().getICache().getOffset(lma);
                                                else
                                                        offset = (int)
sim.getSplitCache().getDCache().getOffset(lma);

                                                //cache de instruções
                                                if(sim.getSplitCache().getLastUsedCache() ==
0){
                                                        g.setColor(Color.ORANGE);

                                                        g.fillRect(c1.x + X_IND_BLOCK + 1,
c1.y + (bb.y * i),bsize1.x - 1, bb.y);

                                                        for(int j=0; j<size; j++){
                                                                g.setColor(Color.RED);
                                                                g.fillRect(c1.x + X_IND_BLOCK
+ (offset*bb.x) + j*bb.x, c1.y + (bb.y * i), bb.x, bb.y);
                                                                g.setColor(Color.BLACK);
                                                                g.drawRect(c1.x + X_IND_BLOCK
+ (offset*bb.x) + j*bb.x, c1.y + (bb.y * i), bb.x, bb.y);

                                                        }
```

```
                                                g.setColor(Color.BLACK);
                                                g.drawRect(c1.x - 3 , c1.y + (bb.y *
i),bsize1.x + X_IND_BLOCK + 3, bb.y);


                                        }else{ //cache de dados

                                                g.setColor(Color.ORANGE);

                                                g.fillRect(c2.x + X_IND_BLOCK + 1,
c2.y + (bb.y * i),bsize2.x - 1, bb.y);

                                                for(int j=0; j<size; j++){
                                                        g.setColor(Color.RED);
                                                        g.fillRect(c2.x + X_IND_BLOCK
+ (offset*bb.x) + j*bb.x, c2.y + (bb.y * i), bb.x, bb.y);
                                                        g.setColor(Color.BLACK);
                                                        g.drawRect(c2.x + X_IND_BLOCK
+ (offset*bb.x) + j*bb.x, c2.y + (bb.y * i), bb.x, bb.y);

                                                }

                                                g.setColor(Color.BLACK);
                                                g.drawRect(c2.x - 3 , c2.y + (bb.y *
i),bsize2.x + X_IND_BLOCK + 3, bb.y);


                                        }
                                }


                        }else if(sim.isUnifiedCache()){


                                int i =
sim.getUnifiedCache().getLastAccessBlockIndex(); //pk se trata do index
                                //nao pode ser inserido código aqui porque ainda
nada foi inicializado
                                //i == -1

                                if(i>=0){

                                        MemAccess lma =
sim.getUnifiedCache().getLastMemAccess();
                                        int offset;
                                        int size = lma.getSize();

                                        offset = (int)
sim.getUnifiedCache().getUCache().getOffset(lma);


                                        //cache de instruções

                                        g.setColor(Color.ORANGE);

                                        g.fillRect(c1.x + X_IND_BLOCK + 1, c1.y +
(bb.y * i),bsize1.x - 1, bb.y);

                                        for(int j=0; j<size; j++){
                                                g.setColor(Color.RED);
                                                g.fillRect(c1.x + X_IND_BLOCK +
(offset*bb.x) + j*bb.x, c1.y + (bb.y * i), bb.x, bb.y);
                                                g.setColor(Color.BLACK);
                                                g.drawRect(c1.x + X_IND_BLOCK +
(offset*bb.x) + j*bb.x, c1.y + (bb.y * i), bb.x, bb.y);
                                        }


                                        g.setColor(Color.BLACK);
                                        g.drawRect(c1.x - 3 , c1.y + (bb.y *
i),bsize1.x + X_IND_BLOCK + 3, bb.y);



                                }

                        }
```

```
                        }

                }//end sim != null

                //para teste imprime uma barra em cima de tudo
                //g.fillRect(0,0,my_x,my_y);

        }



        public void setBlockSelected(int x, int y, int block, int set_index){
                this.selected_set_index = set_index;
                this.blockSelected = new MyPoint(x,y);
                this.isBlockSelected = true;
                if(block <=0){
                        this.blockSelectedSize = bsize1;
                        this.cacheSelected = 0;
                }
                else{
                        this.blockSelectedSize = bsize2;
                        this.cacheSelected = 1;
                }
        }
        public void removeBlockSelected(){
                this.isBlockSelected = false;
        }


}
```

6.2.6 MemAccess

Representa um acesso a memória, esta classe é amplamente usada pelas restantes no programa. Contem o endereço, tamanho e tipo do acesso. Tem as respectivas funções para usar esta classe.

```
package cacheSim;

import java.util.StringTokenizer;

/**
 * Representa um acesso a memória
 * tem endereço de memória, tipo (R/W/I) e tamanho
 * e algumas funções utilitarias.
 * @author Manuel
 *
 */
public class MemAccess {
	private long address; // endereço: número sem sinal, 32 bits, em hexadecimal;
	private char type; //tipo R-leitura de dados, W-escrita de dados, I-acesso a
instrução.
	private int size;  //tamanho: número sem sinal indicando a dimensão do item
transferido em bytes.
	private String hex;

	/**
	 * Representa um acesso a memória
	 * @param address endereço de posição de memória em hexadecimal 32 bits
	 * @param type tipo R-leitura de dados, W-escrita de dados, I-acesso a instrução.
	 * @param size tamanho: número sem sinal indicando a dimensão do item transferido
em bytes.
	 */
	public MemAccess(String address, String type, String size)
	throws MemAccessFormatException
	{
		try{
			this.address = hexToLong(address);//address;
			this.type = type.charAt(0);
			this.size = strToInt(size);
			this.hex = address;
		}catch(Exception e){
			//e.printStackTrace();
			System.out.println(e);
			throw new MemAccessFormatException("Argumentos incompativeis!");
		}

	}
	/**
	 * Representa um acesso a memória
	 * @param ma string com formato: "address type size"
	 */
	public MemAccess(String ma)
	throws MemAccessFormatException
	{
		try{
			StringTokenizer ma_st = new StringTokenizer(ma);
			this.hex = ma_st.nextToken();
			this.address = hexToLong(this.hex);
			this.type = ma_st.nextToken().charAt(0);
			this.size = strToInt(ma_st.nextToken());

		}catch(Exception e)
		{
			//e.printStackTrace();
			System.out.println(e);
			throw new MemAccessFormatException("String com formato errado:
'"+ma+"'");
		}
	}
```

```
public MemAccess getCopy()
{
	try{
		return new MemAccess(hex, ""+type, ""+size);
	}catch(Exception e)
	{
		System.out.println(e.getMessage());
		return null;
	}
}

public void incAddress(){
	this.address++;
}

public String getHexAddress(){
	return this.hex;
}
public String getAccessType(){
	return ""+this.type;
}

public boolean isInstructionAccess(){
	if(type == 'i' || type == 'I')
		return true;
	else
		return false;
}

public boolean isDataReadAccess(){
	if(type == 'R' || type == 'r')
		return true;
	else
		return false;
}

public boolean isDataWriteAccess(){
	if(type == 'W' || type == 'w')
		return true;
	else
		return false;
}


public int getSize(){
	return this.size;
}
public long getAddress(){
	return this.address;
}

/**
 * Converte strings para inteiros
 */
private static int strToInt(String s){
	int i = (Integer.valueOf(s)).intValue();
	return i;
}

/**
 * Converte hexadecimais de 32 bits para um long
 * exemplos 'F9' ou '-F9'
 */
private static long hexToLong(String hex){
	//converte valores hexadecimais
	long l = Long.valueOf(hex,16);
	return l;
}

public String toString(){
	return this.hex.toUpperCase()+" "+this.type+" "+this.size;
}

/**
 * Exemplo de como usar o MemAccess
 */
public static void main(String[] args) {
```

```
		try{
		MemAccess x = new MemAccess("3fbfe480","R","8");
		MemAccess y = new MemAccess("3fbfe480   R   8");
		//resultado esperado seria:
		//Address/Type/Size: 1069540480 R 8
		System.out.println("x= "+x);
		System.out.println("y= "+y);
		}catch(Exception e)
		{
			e.printStackTrace();
		}
	}

	public static class MemAccessFormatException
	extends Exception
	{
		//
		static final long serialVersionUID = 0;

		public MemAccessFormatException(String s){
			super(s);
		}

	}

}
```

6.2.7 MemAccessFile

Representa o ficheiro que contém os acessos necessários à simulação. Usa um BufferedReader para aceder aos ficheiros.

```
package cacheSim;

import java.io.*;

/**
 * Representa o ficheiro onde se vai buscar os acessos a memória.
 * Usa um BufferedReader para maior performance.
 * Tem funções para buscar o proximo acesso e para sabes se ja acabou.
 * @author Manuel
 *
 */
public class MemAccessFile {

	private BufferedReader br;

	/**
	 * Representa o ficheiro com os acessos a memória
	 * @param file - ficheiro
	 */
	public MemAccessFile(String file)
	throws FileNotFoundException
	{
		br = new BufferedReader(new FileReader(file));
	}

	public boolean hasMore()
	{
		try{
			if(!br.ready())
				return false;
			else
				return true;
		}catch(Exception e){
			return false;
		}
	}

	public MemAccess getNext()
	throws IOException, MemAccess.MemAccessFormatException
	{
		return new MemAccess(br.readLine());
	}

	/**
	 * Recomeça do inicio do ficheiro
	 * @throws IOException
	 */
	public void reset()
	throws IOException
	{
		br.reset();
	}

	public void close()
	throws IOException
	{
		br.close();
	}



	/**
	 * Demonstração do MemAccessFile
	 * @param args - string com o ficheiro a ser lido
	 */
	public static void main(String[] args) {
```

```
		// TODO Auto-generated method stub

		try{
			//MemAccessFile maf = new
MemAccessFile("d:\\Manuel\\eclipse_workspace\\AAC\\cacheSim\\trace.txt");
			MemAccessFile maf = new MemAccessFile(args[0]);
			while(maf.hasMore()){
				MemAccess ma = maf.getNext();
				System.out.println(ma.toString());
			}
		}catch(Exception e){
			System.out.println(e);
			//e.printStackTrace();
		}



	}

}
```

6.2.8 MyPoint

Classe utilitária, tanto pode representar uma coordenada como a dimensão de uma imagem 2D, por exemplo.

```
package cacheSim;

/**
 * Classe utilitaria
 * Pode ser usada para representar um ponto, coordenada, tamanho de algum objecto 2D
 * @author Manuel
 *
 */
public class MyPoint {


		public int x;
		public int y;
		public MyPoint(int x, int y){
			this.x = x;
			this.y = y;

		}

}
```

6.2.9 NextLevelAccessDialog

Representa a janela que mostra os acessos ao próximo nível de memória resultantes no decorrer da simulação. Por uma questão de performance só mostra os últimos 100 acessos de cada vez. Tem a possibilidade de guardar para um ficheiro todos os acessos no formato especificado no enunciado do trabalho. Utiliza uma lista ligada para guardar os acessos e manter alguma performance de escrita e gestão dinâmica de memória.

```
package cacheSim;

import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.Font;



import javax.swing.JDialog;

import javax.swing.JPanel;

import javax.swing.JButton;


import java.awt.ScrollPane;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;


import java.util.LinkedList;
import java.util.ListIterator;

import javax.swing.JFileChooser;
import javax.swing.JTextArea;
import javax.swing.JLabel;

/**
 * Esta classe representa a janela que mostra os acesso ao próximo nivel de memória
 * O utilizador pode salvar esses acesso para um ficheiro. Pode esconder a janela para
maior rapidez na simulação.
 * @author Manuel
 *
 */
public class NextLevelAccessDialog extends JDialog {

	public static final long serialVersionUID = 0;
	private JPanel jContentPane = null;
	private JPanel jPanelControls = null;
	private JButton jButtonHide = null;
	private JButton jButtonSave = null;


	private LinkedList<MemAccess> array = new LinkedList<MemAccess>();

	private JTextArea jTextArea = null;
	private ScrollPane scrollPane = null;
	private JPanel jPanel = null;
	private JLabel jLabel = null;
	private JLabel jLabel1 = null;
	/**
	 * This is the default constructor
	 */
	public NextLevelAccessDialog(RunProgram frame) {
		super(frame, false);
		initialize();
		this.setLocation(frame.getSize().width, 0);
```

```
	}

	/**
	 * This method initializes this
	 *
	 * @return void
	 */
	private void initialize() {
		this.setSize(270, 600);
		this.setContentPane(getJContentPane());
		this.setTitle("Next Level Access");

	}

	/**
	 * This method initializes jContentPane
	 *
	 * @return javax.swing.JPanel
	 */
	private JPanel getJContentPane() {
		if (jContentPane == null) {
			jContentPane = new JPanel();
			jContentPane.setLayout(new BorderLayout());
			jContentPane.add(getJPanelControls(),
java.awt.BorderLayout.NORTH);
			jContentPane.add(getScrollPane(), java.awt.BorderLayout.CENTER);
			jContentPane.add(getJPanel(), java.awt.BorderLayout.SOUTH);
		}
		return jContentPane;
	}

	/**
	 * This method initializes jPanelControls
	 *
	 * @return javax.swing.JPanel
	 */
	private JPanel getJPanelControls() {
		if (jPanelControls == null) {
			jPanelControls = new JPanel(new FlowLayout(FlowLayout.LEFT));
			jPanelControls.add(getJButtonHide(), null);
			jPanelControls.add(getJButtonSave(), null);
		}
		return jPanelControls;
	}

	/**
	 * This method initializes jButtonHide
	 *
	 * @return javax.swing.JButton
	 */
	private JButton getJButtonHide() {
		if (jButtonHide == null) {
			jButtonHide = new JButton();
			jButtonHide.setText("Hide");
			jButtonHide.addActionListener(new java.awt.event.ActionListener()
{
				public void actionPerformed(java.awt.event.ActionEvent e) {
					NextLevelAccessDialog.this.setVisible(false);
				}
			});
		}
		return jButtonHide;
	}

	/**
	 * This method initializes jButtonSave
	 *
	 * @return javax.swing.JButton
	 */
	private JButton getJButtonSave() {
		if (jButtonSave == null) {
			jButtonSave = new JButton();
			jButtonSave.setText("Save to file...");
			jButtonSave.addActionListener(new java.awt.event.ActionListener()
{
				public void actionPerformed(java.awt.event.ActionEvent e) {
```

```
                                        JFileChooser chooser = new
JFileChooser(System.getProperty("user.dir"));
                                        //chooser.setDialogType(JFileChooser.SAVE_DIALOG);
                                        //chooser.setDialogTitle("Save as...");
                                        // Note: source for ExampleFileFilter can be found
in FileChooserDemo,
                                        // under the demo/jfc directory in the Java 2 SDK,
Standard Edition.
                                        int returnVal =
chooser.showSaveDialog(NextLevelAccessDialog.this);
                                        if(returnVal == JFileChooser.APPROVE_OPTION) {
                                                System.out.println("You chose to open this
file: " +

        chooser.getSelectedFile().getName());

        System.out.println("File="+chooser.getSelectedFile().getAbsolutePath());
                                                File f = chooser.getSelectedFile();
                                                if(f.exists()){
                                                        (new
WarningDialog(NextLevelAccessDialog.this,"Erro", "Ficheiro '+"+f.getName()+"' já existe
escolha outro nome!")).setVisible(true);

                                                }else{
                                                        try{
                                                                saveToFile(f);
                                                        }catch(Exception exc){
                                                                (new
WarningDialog(NextLevelAccessDialog.this,"Erro", "Problemas ao escrever para o ficheiro
'"+f.getName()+"'")).setVisible(true);
                                                                if(f.delete())

        System.out.println("File deleted...");
                                                        }
                                                }


        //NextLevelAccessDialog.this.jTextFieldTraceFile.setText(chooser.getSelectedFile(
).getAbsolutePath());
                                        }
                                }
                        });
                }
                return jButtonSave;
        }

        private void saveToFile(File file)
        throws IOException{

                FileWriter fw=new FileWriter(file);
                BufferedWriter bw=new BufferedWriter(fw);

                ListIterator itr = this.array.listIterator(0);
                while(itr.hasNext()){
                        bw.write(itr.next()+"\n");
                }

                bw.close();
                fw.close();
        }

        public void clearList(){
                this.jTextArea.setText("");
                this.array = new LinkedList<MemAccess>();
        }

        public void insertMemAccess(MemAccess ma){

                //jTextArea.setText(jTextArea.getText()+ma.toString()+"\n");

                array.add(ma);


                if(this.isVisible()){
                        int index = array.size() - 100;
                        if(index < 0)
                                index = 0;
```

```
                        ListIterator<MemAccess> itr = array.listIterator(index);



                        String s = new String();
                        while(itr.hasNext()){
                                s +="\n"+itr.next().toString();
                        }

                        jTextArea.setText(s);

        this.scrollPane.setScrollPosition(0,this.jTextArea.getSize().height+50);
                }



                /*
                 MemAccess [] mas = new MemAccess[this.maArray.length + 1];
                 for(int i=0; i<this.maArray.length; i++){
                 mas[i] = this.maArray[i];
                 }
                 mas[this.maArray.length] = ma;
                 maArray = mas;*/


                //this.jList.setSelectedIndex(maArray.length-1);
                //this.jList.setAutoscrolls(true);


        }

        /**
         * This method initializes jTextArea
         *
         * @return javax.swing.JTextArea
         */
        private JTextArea getJTextArea() {
                if (jTextArea == null) {
                        jTextArea = new JTextArea();
                        jTextArea.setBounds(new java.awt.Rectangle(5,101,10,16));
                        jTextArea.setFont(new Font("Courier New",Font.PLAIN, 12));


                }
                return jTextArea;
        }

        /**
         * This method initializes scrollPane
         *
         * @return java.awt.ScrollPane
         */
        private ScrollPane getScrollPane() {
                if (scrollPane == null) {
                        scrollPane = new ScrollPane();
                        scrollPane.add(getJTextArea(), null);
                }
                return scrollPane;
        }

        /**
         * This method initializes jPanel
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJPanel() {
                if (jPanel == null) {
                        jLabel1 = new JLabel();
                        jLabel1.setText("  to see the entire list save to a file
first...");
                        jLabel1.setForeground(new java.awt.Color(204,0,0));
                        jLabel = new JLabel();
                        jLabel.setText("  Only shows the last 100 next level access,");
                        jLabel.setForeground(new java.awt.Color(204,0,0));
                        jPanel = new JPanel(new BorderLayout());
                        jPanel.setBackground(new java.awt.Color(241,242,245));
```

```
                        jPanel.add(jLabel, BorderLayout.NORTH);
                        jPanel.add(jLabel1, BorderLayout.SOUTH);
                }
                return jPanel;
        }




}
```

6.2.10 RunProgram

A classe que arranca o programa, embora seja possível executar em modo consola cada um dos tipos de Cache (SplitCache ou UnifiedCache) para simulações maiores.

É uma classe que extende JFrame e que contem todos os componentes gráficos da janela principal. Esta classe também contem um objecto do tipo SimulationThread que processa a simulação e tem um objecto do tipo MainCanvas adicionado à sua parte gráfica para poder renderizar o estado da cache e ver a simulação a correr.

```
package cacheSim;


import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;



import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Font;

import java.awt.GridLayout;


import javax.swing.BorderFactory;
import javax.swing.ImageIcon;
import javax.swing.JPanel;
import javax.swing.JMenuItem;
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JFrame;

import javax.swing.JButton;
import javax.swing.JLabel;

import javax.swing.JTextField;
import java.awt.ScrollPane;
import javax.swing.JSlider;

/**
 * Classe gráfica principal neste projecto, a partir daqui pode-se fazer tudo.
 * Ver manual do utilizador para saber como é constituida.
 * Cria/destroi a thread de simulação que se encarrega de actualizar esta interface.
 * @author Manuel
 *
 */
public class RunProgram extends JFrame {

	public static final long serialVersionUID = 0;
	public static final int START_SLEEP = 1000;
	public static final int MIN_SLEEP = 1;
	public static final int MAX_SLEEP = 2000;

	private boolean nextLevelRegistry;
	private NextLevelAccessDialog nextLevelDialog;

	public MainCanvas myCanvas;

	private SimulationThread animationThread;

	private JPanel jContentPane = null;

	private JMenuBar jJMenuBar = null;

	private JMenu fileMenu = null;
```

```
    private JMenu helpMenu = null;

    private JMenuItem exitMenuItem = null;

    private JMenuItem aboutMenuItem = null;


    private JPanel jPanelToolbar = null;

    private JPanel jPanelLegenda = null;


    private JButton jButtonPlay = null;

    private JButton jButtonPause = null;

    private JButton jButtonStepInto = null;

    private JPanel jPanelToolbarControls = null;

    private JMenuItem jMenuItemNewSim = null;

    private JPanel jPanelStatus = null;

    private JLabel jLabelStatus = null;

    private JPanel jPanelEstatisticas = null;

    private JPanel jPanelEstatisticasControls = null;

    private JLabel jLabel2 = null;

    private JLabel jLabel3 = null;

    private JTextField jTextFieldAcessos = null;

    private JLabel jLabel4 = null;

    private JLabel jLabel5 = null;

    private JTextField jTextFieldCacheHits = null;

    private JLabel jLabel6 = null;

    private JTextField jTextFieldCacheMiss = null;

    public ScrollPane scrollPaneCanvas = null;

    private JSlider jSliderSpeed = null;

    private JTextField jTextFieldSpeed = null;
    private JPanel jPanelAccess = null;
    private JPanel jPanelAccessControls = null;
    private JLabel jLabel = null;
    private JTextField jTextFieldAddress = null;
    private JLabel jLabel1 = null;
    private JTextField jTextFieldAccessType = null;
    private JLabel jLabel7 = null;
    private JTextField jTextFieldAccessSize = null;
    private JMenuItem jMenuItemNewSim2 = null;
    private JPanel jPanelSecundaryControls = null;
    private JButton jButtonShowNextLevelAccess = null;
    private JLabel jLabel8 = null;
    private JTextField jTextFieldWriteBacks = null;
    private JLabel jLabel9 = null;
    private JTextField jTextFieldWriteThroughs = null;
    private JLabel jLabel10 = null;
    private JTextField jTextFieldNextLevel = null;
    private JPanel jPanelNaoEstatisticas = null;
    private JLabel jLabel11 = null;
    private JLabel jLabel12 = null;
    private JLabel jLabel13 = null;
    private JTextField jTextFieldIAccess = null;
    private JLabel jLabel14 = null;
    private JTextField jTextFieldDRAccess = null;
```

```
	private JLabel jLabel15 = null;
	private JTextField jTextFieldDWAccess = null;
	private JPanel jPanelAccessDecoded = null;
	private JLabel jLabelDecimal = null;
	private JTextField jTextFieldTag = null;
	private JPanel jPanel = null;
	private JPanel jPanel1 = null;
	private JTextField jTextFieldBinario = null;
	private JTextField jTextFieldBinario2 = null;
	private JTextField jTextFieldBinario3 = null;
	private JTextField jTextFieldIndex = null;
	private JTextField jTextFieldOffset = null;
	private JLabel jLabelBinary = null;
	private JPanel jPanelDecodedTop = null;
	private JLabel jLabelBits = null;
	public void setNextLevelRegistry(boolean b){
		this.nextLevelRegistry = b;
	}

	public boolean getNextLevelRegistry(){
		return this.nextLevelRegistry;
	}

	public void setReadyStatus(boolean b){
		this.myCanvas.repaint();
		if(b){
			jButtonPlay.setEnabled(true);
			jButtonPause.setEnabled(true);
			jButtonStepInto.setEnabled(true);

		}else{
			jButtonPlay.setEnabled(false);
			jButtonPause.setEnabled(false);
			jButtonStepInto.setEnabled(false);

		}

	}

	public void addNextLevelAccess(MemAccess ma){
		if(this.getNextLevelRegistry())
			this.nextLevelDialog.insertMemAccess(ma);
	}

	/**
	 * imprime na interface qual o próximo acesso
	 * @param ma - O acesso a memória
	 */
	public void setNextAccess(MemAccess ma){
		jTextFieldAddress.setText(""+ma.getHexAddress().toUpperCase());
		jTextFieldAccessType.setText(""+ma.getAccessType());
		jTextFieldAccessSize.setText(""+ma.getSize());
	}

	public void setNextDecodedAddress(long tag, int index, long offset, int b1, int
b2, int b3){
		this.jLabelBits.setText("( tag = "+b1+" bits | index = "+b2+" bits |
offset = "+b3+" bits )");
		//this.jLabelBinary.setText()
		/*
		 this.jTextFieldTag.setColumns(b1);
		 this.jTextFieldBinario.setColumns(b1);

		 this.jTextFieldIndex.setColumns(b2);
		 this.jTextFieldBinario2.setColumns(b2);

		 this.jTextFieldOffset.setColumns(b3);
		 this.jTextFieldBinario3.setColumns(b3);
		 */

		this.jTextFieldTag.setText(""+tag);
		this.jTextFieldBinario.setText(bitFormat(tag, b1));

		this.jTextFieldIndex.setText(""+index);
		this.jTextFieldBinario2.setText(bitFormat(index, b2));

		this.jTextFieldOffset.setText(""+offset);
```

```
                this.jTextFieldBinario3.setText(bitFormat(offset, b3));
        }

        private String bitFormat(long n, int n_bits){
                String s = Long.toBinaryString(n);
                //for(int i=0; i < (n_bits - s.length()); i++)
                while(s.length() < n_bits)
                        s = "0" + s;
                return s;
        }
        private String bitFormat(int n, int n_bits){
                String s = Integer.toBinaryString(n);
                //for(int i=0; i < (n_bits - s.length()); i++)
                while(s.length() < n_bits)
                        s = "0" + s;
                return s;

        }

        public void resetStatistcsBackColor(){
                Color c1 = Color.LIGHT_GRAY;
                this.jTextFieldAcessos.setBackground(c1);
                this.jTextFieldCacheHits.setBackground(c1);
                this.jTextFieldCacheMiss.setBackground(c1);
                this.jTextFieldWriteThroughs.setBackground(c1);
                this.jTextFieldWriteBacks.setBackground(c1);
                this.jTextFieldNextLevel.setBackground(c1);
                this.jTextFieldIAccess.setBackground(c1);
                this.jTextFieldDRAccess.setBackground(c1);
                this.jTextFieldDWAccess.setBackground(c1);

        }
        public void setStatistcs(long acessos, long hits, long misses, long wr_throughs,
long wr_backs, long next_level, long i_access, long dr_access, long dw_access){
                Color c1 = Color.LIGHT_GRAY;
                Color c2 = new Color(255,128,0);
                Color c3 = new Color(255,0,0);
                try{
                        //numero de acessos
                        if(Long.parseLong(this.jTextFieldIAccess.getText()) != acessos)
                                this.jTextFieldAcessos.setBackground(c2);
                        else
                                this.jTextFieldAcessos.setBackground(c1);
                        this.jTextFieldAcessos.setText(""+acessos);

                        //numero de hits
                        if(Long.parseLong(this.jTextFieldCacheHits.getText()) != hits)
                                this.jTextFieldCacheHits.setBackground(c2);
                        else
                                this.jTextFieldCacheHits.setBackground(c1);
                        this.jTextFieldCacheHits.setText(""+hits);

                        //numero de misses
                        if(Long.parseLong(this.jTextFieldCacheMiss.getText()) != misses)
                                this.jTextFieldCacheMiss.setBackground(c2);
                        else
                                this.jTextFieldCacheMiss.setBackground(c1);

                        this.jTextFieldCacheMiss.setText(""+misses);

                        //numero de write-throughs
                        if(Long.parseLong(this.jTextFieldWriteThroughs.getText()) !=
wr_throughs)
                                this.jTextFieldWriteThroughs.setBackground(c2);
                        else
                                this.jTextFieldWriteThroughs.setBackground(c1);

                        this.jTextFieldWriteThroughs.setText(""+wr_throughs);

                        //numero de write-backs
                        if(Long.parseLong(this.jTextFieldWriteBacks.getText()) !=
wr_backs)
                                this.jTextFieldWriteBacks.setBackground(c2);
                        else
                                this.jTextFieldWriteBacks.setBackground(c1);
                        this.jTextFieldWriteBacks.setText(""+wr_backs);
```

```
                        //numero de acessos ao proximo nivel
                        long n = next_level -
Long.parseLong(jTextFieldNextLevel.getText());
                        if(n >= 2)
                                this.jTextFieldNextLevel.setBackground(c3);
                        if(n == 1)
                                this.jTextFieldNextLevel.setBackground(c2);
                        if(n == 0)
                                this.jTextFieldNextLevel.setBackground(c1);
                        this.jTextFieldNextLevel.setText(""+next_level);

                        /*
                         if(Long.parseLong(jTextFieldNextLevel.getText()) != next_level)
                         this.jTextFieldNextLevel.setBackground(c2);
                         else
                         this.jTextFieldNextLevel.setBackground(c1);
                         this.jTextFieldNextLevel.setText(""+next_level);
                         */

                        //instruction access
                        if(Long.parseLong(jTextFieldIAccess.getText()) != i_access)
                                this.jTextFieldIAccess.setBackground(c2);
                        else
                                this.jTextFieldIAccess.setBackground(c1);
                        this.jTextFieldIAccess.setText(""+i_access);

                        //datawrite access
                        if(Long.parseLong(jTextFieldDRAccess.getText()) != dr_access)
                                jTextFieldDRAccess.setBackground(c2);
                        else
                                jTextFieldDRAccess.setBackground(c1);
                        this.jTextFieldDRAccess.setText(""+dr_access);

                        if(Long.parseLong(jTextFieldDWAccess.getText())!= dw_access)
                                jTextFieldDWAccess.setBackground(c2);
                        else
                                jTextFieldDWAccess.setBackground(c1);
                        this.jTextFieldDWAccess.setText(""+dw_access);
                }catch(Exception e){
                        System.out.println("Exception: "+e.getMessage());
                }
        }

        public void setupSimThread(SimParams params){
                this.nextLevelDialog.clearList();
                if(animationThread == null){
                        animationThread = new SimulationThread(RunProgram.this, params);
                        animationThread.start();
                }else{
                        try{
                                animationThread.stopThread();
                                animationThread.join();
                        }catch(Exception exc){
                                System.out.println(exc.getMessage());
                        }
                        animationThread = new SimulationThread(RunProgram.this, params);
                        animationThread.setParams(params);
                        animationThread.start();
                        this.scrollPaneCanvas.setScrollPosition(0,0);

                }
        }
        public void setStatusMessage(String s){
                this.jLabelStatus.setText(s);
        }
        public int getSleepTime(){
                if(this.jSliderSpeed != null){
                        return this.jSliderSpeed.getValue();
                }
                else
                        return RunProgram.START_SLEEP;
        }

        public void setScrollPosition(int x, int y){
                //Dimension d = this.getthis.scrollPaneCanvas.getSize();
                this.scrollPaneCanvas.setScrollPosition(x,y-100);
        }
```

```
	/**
	 * This method initializes jPanelToolbar
	 *
	 * @return javax.swing.JPanel
	 */
	private JPanel getJPanelToolbar() {
		if (jPanelToolbar == null) {
			jPanelToolbar = new JPanel(new BorderLayout());
			jPanelToolbar.add(getJPanel(), BorderLayout.NORTH);
			jPanelToolbar.add(getJPanelAccess(), java.awt.BorderLayout.WEST);
			jPanelToolbar.add(getJPanelAccessDecoded(),
java.awt.BorderLayout.SOUTH);


		}
		return jPanelToolbar;
	}

	/**
	 * This method initializes jPanelLegenda
	 *
	 * @return javax.swing.JPanel
	 */
	private JPanel getJPanelLegenda() {
		if (jPanelLegenda == null) {
			jPanelLegenda = new JPanel();
			jPanelLegenda.setLayout(new BorderLayout());
			jPanelLegenda.add(getJPanelStatus(), BorderLayout.WEST);
			jPanelLegenda.add(getJPanelSecundaryControls(),
java.awt.BorderLayout.EAST);
		}
		return jPanelLegenda;
	}



	/**
	 * This method initializes jButtonPlay
	 *
	 * @return javax.swing.JButton
	 */
	private JButton getJButtonPlay() {
		if (jButtonPlay == null) {


			jButtonPlay = new JButton(createImageIcon("play.gif", "teste"));
			jButtonPlay.setEnabled(false);

			jButtonPlay.setText("play");
			jButtonPlay.addActionListener(new java.awt.event.ActionListener()
{
				public void actionPerformed(java.awt.event.ActionEvent e) {
					System.out.println("play()"); // TODO Auto-
generated Event stub actionPerformed()

					//garante que só cria uma thread
					if (animationThread == null) {
						//animationThread = new
SimulationThread(RunProgram.this);
						//animationThread.start();

					}
					else{
						animationThread.setPause(false);
						animationThread.setPlayMode(true);
					}
				}
			});



		}
```

```
		return jButtonPlay;
	}

	protected static ImageIcon createImageIcon(String path,String description) {
		java.net.URL imgURL = RunProgram.class.getResource(path);
		if (imgURL != null) {
			return new ImageIcon(imgURL, description);
		} else {
			System.err.println("Couldn't find file: " + path);
			return null;
		}
	}

	/**
	 * This method initializes jButtonPause
	 *
	 * @return javax.swing.JButton
	 */
	private JButton getJButtonPause() {
		if (jButtonPause == null) {
			//jButtonPause = new JButton();
			jButtonPause = new JButton(createImageIcon("pause.gif", "teste"));
			jButtonPause.setEnabled(false);
			jButtonPause.setText("pause");
			jButtonPause.addActionListener(new java.awt.event.ActionListener()
{
				public void actionPerformed(java.awt.event.ActionEvent e) {
					System.out.println("pause()"); // TODO Auto-
generated Event stub actionPerformed()
					if(animationThread != null){
						animationThread.pauseAnimation();
						animationThread.setPlayMode(true);
					}
				}
			});
		}
		return jButtonPause;
	}

	/**
	 * This method initializes jButtonStepInto
	 *
	 * @return javax.swing.JButton
	 */
	private JButton getJButtonStepInto() {
		if (jButtonStepInto == null) {
			//jButtonStepInto = new JButton();
			jButtonStepInto = new JButton(createImageIcon("step.gif","oh
yaeaH!"));
			jButtonStepInto.setEnabled(false);
			jButtonStepInto.setText("next step");
			jButtonStepInto.addActionListener(new
java.awt.event.ActionListener() {
				public void actionPerformed(java.awt.event.ActionEvent e) {
					RunProgram.this.jSliderSpeed.setValue(1);
					//System.out.println("stepInto()"); // TODO Auto-
generated Event stub actionPerformed()
					RunProgram.this.animationThread.setStepInto(true);
					RunProgram.this.animationThread.setPlayMode(true);
					RunProgram.this.animationThread.setPause(true);
				}
			});
		}
		return jButtonStepInto;
	}

	/**
	 * This method initializes jPanel
	 *
	 * @return javax.swing.JPanel
	 */
	private JPanel getJPanel() {
		if (jPanelToolbarControls == null) {
			jPanelToolbarControls = new JPanel(new
FlowLayout(FlowLayout.LEFT));
			jPanelToolbarControls.add(getJButtonPlay());
			jPanelToolbarControls.add(getJButtonPause());
```

```
                        jPanelToolbarControls.add(getJButtonStepInto());
                        jPanelToolbarControls.add(getJSliderSpeed());
                        jPanelToolbarControls.add(getJTextFieldSpeed());
                }
                return jPanelToolbarControls;
        }

        /**
         * This method initializes jMenuItemNewSim
         *
         * @return javax.swing.JMenuItem
         */
        private JMenuItem getJMenuItemNewSim() {
                if (jMenuItemNewSim == null) {
                        jMenuItemNewSim = new JMenuItem();
                        jMenuItemNewSim.setText("New UnifiedCache Simulation");
                        jMenuItemNewSim.addActionListener(new
java.awt.event.ActionListener() {
                                public void actionPerformed(java.awt.event.ActionEvent e) {
                                        System.out.println("actionPerformed()"); // TODO
Auto-generated Event stub actionPerformed()
                                        UnifiedCacheDialog newSimDialog = new
UnifiedCacheDialog(RunProgram.this);
                                        newSimDialog.setVisible(true);
                                }
                        });
                }
                return jMenuItemNewSim;
        }

        /**
         * This method initializes jPanelStatus
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJPanelStatus() {
                if (jPanelStatus == null) {
                        jLabelStatus = new JLabel();
                        jLabelStatus.setText("Estado do programa...");
                        jPanelStatus = new JPanel(new BorderLayout());
                        jPanelStatus.add(jLabelStatus, BorderLayout.WEST);

                }
                return jPanelStatus;
        }

        /**
         * This method initializes jPanelEstatisticas
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJPanelEstatisticas() {
                if (jPanelEstatisticas == null) {
                        jPanelEstatisticas = new JPanel();
                        jPanelEstatisticas.setLayout(new BorderLayout());

                        jPanelEstatisticas.add(getJPanelEstatisticasControls(),
BorderLayout.NORTH);
                        jPanelEstatisticas.add(getJPanelNaoEstatisticas(),
BorderLayout.SOUTH);
                }
                return jPanelEstatisticas;
        }

        /**
         * This method initializes jPanelEstatisticasControls
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJPanelEstatisticasControls() {
                if (jPanelEstatisticasControls == null) {
                        jLabel10 = new JLabel();
                        jLabel10.setText("Next Level");
                        jLabel10.setHorizontalAlignment(JLabel.RIGHT);
                        jLabel9 = new JLabel();
                        jLabel9.setText("Write-Throughs");
                        jLabel9.setHorizontalAlignment(JLabel.RIGHT);
```

```
                        jLabel8 = new JLabel();
                        jLabel8.setText("Write-backs");
                        jLabel8.setHorizontalAlignment(JLabel.RIGHT);
                        jLabel6 = new JLabel();
                        jLabel6.setText("Cache Misses");
                        jLabel6.setHorizontalAlignment(JLabel.RIGHT);
                        jLabel5 = new JLabel();
                        jLabel5.setText("Cache Hits");
                        jLabel5.setHorizontalAlignment(JLabel.RIGHT);
                        jLabel4 = new JLabel();
                        jLabel4.setText("");
                        jLabel3 = new JLabel();
                        jLabel3.setText("Estatisticas  ");
                        jLabel2 = new JLabel();
                        jLabel2.setHorizontalAlignment(JLabel.RIGHT);
                        jLabel2.setText("Acessos  ");
                        jPanelEstatisticasControls = new JPanel();
                        jPanelEstatisticasControls.setLayout(new GridLayout(7,2,5,5));

                        jPanelEstatisticasControls.add(jLabel4);
                        jPanelEstatisticasControls.add(jLabel3);
                        jPanelEstatisticasControls.add(jLabel2);
                        jPanelEstatisticasControls.add(getJTextFieldAcessos());
                        jPanelEstatisticasControls.add(jLabel5);
                        jPanelEstatisticasControls.add(getJTextFieldCacheHits(), null);
                        jPanelEstatisticasControls.add(jLabel6, null);
                        jPanelEstatisticasControls.add(getJTextFieldCacheMiss(), null);

                        jPanelEstatisticasControls.add(jLabel9, null);
                        jPanelEstatisticasControls.add(getJTextFieldWriteThroughs(),
null);
                        jPanelEstatisticasControls.add(jLabel8, null);
                        jPanelEstatisticasControls.add(getJTextFieldWriteBacks(), null);
                        jPanelEstatisticasControls.add(jLabel10, null);
                        jPanelEstatisticasControls.add(getJTextFieldNextLevel(), null);


                }
                return jPanelEstatisticasControls;
        }

        /**
         * This method initializes jTextFieldAcessos
         *
         * @return javax.swing.JTextField
         */
        private JTextField getJTextFieldAcessos() {
                if (jTextFieldAcessos == null) {
                        jTextFieldAcessos = new JTextField();

                        jTextFieldAcessos.setColumns(8);
                        jTextFieldAcessos.setText("0");
                        jTextFieldAcessos.setEditable(false);
                        jTextFieldAcessos.setBackground(Color.WHITE);


                }
                return jTextFieldAcessos;
        }

        /**
         * This method initializes jTextFieldCacheHits
         *
         * @return javax.swing.JTextField
         */
        private JTextField getJTextFieldCacheHits() {
                if (jTextFieldCacheHits == null) {
                        jTextFieldCacheHits = new JTextField();
                        jTextFieldCacheHits.setText("0");
                        jTextFieldCacheHits.setEditable(false);
                        jTextFieldCacheHits.setBackground(Color.WHITE);
                }
                return jTextFieldCacheHits;
        }

        /**
         * This method initializes jTextFieldCacheMiss
```

```
         *
         * @return javax.swing.JTextField
         */
        private JTextField getJTextFieldCacheMiss() {
                if (jTextFieldCacheMiss == null) {
                        jTextFieldCacheMiss = new JTextField();
                        jTextFieldCacheMiss.setText("0");
                        jTextFieldCacheMiss.setEditable(false);
                        jTextFieldCacheMiss.setBackground(Color.WHITE);
                }
                return jTextFieldCacheMiss;
        }

        /**
         * This method initializes scrollPaneCanvas
         *
         * @return java.awt.ScrollPane
         */
        private ScrollPane getScrollPaneCanvas() {
                if (scrollPaneCanvas == null) {
                        scrollPaneCanvas = new ScrollPane();
                        myCanvas = new MainCanvas();
                        myCanvas.createBufferStrategy(1);
                        scrollPaneCanvas.add(myCanvas);

                }
                return scrollPaneCanvas;
        }

        /**
         * This method initializes jSliderSpeed
         *
         * @return javax.swing.JSlider
         */
        private JSlider getJSliderSpeed() {
                if (jSliderSpeed == null) {
                        jSliderSpeed = new
JSlider(RunProgram.MIN_SLEEP,RunProgram.MAX_SLEEP, RunProgram.START_SLEEP);
                        jSliderSpeed.setName("Speed");
                        jSliderSpeed.setToolTipText("Defines the speed of the simulation,
bigger means slower simulation...");

                        jSliderSpeed.addChangeListener(new
javax.swing.event.ChangeListener() {
                                public void stateChanged(javax.swing.event.ChangeEvent e) {


        RunProgram.this.jTextFieldSpeed.setText(""+RunProgram.this.jSliderSpeed.getValue(
));
                                        if(RunProgram.this.animationThread != null)

        RunProgram.this.animationThread.setSpeed(RunProgram.this.jSliderSpeed.getValue())
;
                                }
                        });
                }
                return jSliderSpeed;
        }



        /**
         * This method initializes jTextFieldSpeed
         *
         * @return javax.swing.JTextField
         */
        private JTextField getJTextFieldSpeed() {
                if (jTextFieldSpeed == null) {
                        jTextFieldSpeed = new JTextField();
                        jTextFieldSpeed.setEditable(false);
                        jTextFieldSpeed.setColumns(4);
                        jTextFieldSpeed.setText(""+RunProgram.START_SLEEP);
                }
                return jTextFieldSpeed;
        }
```

```
/**
 * This method initializes jPanelAccess
 *
 * @return javax.swing.JPanel
 */
private JPanel getJPanelAccess() {
	if (jPanelAccess == null) {
		jPanelAccess = new JPanel(new GridLayout(1,2,5,5));
		jPanelAccess.add(getJPanelAccessControls());
	}
	return jPanelAccess;
}



/**
 * This method initializes jPanelAccessControls
 *
 * @return javax.swing.JPanel
 */
private JPanel getJPanelAccessControls() {
	if (jPanelAccessControls == null) {
		jLabel7 = new JLabel();
		jLabel7.setText("Access Size");
		jLabel1 = new JLabel();
		jLabel1.setText("Access Type");
		jLabel = new JLabel();
		jLabel.setText("Address");
		jPanelAccessControls = new JPanel();
		jPanelAccessControls.add(jLabel, null);
		jPanelAccessControls.add(getJTextFieldAddress(), null);
		jPanelAccessControls.add(jLabel1, null);
		jPanelAccessControls.add(getJTextFieldAccessType(), null);
		jPanelAccessControls.add(jLabel7, null);
		jPanelAccessControls.add(getJTextFieldAccessSize(), null);

	}
	return jPanelAccessControls;
}



/**
 * This method initializes jTextFieldAddress
 *
 * @return javax.swing.JTextField
 */
private JTextField getJTextFieldAddress() {
	if (jTextFieldAddress == null) {
		jTextFieldAddress = new JTextField();
		jTextFieldAddress.setColumns(10);
		jTextFieldAddress.setEditable(false);
		jTextFieldAddress.setBackground(Color.WHITE);
	}
	return jTextFieldAddress;
}



/**
 * This method initializes jTextFieldAccessType
 *
 * @return javax.swing.JTextField
 */
private JTextField getJTextFieldAccessType() {
	if (jTextFieldAccessType == null) {
		jTextFieldAccessType = new JTextField();
		jTextFieldAccessType.setColumns(5);
		jTextFieldAccessType.setEditable(false);
		jTextFieldAccessType.setBackground(Color.WHITE);
	}
	return jTextFieldAccessType;
}
/**
 * This method initializes jTextFieldAccessSize
 *
```

```
	 * @return javax.swing.JTextField
	 */
	private JTextField getJTextFieldAccessSize() {
		if (jTextFieldAccessSize == null) {
			jTextFieldAccessSize = new JTextField();
			jTextFieldAccessSize.setColumns(3);
			jTextFieldAccessSize.setEditable(false);
			jTextFieldAccessSize.setBackground(Color.WHITE);
		}
		return jTextFieldAccessSize;
	}
	/**
	 * This method initializes jMenuItemNewSim2
	 *
	 * @return javax.swing.JMenuItem
	 */
	private JMenuItem getJMenuItemNewSim2() {
		if (jMenuItemNewSim2 == null) {
			jMenuItemNewSim2 = new JMenuItem();
			jMenuItemNewSim2.setText("New SplitCache Simulation");
			jMenuItemNewSim2.addActionListener(new
java.awt.event.ActionListener() {
				public void actionPerformed(java.awt.event.ActionEvent e) {
					System.out.println("actionPerformed()"); // TODO
Auto-generated Event stub actionPerformed()
					SplitCacheDialog newSimDialog = new
SplitCacheDialog(RunProgram.this);
					newSimDialog.setVisible(true);

				}
			});
		}
		return jMenuItemNewSim2;
	}
	/**
	 * This method initializes jPanelSecundaryControls
	 *
	 * @return javax.swing.JPanel
	 */
	private JPanel getJPanelSecundaryControls() {
		if (jPanelSecundaryControls == null) {
			jPanelSecundaryControls = new JPanel();
			jPanelSecundaryControls.add(getJButtonShowNextLevelAccess(),
null);
		}
		return jPanelSecundaryControls;
	}

	/**
	 * This method initializes jButtonShowNextLevelAccess
	 *
	 * @return javax.swing.JButton
	 */
	private JButton getJButtonShowNextLevelAccess() {
		if (jButtonShowNextLevelAccess == null) {
			jButtonShowNextLevelAccess = new JButton();
			jButtonShowNextLevelAccess.setText("Show Next Level Access");
			jButtonShowNextLevelAccess
			.addActionListener(new java.awt.event.ActionListener() {
				public void actionPerformed(java.awt.event.ActionEvent e) {
					if(RunProgram.this.nextLevelDialog.isVisible()){

	RunProgram.this.nextLevelDialog.setVisible(false);

	RunProgram.this.jButtonShowNextLevelAccess.setText("Show Next Level Access");
					}else{

	RunProgram.this.nextLevelDialog.setVisible(true);

	RunProgram.this.jButtonShowNextLevelAccess.setText("Hide Next Level Access");
					}
				}

			});
		}
		return jButtonShowNextLevelAccess;
	}
```

```
/**
 * This method initializes jTextFieldWriteBacks
 *
 * @return javax.swing.JTextField
 */
private JTextField getJTextFieldWriteBacks() {
	if (jTextFieldWriteBacks == null) {
		jTextFieldWriteBacks = new JTextField();
		jTextFieldWriteBacks.setText("0");
		jTextFieldWriteBacks.setEditable(false);
		jTextFieldWriteBacks.setBackground(Color.WHITE);
	}
	return jTextFieldWriteBacks;
}

/**
 * This method initializes jTextFieldWriteThroughs
 *
 * @return javax.swing.JTextField
 */
private JTextField getJTextFieldWriteThroughs() {
	if (jTextFieldWriteThroughs == null) {
		jTextFieldWriteThroughs = new JTextField();
		jTextFieldWriteThroughs.setText("0");
		jTextFieldWriteThroughs.setEditable(false);
		jTextFieldWriteThroughs.setBackground(Color.WHITE);

	}
	return jTextFieldWriteThroughs;
}

/**
 * This method initializes jTextFieldNextLevel
 *
 * @return javax.swing.JTextField
 */
private JTextField getJTextFieldNextLevel() {
	if (jTextFieldNextLevel == null) {
		jTextFieldNextLevel = new JTextField();
		jTextFieldNextLevel.setText("0");
		jTextFieldNextLevel.setEditable(false);
		jTextFieldNextLevel.setBackground(Color.WHITE);
	}
	return jTextFieldNextLevel;
}

/**
 * This method initializes jTextFieldIAccess
 *
 * @return javax.swing.JTextField
 */
private JTextField getJTextFieldIAccess() {
	if (jTextFieldIAccess == null) {
		jTextFieldIAccess = new JTextField();
		jTextFieldIAccess.setText("0");
		jTextFieldIAccess.setEditable(false);
		jTextFieldIAccess.setBackground(Color.WHITE);
	}
	return jTextFieldIAccess;
}

/**
 * This method initializes jPanelNaoEstatisticas
 *
 * @return javax.swing.JPanel
 */
private JPanel getJPanelNaoEstatisticas() {
	if (jPanelNaoEstatisticas == null) {
		jLabel15 = new JLabel();
		jLabel15.setText("Data Writes");
		jLabel15.setHorizontalAlignment(JLabel.RIGHT);
		jLabel14 = new JLabel();
		jLabel14.setText("Data Reads");
		jLabel14.setHorizontalAlignment(JLabel.RIGHT);
		jLabel13 = new JLabel();
		jLabel13.setText("Instructions");
```

```
                        jLabel13.setHorizontalAlignment(JLabel.RIGHT);
                        jLabel12 = new JLabel();
                        jLabel12.setText("Acessos");
                        jLabel11 = new JLabel();
                        jLabel11.setText("");
                        jPanelNaoEstatisticas = new JPanel();
                        jPanelNaoEstatisticas.setLayout(new GridLayout(5,2,5,5));
                        jPanelNaoEstatisticas.add(jLabel11, null);
                        jPanelNaoEstatisticas.add(jLabel12, null);
                        jPanelNaoEstatisticas.add(jLabel13, null);
                        jPanelNaoEstatisticas.add(getJTextFieldIAccess(), null);
                        jPanelNaoEstatisticas.add(jLabel14, null);
                        jPanelNaoEstatisticas.add(getJTextFieldDRAccess(), null);
                        jPanelNaoEstatisticas.add(jLabel15, null);
                        jPanelNaoEstatisticas.add(getJTextFieldDWAccess(), null);

                }
                return jPanelNaoEstatisticas;
        }

        /**
         * This method initializes jTextFieldDRAccess
         *
         * @return javax.swing.JTextField
         */
        private JTextField getJTextFieldDRAccess() {
                if (jTextFieldDRAccess == null) {
                        jTextFieldDRAccess = new JTextField();
                        jTextFieldDRAccess.setText("0");
                        jTextFieldDRAccess.setEditable(false);
                        jTextFieldDRAccess.setBackground(Color.WHITE);
                }
                return jTextFieldDRAccess;
        }

        /**
         * This method initializes jTextFieldDWAccess
         *
         * @return javax.swing.JTextField
         */
        private JTextField getJTextFieldDWAccess() {
                if (jTextFieldDWAccess == null) {
                        jTextFieldDWAccess = new JTextField();
                        jTextFieldDWAccess.setText("0");
                        jTextFieldDWAccess.setEditable(false);
                        jTextFieldDWAccess.setBackground(Color.WHITE);
                }
                return jTextFieldDWAccess;
        }

        /**
         * This method initializes jPanelAccessDecoded
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJPanelAccessDecoded() {
                if (jPanelAccessDecoded == null) {
                        jPanelAccessDecoded = new JPanel(new BorderLayout());

                        jPanelAccessDecoded.add(getJPanel1(), BorderLayout.NORTH);
                        jPanelAccessDecoded.add(getJPanel2(), BorderLayout.CENTER);
                        jPanelAccessDecoded.add(getJPanelDecodedTop(),
java.awt.BorderLayout.SOUTH);

                }
                return jPanelAccessDecoded;
        }

        /**
         * This method initializes jTextFieldTag
         *
         * @return javax.swing.JTextField
         */
        private JTextField getJTextFieldTag() {
                if (jTextFieldTag == null) {
                        jTextFieldTag = new JTextField();
                        jTextFieldTag.setColumns(32);
```

```
                        jTextFieldTag.setEditable(false);
                        jTextFieldTag.setBackground(Color.WHITE);
                        jTextFieldTag.setFont(new Font("Courier New",Font.PLAIN, 12));


                }
                return jTextFieldTag;
        }

        /**
         * This method initializes jPanel
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJPanel2() {
                if (jPanel == null) {
                        jLabelDecimal = new JLabel();
                        jLabelDecimal.setText(" Decimal Address");
                        jLabelDecimal.setFont(new Font("Courier New",Font.PLAIN, 12));
                        jPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));

        //jPanel.setBorder(BorderFactory.createTitledBorder(BorderFactory.createLineBorde
r(Color.LIGHT_GRAY,0),"Address (decimal)"));
                        jPanel.add(getJTextFieldTag(), null);
                        jPanel.add(getJTextFieldIndex(), null);
                        jPanel.add(getJTextFieldOffset(), null);
                        jPanel.add(jLabelDecimal, null);

                }
                return jPanel;
        }

        /**
         * This method initializes jPanel1
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJPanel1() {
                if (jPanel1 == null) {
                        jLabelBinary = new JLabel();
                        jLabelBinary.setText(" Binary  Address");
                        jLabelBinary.setFont(new Font("Courier New",Font.PLAIN, 12));
                        jPanel1 = new JPanel(new FlowLayout(FlowLayout.LEFT));

                        jPanel1.add(getJTextFieldBinario(), null);
                        jPanel1.add(getJTextFieldBinario2(), null);

        //jPanel1.setBorder(BorderFactory.createTitledBorder(BorderFactory.createLineBord
er(Color.LIGHT_GRAY,0),"Address (binary)"));

                        jPanel1.add(getJTextFieldBinario3(), null);
                        jPanel1.add(jLabelBinary, null);

                }
                return jPanel1;
        }

        /**
         * This method initializes jTextFieldBinario
         *
         * @return javax.swing.JTextField
         */
        private JTextField getJTextFieldBinario() {
                if (jTextFieldBinario == null) {
                        jTextFieldBinario = new JTextField();
                        jTextFieldBinario.setColumns(32);
                        jTextFieldBinario.setEditable(false);
                        jTextFieldBinario.setBackground(Color.WHITE);
                        jTextFieldBinario.setFont(new Font("Courier New",Font.PLAIN, 12));

                }
                return jTextFieldBinario;
        }

        /**
         * This method initializes jTextFieldBinario2
         *
```

```
         * @return javax.swing.JTextField
         */
        private JTextField getJTextFieldBinario2() {
                if (jTextFieldBinario2 == null) {
                        jTextFieldBinario2 = new JTextField();
                        jTextFieldBinario2.setColumns(16);
                        jTextFieldBinario2.setEditable(false);
                        jTextFieldBinario2.setBackground(Color.WHITE);
                        jTextFieldBinario2.setFont(new Font("Courier New",Font.PLAIN,
12));
                }
                return jTextFieldBinario2;
        }

        /**
         * This method initializes jTextFieldBinario3
         *
         * @return javax.swing.JTextField
         */
        private JTextField getJTextFieldBinario3() {
                if (jTextFieldBinario3 == null) {
                        jTextFieldBinario3 = new JTextField();
                        jTextFieldBinario3.setColumns(16);
                        jTextFieldBinario3.setEditable(false);
                        jTextFieldBinario3.setBackground(Color.WHITE);
                        jTextFieldBinario3.setFont(new Font("Courier New",Font.PLAIN,
12));
                }
                return jTextFieldBinario3;
        }

        /**
         * This method initializes jTextFieldIndex
         *
         * @return javax.swing.JTextField
         */
        private JTextField getJTextFieldIndex() {
                if (jTextFieldIndex == null) {
                        jTextFieldIndex = new JTextField();
                        jTextFieldIndex.setColumns(16);
                        jTextFieldIndex.setEditable(false);
                        jTextFieldIndex.setBackground(Color.WHITE);
                        jTextFieldIndex.setFont(new Font("Courier New",Font.PLAIN, 12));
                }
                return jTextFieldIndex;
        }

        /**
         * This method initializes jTextFieldOffset
         *
         * @return javax.swing.JTextField
         */
        private JTextField getJTextFieldOffset() {
                if (jTextFieldOffset == null) {
                        jTextFieldOffset = new JTextField();
                        jTextFieldOffset.setColumns(16);
                        jTextFieldOffset.setEditable(false);
                        jTextFieldOffset.setBackground(Color.WHITE);
                        jTextFieldOffset.setFont(new Font("Courier New",Font.PLAIN, 12));

                }
                return jTextFieldOffset;
        }

        /**
         * This method initializes jPanelDecodedTop
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJPanelDecodedTop() {
                if (jPanelDecodedTop == null) {
                        jLabelBits = new JLabel();
                        jLabelBits.setText("( tag | index | offset )");
                        jLabelBits.setFont(new Font("Courier New",Font.PLAIN, 12));
                        jPanelDecodedTop = new JPanel(new FlowLayout(FlowLayout.LEFT));
                        jPanelDecodedTop.add(jLabelBits, null);
                }
```

```
		return jPanelDecodedTop;
	}

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		RunProgram application = new RunProgram();
		application.setVisible(true);



	}

	/**
	 * This is the default constructor
	 */
	public RunProgram() {
		super();
		initialize();
	}

	/**
	 * This method initializes this
	 *
	 * @return void
	 */
	private void initialize() {

		this.setVisible(false);
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		this.setJMenuBar(getJJMenuBar());
		this.setSize(800, 600);
		this.setContentPane(getJContentPane());
		this.setTitle("my Cache Simulator");

		this.nextLevelDialog = new NextLevelAccessDialog(this);
		nextLevelDialog.setVisible(false);

	}

	/**
	 * This method initializes jContentPane
	 *
	 * @return javax.swing.JPanel
	 */
	private JPanel getJContentPane() {
		if (jContentPane == null) {

			jContentPane = new JPanel();
			jContentPane.setBorder(BorderFactory.createEmptyBorder(5,5,5,5));
			jContentPane.setLayout(new BorderLayout());
			jContentPane.add(getJPanelToolbar(), java.awt.BorderLayout.NORTH);
			jContentPane.add(getJPanelLegenda(), java.awt.BorderLayout.SOUTH);
			jContentPane.add(getJPanelEstatisticas(),
java.awt.BorderLayout.EAST);
			jContentPane.add(getScrollPaneCanvas(),
java.awt.BorderLayout.CENTER);
		}
		return jContentPane;
	}

	/**
	 * This method initializes jJMenuBar
	 *
	 * @return javax.swing.JMenuBar
	 */
	private JMenuBar getJJMenuBar() {
		if (jJMenuBar == null) {
			jJMenuBar = new JMenuBar();
			jJMenuBar.add(getFileMenu());
			jJMenuBar.add(getHelpMenu());
		}
		return jJMenuBar;
	}
```

```
	/**
	 * This method initializes jMenu
	 *
	 * @return javax.swing.JMenu
	 */
	private JMenu getFileMenu() {
		if (fileMenu == null) {
			fileMenu = new JMenu();
			fileMenu.setText("File");

			fileMenu.add(getJMenuItemNewSim());

			fileMenu.add(getJMenuItemNewSim2());
			fileMenu.add(getExitMenuItem());
		}
		return fileMenu;
	}

	/**
	 * This method initializes jMenu
	 *
	 * @return javax.swing.JMenu
	 */
	private JMenu getHelpMenu() {
		if (helpMenu == null) {
			helpMenu = new JMenu();
			helpMenu.setText("Help");
			helpMenu.add(getAboutMenuItem());
		}
		return helpMenu;
	}

	/**
	 * This method initializes jMenuItem
	 *
	 * @return javax.swing.JMenuItem
	 */
	private JMenuItem getExitMenuItem() {
		if (exitMenuItem == null) {
			exitMenuItem = new JMenuItem();
			exitMenuItem.setText("Exit");
			exitMenuItem.addActionListener(new ActionListener() {
				public void actionPerformed(ActionEvent e) {
					System.exit(0);
				}
			});
		}
		return exitMenuItem;
	}

	/**
	 * This method initializes jMenuItem
	 *
	 * @return javax.swing.JMenuItem
	 */
	private JMenuItem getAboutMenuItem() {
		if (aboutMenuItem == null) {
			aboutMenuItem = new JMenuItem();
			aboutMenuItem.setText("About");
			aboutMenuItem.addActionListener(new ActionListener() {
				public void actionPerformed(ActionEvent e) {
					//new AboutDialog(RunProgram.this, "About",
true).show();
					new AboutDialog(RunProgram.this).setVisible(true);
				}
			});
		}
		return aboutMenuItem;
	}


}
```

6.2.11 SimParams

Esta classe contem os parâmetros necessários à simulação, entre os quais a cache (SplitCache ou UnifiedCache), qual o tipo de cache usado e o ficheiro com os acessos.

```
package cacheSim;

/**
 * Classe que tem os parametros obrigatório para correr a simulação
 * Capaz de dizer se a cache escolhida é do tipo Split ou Unified por exemplo.
 * @author Manuel
 *
 */
public class SimParams {
	private SplitCache splitCache;
	private UnifiedCache unifiedCache;
	private String traceFile;
	public static final int U_CACHE = 0;
	public static final int S_CACHE = 1;

	public int cacheType;


	public SimParams(SplitCache cache, String fileName){
		this.splitCache = cache;
		this.unifiedCache = null;
		this.traceFile = fileName;
		this.cacheType = S_CACHE;

	}
	public SimParams(UnifiedCache cache, String fileName){
		this.unifiedCache = cache;
		this.splitCache = null;
		this.traceFile = fileName;
		this.cacheType = U_CACHE;
	}
	public boolean isUnifiedCache(){
		if(cacheType == U_CACHE)
			return true;
		else
			return false;
	}
	public boolean isSplitCache(){
		if(cacheType == S_CACHE)
			return true;
		else
			return false;
	}
	public SplitCache getSplitCache(){
		return this.splitCache;
	}
	public UnifiedCache getUnifiedCache(){
		return this.unifiedCache;
	}

	public String getTraceFile(){
		return this.traceFile;
	}


}
```

6.2.12 SimulationThread

É uma das classes mais importantes do programa, executa a simulação e actualiza a interface gráfica (RunProgram), por sua vez os inputs do utilizador na interface são passados para esta thread controlando assim o modo de vizualização, velocidade, etc.

```
package cacheSim;

import java.awt.Dimension;
import java.awt.Point;


/**
 * Classe que representa a thread de simulação. Tem referencias à interface principal,
 * para a poder actualizar.
 * Por usa vez a interface principal pode alterar o estado desta thread.
 * @author Manuel
 *
 */
public class SimulationThread extends Thread{

	private RunProgram window;
	private boolean pause;
	private boolean stepinto;
	private boolean playmode;
	private int sleepTime;
	private SimParams params;
	private final int  SILENT_MODE = 200;
	MemAccessFile maf;

	private boolean terminateThread;
	public SimulationThread(RunProgram window, SimParams simParams){
		this.window = window;
		this.pause = true;
		this.playmode = false;
		this.stepinto = false;
		this.sleepTime = window.getSleepTime();
		terminateThread = false;

		setParams(simParams);
		try{
			maf = new MemAccessFile(simParams.getTraceFile());
		}catch(Exception e){

		}
	}


	public void setParams(SimParams simParams){
		this.params = simParams;
		window.myCanvas.setParams(simParams);
		this.setPause(true);
	}

	public void setPlayMode(boolean mode){
		this.playmode = mode;
	}

	public void run () {

		while (!terminateThread) {
			/* Alter the animation control variables for the
			 next paint */
			if(this.playmode && this.sleepTime <= SILENT_MODE && !this.pause){

				try{
					if(maf.hasMore()){
						window.resetStatistcsBackColor();
```

```
                                        MemAccess ma;

                                        ma = maf.getNext();



                                        window.setNextAccess(ma);
                                        if(params.isSplitCache()){
                                                if(ma.isInstructionAccess()){

                                                        window.setNextDecodedAddress(
        params.getSplitCache().getICache().getTag(ma),
        params.getSplitCache().getICache().getIndex(ma),
        params.getSplitCache().getICache().getOffset(ma),
        params.getSplitCache().getICache().getTagBits(),
        params.getSplitCache().getICache().getIndexBits(),
        params.getSplitCache().getICache().getOffsetBits()
                                                        );
                                                }else{
                                                        window.setNextDecodedAddress(
        params.getSplitCache().getDCache().getTag(ma),
        params.getSplitCache().getDCache().getIndex(ma),
        params.getSplitCache().getDCache().getOffset(ma),
        params.getSplitCache().getDCache().getTagBits(),
        params.getSplitCache().getDCache().getIndexBits(),
        params.getSplitCache().getDCache().getOffsetBits()
                                                        );
                                                }

        params.getSplitCache().doMemAccess(ma);

                                                //regista os acessos ao nivel
seguinte de memória
                                                if(ma.isInstructionAccess()){
                                                        MemAccess [] array =
params.getSplitCache().getICache().getNextLevelAccess();
                                                        if(array != null){
                                                                for(int i=0;
i<array.length; i++){
        window.addNextLevelAccess(array[i]); //ERRADO
                                                                }
                                                        }
                                                }else{
                                                        MemAccess [] array =
params.getSplitCache().getDCache().getNextLevelAccess();
                                                        if(array != null){
                                                                for(int i=0;
i<array.length; i++){
        window.addNextLevelAccess(array[i]); //ERRADO
                                                                }
                                                        }
                                                }

                                        }else if(params.isUnifiedCache()){
                                                window.setNextDecodedAddress(
        params.getUnifiedCache().getUCache().getTag(ma),
```

```
        params.getUnifiedCache().getUCache().getIndex(ma),

        params.getUnifiedCache().getUCache().getOffset(ma),

        params.getUnifiedCache().getUCache().getTagBits(),

        params.getUnifiedCache().getUCache().getIndexBits(),

        params.getUnifiedCache().getUCache().getOffsetBits()
                                                        );


        params.getUnifiedCache().doMemAccess(ma);

                                                        MemAccess [] array =
params.getUnifiedCache().getUCache().getNextLevelAccess();
                                                        if(array != null){
                                                                for(int i=0; i<array.length;
i++){

        window.addNextLevelAccess(array[i]); //ERRADO
                                                                }
                                                        }


                                                }
                                                showStatistcs();
                                        }
                                }
                                catch(Exception e){
                                        System.out.println("Access com o formato errado:
"+e.getMessage());
                                        WarningDialog dial = new WarningDialog(window,
"Error", "Bad Access Format - Ignoring access!\n"+e.getMessage());
                                        dial.setVisible(true);
                                }

                        }else{

                                if(!pause || this.stepinto){
                                        this.stepinto = false;
                                        window.myCanvas.my_x++;


                                        try{
                                                if(maf.hasMore()){

                                                        window.resetStatistcsBackColor();
                                                        MemAccess ma = maf.getNext();







                                                        window.setNextAccess(ma);


        /************************************************/
                                                        /*                SPLIT CACHE
*/
        /************************************************/
                                                        if(params.isSplitCache()){

                                                                if(ma.isInstructionAccess()){



        window.setNextDecodedAddress(

        params.getSplitCache().getICache().getTag(ma),

        params.getSplitCache().getICache().getIndex(ma),

        params.getSplitCache().getICache().getOffset(ma),
```

```
        params.getSplitCache().getICache().getTagBits(),
        params.getSplitCache().getICache().getIndexBits(),
        params.getSplitCache().getICache().getOffsetBits()
                                                                );
                                                        }else{
        window.setNextDecodedAddress(
        params.getSplitCache().getDCache().getTag(ma),
        params.getSplitCache().getDCache().getIndex(ma),
        params.getSplitCache().getDCache().getOffset(ma),
        params.getSplitCache().getDCache().getTagBits(),
        params.getSplitCache().getDCache().getIndexBits(),
        params.getSplitCache().getDCache().getOffsetBits()
                                                                );
                                                        }
                                                        if(ma.isInstructionAccess()){
        //params.getSplitCache().getAccessBlockIndex()
                                                                int scroll_y =
params.getSplitCache().getSetBlockIndex(ma)*window.myCanvas.getBlockByteDrawSize().y +
window.myCanvas.getCache1Pos().y;

        //window.myCanvas.setBackground(new Color(200,255,200));
        window.myCanvas.setBlockSelected(window.myCanvas.getCache1Pos().x+MainCanvas.X_IN
D_BLOCK, scroll_y, 0, params.getSplitCache().getSetIndex(ma));
                                                                Point p =
window.scrollPaneCanvas.getScrollPosition();
                                                                Dimension d =
window.scrollPaneCanvas.getSize();
                                                                if(!(scroll_y > p.y &&
scroll_y < (p.y + d.height)))
        window.scrollPaneCanvas.setScrollPosition(0,scroll_y-100);
        window.myCanvas.repaint();
                                                                //para a simulação até
step-into
                                                                do{
        Thread.sleep(sleepTime);
                                                                }while(!this.stepinto
&& pause && !terminateThread);
                                                                this.stepinto = false;

        window.myCanvas.removeBlockSelected();

        params.getSplitCache().doMemAccess(ma);
                                                                MemAccess [] array =
params.getSplitCache().getICache().getNextLevelAccess();
                                                                if(array != null){
                                                                        for(int i=0;
i<array.length; i++){
        window.addNextLevelAccess(array[i]);
                                                                        }
                                                                }
                                                        }else{
//
        params.getSplitCache().getAccessBlockIndex()
```

```
                                                                int scroll_y =
params.getSplitCache().getSetBlockIndex(ma)*window.myCanvas.getBlockByteDrawSize().y +
window.myCanvas.getCache2Pos().y;


        //window.myCanvas.setBackground(new Color(200,200,255));

        window.myCanvas.setBlockSelected(window.myCanvas.getCache2Pos().x+MainCanvas.X_IN
D_BLOCK, scroll_y, 1, params.getSplitCache().getSetIndex(ma));
                                                                Point p =
window.scrollPaneCanvas.getScrollPosition();
                                                                Dimension d =
window.scrollPaneCanvas.getSize();
                                                                if(!(scroll_y > p.y &&
scroll_y < (p.y + d.height)))
        window.scrollPaneCanvas.setScrollPosition(0,scroll_y-100);

        window.myCanvas.repaint();

                                                                do{

        Thread.sleep(sleepTime);
                                                                }while(!this.stepinto
&& pause && !terminateThread);
                                                                this.stepinto = false;


        window.myCanvas.removeBlockSelected();


        params.getSplitCache().doMemAccess(ma);

                                                                //após fazer o acesso
ve se houveram acessos ao próximo nivel de memória
                                                                MemAccess [] array =
params.getSplitCache().getDCache().getNextLevelAccess();
                                                                if(array != null){
                                                                        for(int i=0;
i<array.length; i++){

        window.addNextLevelAccess(array[i]);
                                                                        }
                                                                }

                                                        }

                                                    }else


        /***********************************************/
                                                        /*              UNIFIED
CACHE                   */
        /***********************************************/

                                                        if(params.isUnifiedCache()){


        window.setNextDecodedAddress(

        params.getUnifiedCache().getUCache().getTag(ma),

        params.getUnifiedCache().getUCache().getIndex(ma),

        params.getUnifiedCache().getUCache().getOffset(ma),

        params.getUnifiedCache().getUCache().getTagBits(),

        params.getUnifiedCache().getUCache().getIndexBits(),

        params.getUnifiedCache().getUCache().getOffsetBits()
                                                                );
```

```
        //params.getSplitCache().getAccessBlockIndex()
                                                                int scroll_y =
params.getUnifiedCache().getSetBlockIndex(ma)*window.myCanvas.getBlockByteDrawSize().y +
window.myCanvas.getCache1Pos().y;


        //window.myCanvas.setBackground(new Color(200,255,200));

        window.myCanvas.setBlockSelected(window.myCanvas.getCache1Pos().x+MainCanvas.X_IN
D_BLOCK, scroll_y, 0, params.getUnifiedCache().getSetIndex(ma));
                                                                Point p =
window.scrollPaneCanvas.getScrollPosition();
                                                                Dimension d =
window.scrollPaneCanvas.getSize();
                                                                if(!(scroll_y > p.y &&
scroll_y < (p.y + d.height)))
        window.scrollPaneCanvas.setScrollPosition(0,scroll_y-100);

        window.myCanvas.repaint();

                                                                //para a simulação até
step-into
                                                                do{

        Thread.sleep(sleepTime);
                                                                }while(!this.stepinto
&& pause && !terminateThread);
                                                                this.stepinto = false;


        window.myCanvas.removeBlockSelected();


        params.getUnifiedCache().doMemAccess(ma);

                                                                MemAccess [] array =
params.getUnifiedCache().getUCache().getNextLevelAccess();
                                                                if(array != null){
                                                                        for(int i=0;
i<array.length; i++){

        window.addNextLevelAccess(array[i]); //ERRADO
                                                                        }
                                                                }


                                                                /*************

window.setNextTag(params.getSplitCache().getICache().getTag(ma));

                                                                 int scroll_y =
params.getUnifiedCache().getSetIndex(ma)*window.myCanvas.getBlockByteDrawSize().y;

window.setScrollPosition(0,scroll_y);

Thread.sleep(sleepTime);


params.getUnifiedCache().doMemAccess(ma);

                                                                 ***********/
                                                        }

                                                //tem que estar aqui para não fazer
repaint quando não deve
                                                window.myCanvas.repaint();
                                                showStatistcs();

                                        }else{

                                                this.setPause(true);
                                                window.setStatusMessage("Simulation
finished!");
                                        }
                                }catch(Exception e){
```

```
                                        System.out.println("Thread > Acesso com o
formato errado: "+e.getMessage());
                                        WarningDialog dial = new
WarningDialog(window, "Error", "Bad Access Format - Ignoring access!\n"+e.getMessage());
                                        dial.setVisible(true);
                                }

                        }else{
                                window.setStatusMessage("Paused...");
                        }
                }

                //adormece por um bocado...
                try {

                        Thread.sleep(sleepTime);
                }
                catch (InterruptedException ex) {
                }



        }
    }

    private void showStatistcs(){
        if(params.isSplitCache()){
                SplitCache x = params.getSplitCache();
                //(long acessos, long hits, long misses, long wr_throughs, long
wr_back, long next_level, long i_access, long dr_access, long dw_access){
                window.setStatistcs(x.getNumAccess(), x.getNumHits(),
x.getNumMisses(), x.getNumWriteThroughs(), x.getNumWriteBacks(),
x.getNumNextLevelAccess(), x.getNumIAccess(), x.getNumDRAccess(), x.getNumDWAccess());

        }
        if(params.isUnifiedCache()){
                UnifiedCache x = params.getUnifiedCache();
                window.setStatistcs(x.getNumAccess(), x.getNumHits(),
x.getNumMisses(), x.getNumWriteThroughs(), x.getNumWriteBacks(),
x.getNumNextLevelAccess(), x.getNumIAccess(), x.getNumDRAccess(), x.getNumDWAccess());

        }
    }

    public void setPause(boolean b){
        this.pause = b;
        this.playmode = false;
        if(pause)
                window.setStatusMessage("Paused...");
        else
                window.setStatusMessage("Running...");
    }

    public void setStepInto(boolean b){
        this.pause = true;
        this.playmode = false;
        this.stepinto = b;
        if(this.stepinto)
                window.setStatusMessage("Step Into Enabled...");
    }

    public void setSpeed(int sleep){
        this.sleepTime = sleep;
        if(this.sleepTime <= SILENT_MODE){
                window.setStatusMessage("Silent Mode...");
        }
    }

    public void pauseAnimation(){
        pause = !pause;
        if(pause)
                window.setStatusMessage("Paused...");
        else
                window.setStatusMessage("Running...");


    }
```

```
	public void stopThread(){
		this.terminateThread = true;
	}

}
```

6.2.13 SplitCache

Esta classe representa uma cache do tipo separada, tem duas variáveis que representam uma cache de instruções e uma cache de dados. A construção desta classe tem como propósito tornar a utilização mais transparente para o programador. Assim agrupa-se as duas caches numa só classe evitando refazer certas partes do código.

```
package cacheSim;

/**
 * Representa uma cache separada, cache de dados mais cache de instruções.
 * Esta classe é executavel para poder correr simulações pesadas.
 * @author Manuel Maia
 *
 */
public class SplitCache {
	private Cache iCache;
	private Cache dCache;
	private int lastUsedCache; //0 para iCache 1 para dCache
	private MemAccess lastMemAccess;


	public SplitCache(Cache iCache, Cache dCache)
	throws Exception
	{
		if(!iCache.isInstrCache())
			throw new Exception("iCache isn't an instruction cache!");
		this.iCache = iCache;
		this.dCache = dCache;

	}
	public MemAccess getLastMemAccess(){
		return this.lastMemAccess;
	}

	public long getNumAccess(){
		return iCache.getNumAccess() + dCache.getNumAccess();
	}
	public long getNumHits(){
		return iCache.getNumHits() + dCache.getNumHits();
	}
	public long getNumMisses(){
		return iCache.getNumMisses() + dCache.getNumMisses();
	}
	public long getNumWriteThroughs(){
		return dCache.getNumWriteThroughs() + iCache.getNumWriteThroughs();
	}
	public long getNumWriteBacks(){
		return dCache.getNumWriteBacks()+iCache.getNumWriteBacks();
	}
	public long getNumIAccess(){
		return dCache.getNumIAccess() + iCache.getNumIAccess();
	}
	public long getNumDRAccess(){
		return dCache.getNumDRAccess() + iCache.getNumDRAccess();
	}
	public long getNumDWAccess(){
		return dCache.getNumDWAccess() + iCache.getNumDWAccess();
	}
	public long getNumNextLevelAccess(){
		return dCache.getNumNextLevelAccess() + iCache.getNumNextLevelAccess();
	}

	//retorna o indice do ultimo bloco a ser acedido
	public int getLastAccessBlockIndex(){
		if(this.lastUsedCache == 0)
			return iCache.getLastAccessBlockIndex();
		else
			return dCache.getLastAccessBlockIndex();
```

```
	}

	public int getLastUsedCache(){
		return this.lastUsedCache;
	}

	public int getSetIndex(MemAccess ma){
		if(ma.isInstructionAccess())
			return iCache.getSetIndex(ma);
		else
			return dCache.getSetIndex(ma);
	}

//	retorna o index do bloco ao qual pertence o primeiro set
	public int getSetBlockIndex(MemAccess ma){
		if(ma.isInstructionAccess())
			return (iCache.getSetIndex(ma) * iCache.getAssociatividade());
		else
			return (dCache.getSetIndex(ma) * dCache.getAssociatividade());
	}

	public void doMemAccess(MemAccess ma)
	throws Cache.CacheFormatException
	{

		this.lastMemAccess = ma.getCopy();


		if(ma.isInstructionAccess()){
			iCache.doMemAccess(ma);
			this.lastUsedCache = 0;
		}
		else{
			dCache.doMemAccess(ma);
			this.lastUsedCache = 1;
		}
	}

	public Cache getICache(){
		return iCache;
	}
	public Cache getDCache(){
		return dCache;
	}


	public static void main(String args[]){
		try{

			if(args.length != 6){
				System.out.println("Wrong usage! try:");
				System.out.println("java cacheSim.SplitCache"+
				"\n\t<cache size (KB)> \n\t<block size (bytes)>
\n\t<associativity> \n\t<LRU | FIFO | RANDOM> \n\t<WR_BACK | WR_THROUGH>\n\t<trace
file>");
				System.exit(0);
			}

			int size = Integer.parseInt(args[0]);
			int block = Integer.parseInt(args[1]);
			int assoc = Integer.parseInt(args[2]);
			int sub = -1;
			int wr = -1;
			if(args[3].compareTo("LRU") == 0)
				sub = Cache.S_LRU;
			if(args[3].compareTo("FIFO") == 0)
				sub = Cache.S_FIFO;
			if(args[3].compareTo("RANDOM") == 0)
				sub = Cache.S_RANDOM;

			if(args[4].compareTo("WR_BACK") == 0)
				wr = Cache.WR_BACK_ALLOCATE;
			if(args[4].compareTo("WR_THROUGH") == 0)
				wr = Cache.WR_THROUGH_NO_ALLOCATE;
```

```
                        Cache cache1 = new Cache(size, block, assoc, sub,
Cache.WR_INSTR_CACHE, true);
                        Cache cache2 = new Cache(size, block, assoc, sub, wr, true);
                        SplitCache sCache = new SplitCache(cache1, cache2);


                        MemAccessFile maf = new MemAccessFile(args[5]);
                        int i;
                        for(i=0; maf.hasMore(); i++){
                                try{
                                        MemAccess ma = maf.getNext();
                                        //System.out.println(ma.toString());
                                        sCache.doMemAccess(ma);
                                }catch(Exception e){
                                        System.out.println("Bad Access Format: "+
e.getMessage());
                                }
                        }
                        maf.close();
                        //System.out.println("Read "+i+" instructions!");

                        System.out.println("----------------------------\nInstruction
Cache Statistics\n----------------------------");
                        cache1.printStatistcs();
                        System.out.println("----------------------------\nData Cache
Statistics\n----------------------------");
                        cache2.printStatistcs();

                }catch(Exception e){
                        e.printStackTrace();
                }

        }

}
```

6.2.14 SplitCacheDialog

Esta classe é uma janela de diálogo onde o utilizador altera as características da cache e inicia à simulação. A maior parte do código é gerada pela ferramenta de desenvolvimento.

```
package cacheSim;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.GridLayout;


import javax.swing.BorderFactory;
import javax.swing.JFileChooser;
import javax.swing.JPanel;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JCheckBox;

/**
 * Representa a janela de dialogo onde o utilizador insere os dados da SplitCache e
 * o ficheiro com os acessos.
 *
 * @author Manuel
 *
 */
public class SplitCacheDialog extends JDialog {
	public static final long serialVersionUID=0;

	private JPanel jContentPane = null;

	private JPanel jPanelCenter = null;

	private JPanel jPanel = null;

	private JLabel jLabel = null;

	private JTextField jTextFieldTraceFile = null;

	private JButton jButtonOpenFile = null;

	private final Dimension labelDim = new Dimension(100,25);

	private JPanel jPanel12 = null;

	private JButton jButtonCancel = null;

	private JButton jButtonOK = null;

	private RunProgram mainFrame;

	private JPanel jPanel13 = null;

	private JPanel jPanel1 = null;

	private JPanel jPanel2 = null;

	private JPanel jPanel3 = null;

	private JLabel jLabel1 = null;

	private JTextField jTextField = null;

	private JLabel jLabel2 = null;

	private JPanel jPanel4 = null;
```

```
private JPanel jPanel5 = null;
private JLabel jLabel3 = null;
private JTextField jTextField1 = null;
private JLabel jLabel4 = null;
private JPanel jPanel6 = null;
private JPanel jPanel7 = null;
private JLabel jLabel5 = null;
private JTextField jTextField2 = null;
private JLabel jLabel6 = null;
private JPanel jPanel8 = null;
private JPanel jPanel9 = null;
private JLabel jLabel7 = null;
private JComboBox jComboBoxReplacePolicy = null;
private JPanel jPanel10 = null;
private JPanel jPanel11 = null;
private JLabel jLabel8 = null;
private JComboBox jComboBoxWritePolicy = null;
private JPanel jPanel14 = null;
private JPanel jPanel15 = null;
private JPanel jPanel16 = null;
private JLabel jLabel9 = null;
private JTextField jTextField3 = null;
private JLabel jLabel10 = null;
private JPanel jPanel17 = null;
private JPanel jPanel18 = null;
private JLabel jLabel11 = null;
private JTextField jTextField4 = null;
private JLabel jLabel12 = null;
private JPanel jPanel19 = null;
private JPanel jPanel20 = null;
private JLabel jLabel13 = null;
private JTextField jTextField5 = null;
private JLabel jLabel14 = null;
private JPanel jPanel21 = null;
private JPanel jPanel22 = null;
private JLabel jLabel15 = null;
private JComboBox jComboBoxReplacePolicyICache = null;
private JPanel jPanel23 = null;
private JPanel jPanel24 = null;
```

```
private JLabel jLabel16 = null;

private JComboBox jComboBoxWritePolicyICache = null;

private JCheckBox jCheckBoxNextLevelReg = null;

/**
 * This is the default constructor
 */
public SplitCacheDialog(RunProgram frame) {
	super(frame, true);
	initialize();
	this.setTitle("New Simulation...");
	this.setSize(450,575);
	this.setResizable(false);
	this.setLocationRelativeTo(null);
	mainFrame = frame;
}

/**
 * This method initializes this
 *
 * @return void
 */
private void initialize() {

	//this.setPreferredSize(new Dimension(400,400));
	this.setContentPane(getJContentPane());
}

/**
 * This method initializes jContentPane
 *
 * @return javax.swing.JPanel
 */
private JPanel getJContentPane() {
	if (jContentPane == null) {
		jContentPane = new JPanel();
		jContentPane.setLayout(new BorderLayout());
		jContentPane.add(getJPanelCenter(), java.awt.BorderLayout.CENTER);

jContentPane.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));

	}
	return jContentPane;
}

/**
 * This method initializes jPanelCenter
 *
 * @return javax.swing.JPanel
 */
private JPanel getJPanelCenter() {
	if (jPanelCenter == null) {
		jPanelCenter = new JPanel(new BorderLayout());
		jPanelCenter.add(getJPanel(), BorderLayout.NORTH);
		jPanelCenter.add(getJPanel13(), java.awt.BorderLayout.CENTER);
		jPanelCenter.add(getJPanel12(), java.awt.BorderLayout.SOUTH);


	}
	return jPanelCenter;
}

/**
 * This method initializes jPanel
 *
 * @return javax.swing.JPanel
 */
private JPanel getJPanel() {
	if (jPanel == null) {
		jLabel = new JLabel();
		jLabel.setText("Trace File");
		jPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
		jPanel.add(jLabel, null);
		jPanel.add(getJTextFieldTraceFile(), null);
```

```
                        jPanel.add(getJButtonOpenFile(), null);


                }
                return jPanel;
        }

        /**
         * This method initializes jTextFieldTraceFile
         *
         * @return javax.swing.JTextField
         */
        private JTextField getJTextFieldTraceFile() {
                if (jTextFieldTraceFile == null) {
                        jTextFieldTraceFile = new JTextField();
                        jTextFieldTraceFile.setColumns(20);
                        jTextFieldTraceFile.setEditable(false);
                        jTextFieldTraceFile.setBackground(new Color(250,250,250));
                        jTextFieldTraceFile.setText("");
                }
                return jTextFieldTraceFile;
        }

        /**
         * This method initializes jButtonOpenFile
         *
         * @return javax.swing.JButton
         */
        private JButton getJButtonOpenFile() {
                if (jButtonOpenFile == null) {
                        jButtonOpenFile = new JButton();
                        jButtonOpenFile.setText("...");

                        jButtonOpenFile.addActionListener(new
java.awt.event.ActionListener() {
                                public void actionPerformed(java.awt.event.ActionEvent e) {
                                        System.out.println("actionPerformed()"); // TODO
Auto-generated Event stub actionPerformed()

                                        JFileChooser chooser = new
JFileChooser(System.getProperty("user.dir"));
                                        // Note: source for ExampleFileFilter can be found
in FileChooserDemo,
                                        // under the demo/jfc directory in the Java 2 SDK,
Standard Edition.
                                        int returnVal =
chooser.showOpenDialog(SplitCacheDialog.this);
                                        if(returnVal == JFileChooser.APPROVE_OPTION) {
                                                System.out.println("You chose to open this
file: " +

        chooser.getSelectedFile().getName());

        System.out.println("File="+chooser.getSelectedFile().getAbsolutePath());

        SplitCacheDialog.this.jTextFieldTraceFile.setText(chooser.getSelectedFile().getAb
solutePath());
                                        }
                                }
                        });
                }
                return jButtonOpenFile;
        }

        /**
         * This method initializes jPanel12
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJPanel12() {
                if (jPanel12 == null) {
                        jPanel12 = new JPanel(new FlowLayout(FlowLayout.RIGHT));
                        jPanel12.add(getJCheckBoxNextLevelReg(), null);
                        jPanel12.add(getJButtonCancel(), null);
                        jPanel12.add(getJButtonOK(), null);

                }
```

```
		return jPanel12;
	}

	/**
	 * This method initializes jButtonCancel
	 *
	 * @return javax.swing.JButton
	 */
	private JButton getJButtonCancel() {
		if (jButtonCancel == null) {
			jButtonCancel = new JButton();
			jButtonCancel.setText("Cancel");
			jButtonCancel.setPreferredSize(labelDim);
			jButtonCancel.addActionListener(new
java.awt.event.ActionListener() {
				public void actionPerformed(java.awt.event.ActionEvent e) {
					System.out.println("simulationCancel()"); // TODO
Auto-generated Event stub actionPerformed()
					SplitCacheDialog.this.setVisible(false);
				}
			});
		}
		return jButtonCancel;
	}

	/**
	 * This method initializes jButtonOK
	 *
	 * @return javax.swing.JButton
	 */
	private JButton getJButtonOK() {
		if (jButtonOK == null) {
			jButtonOK = new JButton();
			jButtonOK.setText("OK");
			jButtonOK.setPreferredSize(labelDim);
			jButtonOK.addActionListener(new java.awt.event.ActionListener() {
				public void actionPerformed(java.awt.event.ActionEvent e) {

					try{
						int i_kb =
Integer.parseInt(jTextField3.getText());
						int i_sets =
Integer.parseInt(jTextField4.getText());
						int i_block =
Integer.parseInt(jTextField5.getText());
						int i_sub =
((MyItem)jComboBoxReplacePolicyICache.getSelectedItem()).value;

						int d_kb =
Integer.parseInt(jTextField.getText());
						int d_sets =
Integer.parseInt(jTextField1.getText());
						int d_block =
Integer.parseInt(jTextField2.getText());
						int d_wr =
((MyItem)jComboBoxWritePolicy.getSelectedItem()).value;
						int d_sub =
((MyItem)jComboBoxReplacePolicy.getSelectedItem()).value;

						Cache iCache = new Cache(i_kb, i_block,
i_sets, i_sub, Cache.WR_INSTR_CACHE, true);
						Cache dCache = new Cache(d_kb, d_block,
d_sets, d_sub, d_wr,true);
						SplitCache sCache = new SplitCache(iCache,
dCache);
						//SimParams simParams = new SimParams();

	if(jTextFieldTraceFile.getText().compareTo("") == 0){
							WarningDialog wd = new
WarningDialog(SplitCacheDialog.this, "Warning", "\nYou have to select a compatible trace
file!\n\nUse the '...' button to find it.");
							wd.setVisible(true);
						}else{

	mainFrame.setNextLevelRegistry(jCheckBoxNextLevelReg.isSelected());
							mainFrame.setupSimThread(new
SimParams(sCache,jTextFieldTraceFile.getText()));
```

```
                                                mainFrame.setReadyStatus(true);

        SplitCacheDialog.this.setVisible(false);
                                                }
                                        }catch(Exception exc){
                                                WarningDialog wd = new
WarningDialog(SplitCacheDialog.this, "Erro", exc.toString());
                                                wd.setVisible(true);
                                        }

                                }
                        });
                }
                return jButtonOK;
        }

        /**
         * This method initializes jPanel13
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJPanel13() {
                if (jPanel13 == null) {
                        jPanel13 = new JPanel(new BorderLayout());
                        jPanel13.add(getJPanel1(), BorderLayout.NORTH);
                        jPanel13.add(getJPanel14(), java.awt.BorderLayout.SOUTH);
                }
                return jPanel13;
        }

        /**
         * This method initializes jPanel1
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJPanel1() {
                if (jPanel1 == null) {
                        jPanel1 = new JPanel(new GridLayout(5, 1, 5, 5));

        jPanel1.setBorder(BorderFactory.createTitledBorder(BorderFactory.createLineBorder
(Color.BLACK, 1), "Data Cache"));
                        jPanel1.add(getJPanel2(), null);
                        jPanel1.add(getJPanel4(), null);
                        jPanel1.add(getJPanel6(), null);
                        jPanel1.add(getJPanel8(), null);
                        jPanel1.add(getJPanel10(), null);
                }
                return jPanel1;
        }

        /**
         * This method initializes jPanel2
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJPanel2() {
                if (jPanel2 == null) {
                        jLabel2 = new JLabel();
                        jLabel2.setText("(kilobytes)");
                        jPanel2 = new JPanel(new FlowLayout(FlowLayout.LEFT));
                        jPanel2.add(getJPanel3(), null);
                        jPanel2.add(getJTextField(), null);
                        jPanel2.add(jLabel2, null);
                }
                return jPanel2;
        }

        /**
         * This method initializes jPanel3
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJPanel3() {
                if (jPanel3 == null) {
                        jLabel1 = new JLabel();
                        jLabel1.setText("size");
                        jPanel3 = new JPanel();
```

```
                jPanel3.setPreferredSize(labelDim);
                jPanel3.add(jLabel1, null);
        }
        return jPanel3;
}

/**
 * This method initializes jTextField
 *
 * @return javax.swing.JTextField
 */
private JTextField getJTextField() {
        if (jTextField == null) {
                jTextField = new JTextField();
                jTextField.setColumns(10);
                jTextField.setText("16");
        }
        return jTextField;
}

/**
 * This method initializes jPanel4
 *
 * @return javax.swing.JPanel
 */
private JPanel getJPanel4() {
        if (jPanel4 == null) {
                jLabel4 = new JLabel();
                jLabel4.setText("(number of blocks per set)");
                jPanel4 = new JPanel(new FlowLayout(FlowLayout.LEFT));
                jPanel4.add(getJPanel5(), null);
                jPanel4.add(getJTextField1(), null);
                jPanel4.add(jLabel4, null);
        }
        return jPanel4;
}

/**
 * This method initializes jPanel5
 *
 * @return javax.swing.JPanel
 */
private JPanel getJPanel5() {
        if (jPanel5 == null) {
                jLabel3 = new JLabel();
                jLabel3.setText("associativity");
                jPanel5 = new JPanel();
                jPanel5.setPreferredSize(labelDim);
                jPanel5.add(jLabel3, null);
        }
        return jPanel5;
}

/**
 * This method initializes jTextField1
 *
 * @return javax.swing.JTextField
 */
private JTextField getJTextField1() {
        if (jTextField1 == null) {
                jTextField1 = new JTextField();
                jTextField1.setColumns(10);
                jTextField1.setText("2");
        }
        return jTextField1;
}

/**
 * This method initializes jPanel6
 *
 * @return javax.swing.JPanel
 */
private JPanel getJPanel6() {
        if (jPanel6 == null) {
                jLabel6 = new JLabel();
                jLabel6.setText("(bytes)");
                jPanel6 = new JPanel(new FlowLayout(FlowLayout.LEFT));
```

```
                jPanel6.add(getJPanel7(), null);
                jPanel6.add(getJTextField2(), null);
                jPanel6.add(jLabel6, null);
        }
        return jPanel6;
}

/**
 * This method initializes jPanel7
 *
 * @return javax.swing.JPanel
 */
private JPanel getJPanel7() {
        if (jPanel7 == null) {
                jLabel5 = new JLabel();
                jLabel5.setText("block size");
                jPanel7 = new JPanel();
                jPanel7.setPreferredSize(labelDim);
                jPanel7.add(jLabel5, null);
        }
        return jPanel7;
}

/**
 * This method initializes jTextField2
 *
 * @return javax.swing.JTextField
 */
private JTextField getJTextField2() {
        if (jTextField2 == null) {
                jTextField2 = new JTextField();
                jTextField2.setColumns(10);
                jTextField2.setText("64");
        }
        return jTextField2;
}

/**
 * This method initializes jPanel8
 *
 * @return javax.swing.JPanel
 */
private JPanel getJPanel8() {
        if (jPanel8 == null) {
                jPanel8 = new JPanel(new FlowLayout(FlowLayout.LEFT));
                jPanel8.add(getJPanel9(), null);
                jPanel8.add(getJComboBoxReplacePolicy(), null);
        }
        return jPanel8;
}

/**
 * This method initializes jPanel9
 *
 * @return javax.swing.JPanel
 */
private JPanel getJPanel9() {
        if (jPanel9 == null) {
                jLabel7 = new JLabel();
                jLabel7.setText("replace policy");
                jPanel9 = new JPanel();
                jPanel9.setPreferredSize(labelDim);
                jPanel9.add(jLabel7, null);
        }
        return jPanel9;
}

/**
 * This method initializes jComboBox
 *
 * @return javax.swing.JComboBox
 */
private JComboBox getJComboBoxReplacePolicy() {
        if (jComboBoxReplacePolicy == null) {
                jComboBoxReplacePolicy = new JComboBox();
                jComboBoxReplacePolicy.addItem(new MyItem("LRU", Cache.S_LRU));
```

```
                        jComboBoxReplacePolicy.addItem(new MyItem("Random",
Cache.S_RANDOM));
                        jComboBoxReplacePolicy.addItem(new MyItem("Fifo", Cache.S_FIFO));

                }
                return jComboBoxReplacePolicy;
        }

        /**
         * This method initializes jPanel10
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJPanel10() {
                if (jPanel10 == null) {
                        jPanel10 = new JPanel(new FlowLayout(FlowLayout.LEFT));
                        jPanel10.add(getJPanel11(), null);
                        jPanel10.add(getJComboBoxWritePolicy(), null);
                }
                return jPanel10;
        }

        /**
         * This method initializes jPanel11
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJPanel11() {
                if (jPanel11 == null) {
                        jLabel8 = new JLabel();
                        jLabel8.setText("write policy");
                        jPanel11 = new JPanel();
                        jPanel11.setPreferredSize(labelDim);
                        jPanel11.add(jLabel8, null);
                }
                return jPanel11;
        }

        /**
         * This method initializes jComboBox1
         *
         * @return javax.swing.JComboBox
         */
        private JComboBox getJComboBoxWritePolicy() {
                if (jComboBoxWritePolicy == null) {
                        jComboBoxWritePolicy = new JComboBox();
                        jComboBoxWritePolicy.addItem(new MyItem("write-back + write-
allocate",Cache.WR_BACK_ALLOCATE));
                        jComboBoxWritePolicy.addItem(new MyItem("write-through + write-no-
allocate",Cache.WR_THROUGH_NO_ALLOCATE));
                }
                return jComboBoxWritePolicy;
        }

        /**
         * This method initializes jPanel14
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJPanel14() {
                if (jPanel14 == null) {
                        jPanel14 = new JPanel(new GridLayout(5, 1, 5, 5));

        jPanel14.setBorder(BorderFactory.createTitledBorder(BorderFactory.createLineBorde
r(Color.BLACK, 1), "Instruction Cache"));
                        jPanel14.add(getJPanel15(), null);
                        jPanel14.add(getJPanel17(), null);
                        jPanel14.add(getJPanel19(), null);
                        jPanel14.add(getJPanel21(), null);
                        jPanel14.add(getJPanel23(), null);
                }
                return jPanel14;
        }

        /**
         * This method initializes jPanel15
         *
```

```
	 * @return javax.swing.JPanel
	 */
	private JPanel getJPanel15() {
		if (jPanel15 == null) {
			jLabel10 = new JLabel();
			jLabel10.setText("(kilobytes)");
			jPanel15 = new JPanel(new FlowLayout(FlowLayout.LEFT));
			jPanel15.add(getJPanel16(), null);
			jPanel15.add(getJTextField3(), null);
			jPanel15.add(jLabel10, null);
		}
		return jPanel15;
	}

	/**
	 * This method initializes jPanel16
	 *
	 * @return javax.swing.JPanel
	 */
	private JPanel getJPanel16() {
		if (jPanel16 == null) {
			jLabel9 = new JLabel();
			jLabel9.setText("size");
			jPanel16 = new JPanel();
			jPanel16.setPreferredSize(labelDim);
			jPanel16.add(jLabel9, null);
		}
		return jPanel16;
	}

	/**
	 * This method initializes jTextField3
	 *
	 * @return javax.swing.JTextField
	 */
	private JTextField getJTextField3() {
		if (jTextField3 == null) {
			jTextField3 = new JTextField();
			jTextField3.setColumns(10);
			jTextField3.setText("16");
		}
		return jTextField3;
	}

	/**
	 * This method initializes jPanel17
	 *
	 * @return javax.swing.JPanel
	 */
	private JPanel getJPanel17() {
		if (jPanel17 == null) {
			jLabel12 = new JLabel();
			jLabel12.setText("(number of blocks per set)");
			jPanel17 = new JPanel(new FlowLayout(FlowLayout.LEFT));
			jPanel17.add(getJPanel18(), null);
			jPanel17.add(getJTextField4(), null);
			jPanel17.add(jLabel12, null);
		}
		return jPanel17;
	}

	/**
	 * This method initializes jPanel18
	 *
	 * @return javax.swing.JPanel
	 */
	private JPanel getJPanel18() {
		if (jPanel18 == null) {
			jLabel11 = new JLabel();
			jLabel11.setText("associativity");
			jPanel18 = new JPanel();
			jPanel18.setPreferredSize(labelDim);
			jPanel18.add(jLabel11, null);
		}
		return jPanel18;
	}
```

```
/**
 * This method initializes jTextField4
 *
 * @return javax.swing.JTextField
 */
private JTextField getJTextField4() {
	if (jTextField4 == null) {
		jTextField4 = new JTextField();
		jTextField4.setColumns(10);
		jTextField4.setText("2");
	}
	return jTextField4;
}

/**
 * This method initializes jPanel19
 *
 * @return javax.swing.JPanel
 */
private JPanel getJPanel19() {
	if (jPanel19 == null) {
		jLabel14 = new JLabel();
		jLabel14.setText("(bytes)");
		jPanel19 = new JPanel(new FlowLayout(FlowLayout.LEFT));
		jPanel19.add(getJPanel20(), null);
		jPanel19.add(getJTextField5(), null);
		jPanel19.add(jLabel14, null);
	}
	return jPanel19;
}

/**
 * This method initializes jPanel20
 *
 * @return javax.swing.JPanel
 */
private JPanel getJPanel20() {
	if (jPanel20 == null) {
		jLabel13 = new JLabel();
		jLabel13.setText("block size");
		jPanel20 = new JPanel();
		jPanel20.setPreferredSize(labelDim);
		jPanel20.add(jLabel13, null);
	}
	return jPanel20;
}

/**
 * This method initializes jTextField5
 *
 * @return javax.swing.JTextField
 */
private JTextField getJTextField5() {
	if (jTextField5 == null) {
		jTextField5 = new JTextField();
		jTextField5.setColumns(10);
		jTextField5.setText("64");
	}
	return jTextField5;
}

/**
 * This method initializes jPanel21
 *
 * @return javax.swing.JPanel
 */
private JPanel getJPanel21() {
	if (jPanel21 == null) {
		jPanel21 = new JPanel(new FlowLayout(FlowLayout.LEFT));
		jPanel21.add(getJPanel22(), null);
		jPanel21.add(getJComboBoxReplacePolicyICache(), null);
	}
	return jPanel21;
}

/**
 * This method initializes jPanel22
```

```
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJPanel22() {
                if (jPanel22 == null) {
                        jLabel15 = new JLabel();
                        jLabel15.setText("replace policy");
                        jPanel22 = new JPanel();
                        jPanel22.setPreferredSize(labelDim);
                        jPanel22.add(jLabel15, null);
                }
                return jPanel22;
        }

        /**
         * This method initializes jComboBox
         *
         * @return javax.swing.JComboBox
         */
        private JComboBox getJComboBoxReplacePolicyICache() {
                if (jComboBoxReplacePolicyICache == null) {
                        jComboBoxReplacePolicyICache = new JComboBox();
                        jComboBoxReplacePolicyICache.addItem(new
MyItem("LRU",Cache.S_LRU));
                        jComboBoxReplacePolicyICache.addItem(new MyItem("Random",
Cache.S_RANDOM));
                        jComboBoxReplacePolicyICache.addItem(new MyItem("Fifo",
Cache.S_FIFO));
                }
                return jComboBoxReplacePolicyICache;
        }

        /**
         * This method initializes jPanel23
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJPanel23() {
                if (jPanel23 == null) {
                        jPanel23 = new JPanel(new FlowLayout(FlowLayout.LEFT));
                        jPanel23.add(getJPanel24(), null);
                        jPanel23.add(getJComboBoxWritePolicyICache(), null);

                }
                return jPanel23;
        }

        /**
         * This method initializes jPanel24
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJPanel24() {
                if (jPanel24 == null) {
                        jLabel16 = new JLabel();
                        jLabel16.setText("write policy");
                        jPanel24 = new JPanel();
                        jPanel24.setPreferredSize(labelDim);
                        jPanel24.add(jLabel16, null);
                }
                return jPanel24;
        }

        /**
         * This method initializes jComboBox1
         *
         * @return javax.swing.JComboBox
         */
        private JComboBox getJComboBoxWritePolicyICache() {
                if (jComboBoxWritePolicyICache == null) {
                        jComboBoxWritePolicyICache = new JComboBox();
                        jComboBoxWritePolicyICache.addItem(new MyItem("none (instruction
cache)",-1));
                        jComboBoxWritePolicyICache.setEnabled(false);

                }
                return jComboBoxWritePolicyICache;
```

```
	}

	private class MyItem{
		public int value;
		public String desc;
		public MyItem(String descricao, int value){
			desc = descricao;
			this.value = value;
		}

		public String toString(){
			return desc;
		}

	}

	/**
	 * This method initializes jCheckBoxNextLevelReg
	 *
	 * @return javax.swing.JCheckBox
	 */
	private JCheckBox getJCheckBoxNextLevelReg() {
		if (jCheckBoxNextLevelReg == null) {
			jCheckBoxNextLevelReg = new JCheckBox();
			jCheckBoxNextLevelReg.setText("Next Level Access Log");
			jCheckBoxNextLevelReg.setSelected(true);
			jCheckBoxNextLevelReg.addActionListener(new
java.awt.event.ActionListener() {
				public void actionPerformed(java.awt.event.ActionEvent e) {
					//System.out.println("actionPerformed()"); // TODO
Auto-generated Event stub actionPerformed()
					if(jCheckBoxNextLevelReg.isSelected())
						System.out.println("Selected!");
					else
						System.out.println("Not Selected!");
				}
			});
		}
		return jCheckBoxNextLevelReg;
	}

}  //  @jve:decl-index=0:visual-constraint="10,10"
```

6.2.15 UnifiedCache

Representa uma cache unificada, apesar de se puder usar directamente um objecto do tipo Cache optou-se por esta solução para normalizar o programa.

```
package cacheSim;

/**
 * Representa uma cache unificada (cache de dado + cache de instruções, tudo na mesma
cache)
 * Esta classe é executável para poder correr simulações pesadas.
 * @author Manuel
 *
 */
public class UnifiedCache {

	Cache uCache;

	private MemAccess lastMemAccess;
	public UnifiedCache(Cache uCache)
	throws Exception
	{
		if(uCache.isInstrCache())
			throw new Exception("unified cache can't be an instruction
cache!");
		this.uCache = uCache;


	}

	public MemAccess getLastMemAccess(){
		return this.lastMemAccess;
	}

	public long getNumAccess(){
		return uCache.getNumAccess();
	}
	public long getNumHits(){
		return uCache.getNumHits();
	}
	public long getNumMisses(){
		return uCache.getNumMisses();
	}
	public long getNumWriteThroughs(){
		return uCache.getNumWriteThroughs();
	}
	public long getNumWriteBacks(){
		return uCache.getNumWriteBacks();
	}
	public long getNumIAccess(){
		return uCache.getNumIAccess();
	}
	public long getNumDRAccess(){
		return uCache.getNumDRAccess();
	}
	public long getNumDWAccess(){
		return uCache.getNumDWAccess();
	}
	public long getNumNextLevelAccess(){
		return uCache.getNumNextLevelAccess();
	}


	public Cache getUCache(){
		return uCache;
	}

	public int getLastAccessBlockIndex(){

		return uCache.getLastAccessBlockIndex();

	}
```

```
	public int getSetIndex(MemAccess ma){
		return uCache.getSetIndex(ma);
	}

	//retorna o index do bloco ao qual pertence o primeiro set
	public int getSetBlockIndex(MemAccess ma){
		return (uCache.getSetIndex(ma) * uCache.getAssociatividade());

	}

	public void doMemAccess(MemAccess ma)
	throws Cache.CacheFormatException
	{
		this.lastMemAccess = ma.getCopy();
		uCache.doMemAccess(ma);
	}

	public static void main(String args[]){
		try{

			if(args.length != 6){
				System.out.println("Wrong usage! try:");
				System.out.println("java cacheSim.SplitCache"+
				"\n\t<cache size (KB)> \n\t<block size (bytes)>
\n\t<associativity> \n\t<LRU | FIFO | RANDOM> \n\t<WR_BACK | WR_THROUGH>\n\t<trace
file>");
				System.exit(0);
			}

			int size = Integer.parseInt(args[0]);
			int block = Integer.parseInt(args[1]);
			int assoc = Integer.parseInt(args[2]);
			int sub = -1;
			int wr = -1;
			if(args[3].compareTo("LRU") == 0)
				sub = Cache.S_LRU;
			if(args[3].compareTo("FIFO") == 0)
				sub = Cache.S_FIFO;
			if(args[3].compareTo("RANDOM") == 0)
				sub = Cache.S_RANDOM;

			if(args[4].compareTo("WR_BACK") == 0)
				wr = Cache.WR_BACK_ALLOCATE;
			if(args[4].compareTo("WR_THROUGH") == 0)
				wr = Cache.WR_THROUGH_NO_ALLOCATE;

			Cache cache = new Cache(size, block, assoc, sub, wr, false);
			UnifiedCache uCache = new UnifiedCache(cache);


			MemAccessFile maf = new MemAccessFile(args[5]);
			int i;
			for(i=0; maf.hasMore(); i++){
				MemAccess ma = maf.getNext();
				//System.out.println(ma.toString());
				uCache.doMemAccess(ma);
			}
			maf.close();
			//System.out.println("Read "+i+" instructions!");

			System.out.println("----------------------------\nUnified Cache
Statistics\n----------------------------");
			cache.printStatistcs();

		}catch(Exception e){
			e.printStackTrace();
		}

	}

}
```

6.2.16 UnifiedCacheDialog

Janela de dialogo onde o utilizador escolhe as caracteristicas da cache unificada e arranca a simulação.

```
package cacheSim;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.GridLayout;

import javax.swing.BorderFactory;
import javax.swing.JFileChooser;
import javax.swing.JPanel;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JButton;
import javax.swing.JComboBox;


import javax.swing.JCheckBox;

/**
 * Janela de dialogo com o utilizador para inserir os dados da cache unificada
 * e o ficheiro de acessos.
 * @author Manuel
 *
 */
public class UnifiedCacheDialog extends JDialog {
	public static final long serialVersionUID=0;

	private JPanel jContentPane = null;

	private JPanel jPanelCenter = null;

	private JPanel jPanel = null;

	private JLabel jLabel = null;

	private JTextField jTextFieldTraceFile = null;

	private JButton jButtonOpenFile = null;

	private JPanel jPanel1 = null;

	private JPanel jPanel2 = null;

	private JTextField jTextFieldBlockSize = null;

	private JPanel jPanel3 = null;

	private JTextField jTextFieldSize = null;

	private JPanel jPanel4 = null;

	private JTextField jTextFieldAssociatividade = null;

	private JLabel jLabel4 = null;

	private JLabel jLabel5 = null;

	private JLabel jLabel6 = null;

	private JPanel jPanel5 = null;

	private JComboBox jComboBoxReplacePolicy = null;

	private JPanel jPanel6 = null;
```

```
	private JLabel jLabel1 = null;

	private JPanel jPanel7 = null;

	private JLabel jLabel3 = null;

	private final Dimension labelDim = new Dimension(100,25);

	private JPanel jPanel8 = null;

	private JLabel jLabel2 = null;

	private JPanel jPanel9 = null;

	private JLabel jLabel7 = null;

	private JPanel jPanel10 = null;

	private JPanel jPanel11 = null;

	private JLabel jLabel8 = null;

	private JComboBox jComboBoxWritePolicy = null;

	private JPanel jPanel12 = null;

	private JButton jButtonCancel = null;

	private JButton jButtonOK = null;

	private RunProgram mainFrame;

	private JCheckBox jCheckBoxRegistry = null;
	/**
	 * This is the default constructor
	 */
	public UnifiedCacheDialog(RunProgram frame) {
		super(frame, true);
		initialize();
		this.setTitle("New Simulation...");
		this.setSize(450,350);
		this.setLocationRelativeTo(null);
		this.setResizable(false);
		mainFrame = frame;
	}

	/**
	 * This method initializes this
	 *
	 * @return void
	 */
	private void initialize() {

		//this.setPreferredSize(new Dimension(400,400));
		this.setContentPane(getJContentPane());
	}

	/**
	 * This method initializes jContentPane
	 *
	 * @return javax.swing.JPanel
	 */
	private JPanel getJContentPane() {
		if (jContentPane == null) {
			jContentPane = new JPanel();
			jContentPane.setLayout(new BorderLayout());
			jContentPane.add(getJPanelCenter(), java.awt.BorderLayout.CENTER);

	jContentPane.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));

		}
		return jContentPane;
	}

	/**
	 * This method initializes jPanelCenter
	 *
```

```
	 * @return javax.swing.JPanel
	 */
	private JPanel getJPanelCenter() {
		if (jPanelCenter == null) {
			jPanelCenter = new JPanel(new BorderLayout());
			jPanelCenter.add(getJPanel(), BorderLayout.NORTH);
			jPanelCenter.add(getJPanel1(), BorderLayout.CENTER);
			jPanelCenter.add(getJPanel12(), java.awt.BorderLayout.SOUTH);

		}
		return jPanelCenter;
	}

	/**
	 * This method initializes jPanel
	 *
	 * @return javax.swing.JPanel
	 */
	private JPanel getJPanel() {
		if (jPanel == null) {
			jLabel = new JLabel();
			jLabel.setText("Trace File");
			jPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
			jPanel.add(jLabel, null);
			jPanel.add(getJTextFieldTraceFile(), null);
			jPanel.add(getJButtonOpenFile(), null);


		}
		return jPanel;
	}

	/**
	 * This method initializes jTextFieldTraceFile
	 *
	 * @return javax.swing.JTextField
	 */
	private JTextField getJTextFieldTraceFile() {
		if (jTextFieldTraceFile == null) {
			jTextFieldTraceFile = new JTextField();
			jTextFieldTraceFile.setColumns(20);
		}
		return jTextFieldTraceFile;
	}

	/**
	 * This method initializes jButtonOpenFile
	 *
	 * @return javax.swing.JButton
	 */
	private JButton getJButtonOpenFile() {
		if (jButtonOpenFile == null) {
			jButtonOpenFile = new JButton();
			jButtonOpenFile.setText("...");

			jButtonOpenFile.addActionListener(new
java.awt.event.ActionListener() {
				public void actionPerformed(java.awt.event.ActionEvent e) {
					System.out.println("actionPerformed()"); // TODO
Auto-generated Event stub actionPerformed()

					JFileChooser chooser = new
JFileChooser(System.getProperty("user.dir"));
					// Note: source for ExampleFileFilter can be found
in FileChooserDemo,
					// under the demo/jfc directory in the Java 2 SDK,
Standard Edition.
					int returnVal =
chooser.showOpenDialog(UnifiedCacheDialog.this);
					if(returnVal == JFileChooser.APPROVE_OPTION) {
						System.out.println("You chose to open this
file: " +

	chooser.getSelectedFile().getName());

	System.out.println("File="+chooser.getSelectedFile().getAbsolutePath());
```

```
	UnifiedCacheDialog.this.jTextFieldTraceFile.setText(chooser.getSelectedFile().get
AbsolutePath());
                                }
                        }
                });
        }
        return jButtonOpenFile;
    }

    /**
     * This method initializes jPanel1
     *
     * @return javax.swing.JPanel
     */
    private JPanel getJPanel1() {
        if (jPanel1 == null) {
                jPanel1 = new JPanel(new GridLayout(5,1,5,5));

    jPanel1.setBorder(BorderFactory.createTitledBorder(BorderFactory.createLineBorder
(Color.BLACK,1),"Unified Cache"));
                jPanel1.add(getJPanel3(), null);
                jPanel1.add(getJPanel4(), null);
                jPanel1.add(getJPanel2(), null);
                jPanel1.add(getJPanel5(), null);
                jPanel1.add(getJPanel10(), null);


        }
        return jPanel1;
    }

    /**
     * This method initializes jPanel2
     *
     * @return javax.swing.JPanel
     */
    private JPanel getJPanel2() {
        if (jPanel2 == null) {
                jLabel6 = new JLabel();
                jLabel6.setText("(bytes)");
                jPanel2 = new JPanel(new FlowLayout(FlowLayout.LEFT));
                jPanel2.add(getJPanel8(), null);
                jPanel2.add(getJTextFieldBlockSize(), null);
                jPanel2.add(jLabel6, null);
        }
        return jPanel2;
    }

    /**
     * This method initializes jTextField
     *
     * @return javax.swing.JTextField
     */
    private JTextField getJTextFieldBlockSize() {
        if (jTextFieldBlockSize == null) {
                jTextFieldBlockSize = new JTextField();
                jTextFieldBlockSize.setColumns(10);
                jTextFieldBlockSize.setText("64");
        }
        return jTextFieldBlockSize;
    }

    /**
     * This method initializes jPanel3
     *
     * @return javax.swing.JPanel
     */
    private JPanel getJPanel3() {
        if (jPanel3 == null) {
                jLabel5 = new JLabel();
                jLabel5.setText("(kilobytes)");

                jPanel3 = new JPanel(new FlowLayout(FlowLayout.LEFT));
                jPanel3.add(getJPanel6(), null);
                jPanel3.add(getJTextFieldSize(), null);
                jPanel3.add(jLabel5, null);
```

```
		}
		return jPanel3;
	}

	/**
	 * This method initializes jTextField1
	 *
	 * @return javax.swing.JTextField
	 */
	private JTextField getJTextFieldSize() {
		if (jTextFieldSize == null) {
			jTextFieldSize = new JTextField();
			jTextFieldSize.setColumns(10);
			jTextFieldSize.setText("32");
		}
		return jTextFieldSize;
	}

	/**
	 * This method initializes jPanel4
	 *
	 * @return javax.swing.JPanel
	 */
	private JPanel getJPanel4() {
		if (jPanel4 == null) {
			jLabel4 = new JLabel();
			jLabel4.setText("(number of blocks per set)");
			jPanel4 = new JPanel(new FlowLayout(FlowLayout.LEFT));
			jPanel4.add(getJPanel7(), null);
			jPanel4.add(getJTextFieldAssociatividade(), null);
			jPanel4.add(jLabel4, null);
		}
		return jPanel4;
	}

	/**
	 * This method initializes jTextField2
	 *
	 * @return javax.swing.JTextField
	 */
	private JTextField getJTextFieldAssociatividade() {
		if (jTextFieldAssociatividade == null) {
			jTextFieldAssociatividade = new JTextField();
			jTextFieldAssociatividade.setColumns(10);
			jTextFieldAssociatividade.setText("2");
		}
		return jTextFieldAssociatividade;
	}

	/**
	 * This method initializes jPanel5
	 *
	 * @return javax.swing.JPanel
	 */
	private JPanel getJPanel5() {
		if (jPanel5 == null) {
			jPanel5 = new JPanel(new FlowLayout(FlowLayout.LEFT));
			jPanel5.add(getJPanel9(), null);
			jPanel5.add(getJComboBoxReplacePolicy(), null);
		}
		return jPanel5;
	}

	/**
	 * This method initializes jComboBoxReplacePolicy
	 *
	 * @return javax.swing.JComboBox
	 */
	private JComboBox getJComboBoxReplacePolicy() {
		if (jComboBoxReplacePolicy == null) {
			jComboBoxReplacePolicy = new JComboBox();
			jComboBoxReplacePolicy.addItem(new MyItem("LRU", Cache.S_LRU));
```

```
                        jComboBoxReplacePolicy.addItem(new MyItem("Random",
Cache.S_RANDOM));
                        jComboBoxReplacePolicy.addItem(new MyItem("Fifo", Cache.S_FIFO));
                }
                return jComboBoxReplacePolicy;
        }

        /**
         * This method initializes jPanel6
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJPanel6() {
                if (jPanel6 == null) {
                        jLabel1 = new JLabel();
                        jLabel1.setText("size");
                        jPanel6 = new JPanel();

                        jPanel6.setPreferredSize(labelDim);
                        jPanel6.add(jLabel1, null);
                }
                return jPanel6;
        }

        /**
         * This method initializes jPanel7
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJPanel7() {
                if (jPanel7 == null) {
                        jLabel3 = new JLabel();
                        jLabel3.setText("associativity");
                        jPanel7 = new JPanel();
                        jPanel7.add(jLabel3, null);
                        jPanel7.setPreferredSize(labelDim);
                }
                return jPanel7;
        }

        /**
         * This method initializes jPanel8
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJPanel8() {
                if (jPanel8 == null) {
                        jLabel2 = new JLabel();
                        jLabel2.setText("block size");
                        jPanel8 = new JPanel();
                        jPanel8.add(jLabel2, null);
                        jPanel8.setPreferredSize(labelDim);
                }
                return jPanel8;
        }

        /**
         * This method initializes jPanel9
         *
         * @return javax.swing.JPanel
         */
        private JPanel getJPanel9() {
                if (jPanel9 == null) {
                        jLabel7 = new JLabel();
                        jLabel7.setText("replace policy");
                        jPanel9 = new JPanel();
                        jPanel9.add(jLabel7, null);
                        jPanel9.setPreferredSize(labelDim);
                }
                return jPanel9;
        }

        /**
         * This method initializes jPanel10
         *
         * @return javax.swing.JPanel
         */
```

```
	private JPanel getJPanel10() {
		if (jPanel10 == null) {
			jPanel10 = new JPanel(new FlowLayout(FlowLayout.LEFT));
			jPanel10.add(getJPanel11(), null);
			jPanel10.add(getJComboBoxWritePolicy(), null);
		}
		return jPanel10;
	}

	/**
	 * This method initializes jPanel11
	 *
	 * @return javax.swing.JPanel
	 */
	private JPanel getJPanel11() {
		if (jPanel11 == null) {
			jLabel8 = new JLabel();
			jLabel8.setText("write policy");
			jPanel11 = new JPanel();
			jPanel11.add(jLabel8, null);
			jPanel11.setPreferredSize(labelDim);
		}
		return jPanel11;
	}

	/**
	 * This method initializes jComboBox
	 *
	 * @return javax.swing.JComboBox
	 */
	private JComboBox getJComboBoxWritePolicy() {
		if (jComboBoxWritePolicy == null) {
			jComboBoxWritePolicy = new JComboBox();
			jComboBoxWritePolicy.addItem(new MyItem("write-back + write-
allocate",Cache.WR_BACK_ALLOCATE));
			jComboBoxWritePolicy.addItem(new MyItem("write-through + write-no-
allocate",Cache.WR_THROUGH_NO_ALLOCATE));
		}
		return jComboBoxWritePolicy;
	}

	/**
	 * This method initializes jPanel12
	 *
	 * @return javax.swing.JPanel
	 */
	private JPanel getJPanel12() {
		if (jPanel12 == null) {
			jPanel12 = new JPanel(new FlowLayout(FlowLayout.RIGHT));
			jPanel12.add(getJCheckBoxRegistry(), null);
			jPanel12.add(getJButtonCancel(), null);
			jPanel12.add(getJButtonOK(), null);

		}
		return jPanel12;
	}

	/**
	 * This method initializes jButtonCancel
	 *
	 * @return javax.swing.JButton
	 */
	private JButton getJButtonCancel() {
		if (jButtonCancel == null) {
			jButtonCancel = new JButton();
			jButtonCancel.setText("Cancel");
			jButtonCancel.setPreferredSize(labelDim);
			jButtonCancel.addActionListener(new
java.awt.event.ActionListener() {
				public void actionPerformed(java.awt.event.ActionEvent e) {
					System.out.println("simulationCancel()"); // TODO
Auto-generated Event stub actionPerformed()
					UnifiedCacheDialog.this.setVisible(false);
				}
			});
		}
		return jButtonCancel;
```

```
	}

	/**
	 * This method initializes jButtonOK
	 *
	 * @return javax.swing.JButton
	 */
	private JButton getJButtonOK() {
		if (jButtonOK == null) {
			jButtonOK = new JButton();
			jButtonOK.setText("OK");
			jButtonOK.setPreferredSize(labelDim);
			jButtonOK.addActionListener(new java.awt.event.ActionListener() {
				public void actionPerformed(java.awt.event.ActionEvent e) {
					//System.out.println("simulationOK()"); // TODO
Auto-generated Event stub actionPerformed()
					//mainFrame.setReadyStatus(true);
					//UnifiedCacheDialog.this.setVisible(false);


					try{
						int u_kb =
Integer.parseInt(jTextFieldSize.getText());
						int u_sets =
Integer.parseInt(jTextFieldAssociatividade.getText());
						int u_block =
Integer.parseInt(jTextFieldBlockSize.getText());
						int u_sub =
((MyItem)jComboBoxReplacePolicy.getSelectedItem()).value;
						int u_wr =
((MyItem)jComboBoxWritePolicy.getSelectedItem()).value;


						Cache uCache = new Cache(u_kb, u_block,
u_sets, u_sub, u_wr, true);

						UnifiedCache unifiedCache = new
UnifiedCache(uCache);
						//SimParams simParams = new SimParams();

	if(jTextFieldTraceFile.getText().compareTo("") == 0){
							WarningDialog wd = new
WarningDialog(UnifiedCacheDialog.this, "Warning", "\nYou have to select a compatible
trace file!\n\nUse the '...' button to find it.");
							wd.setVisible(true);
						}else{

	mainFrame.setNextLevelRegistry(jCheckBoxRegistry.isSelected());
							mainFrame.setupSimThread(new
SimParams(unifiedCache,jTextFieldTraceFile.getText()));
							mainFrame.setReadyStatus(true);

	UnifiedCacheDialog.this.setVisible(false);
						}
					}catch(Exception exc){
						WarningDialog wd = new
WarningDialog(UnifiedCacheDialog.this, "Erro", exc.toString());
						wd.setVisible(true);
					}

				}
			});
		}
		return jButtonOK;
	}

	private class MyItem{
		public int value;
		public String desc;
		public MyItem(String descricao, int value){
			desc = descricao;
			this.value = value;
		}

		public String toString(){
			return desc;
		}
```

```
	}

	/**
	 * This method initializes jCheckBoxRegistry
	 *
	 * @return javax.swing.JCheckBox
	 */
	private JCheckBox getJCheckBoxRegistry() {
		if (jCheckBoxRegistry == null) {
			jCheckBoxRegistry = new JCheckBox();
			jCheckBoxRegistry.setText("Next Level Access Log");
			jCheckBoxRegistry.setSelected(true);

		}
		return jCheckBoxRegistry;
	}

}  //  @jve:decl-index=0:visual-constraint="10,10"
```

6.2.17 WarningDialog

Janela de dialogo simples, usada para lançar mensagens de alerta. Esta janela bloqueia o acesso à janela que a lançou.

```
package cacheSim;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Frame;


import javax.swing.BorderFactory;

import javax.swing.JPanel;
import javax.swing.JDialog;

import javax.swing.JButton;
import javax.swing.JTextPane;

/**
 * Janela de dialogo de uso geral...
 * Apenas serve de alerta para o utilizador.
 * @author Manuel
 *
 */
public class WarningDialog extends JDialog {

	public static final long serialVersionUID = 0;
	private JPanel jContentPane = null;
	private JPanel jPanel = null;
	private JButton jButton = null;
	private JTextPane jTextPaneMessage = null;

	/**
	 * This is the default constructor
	 */
	public WarningDialog(JDialog frame, String title, String message) {
		super(frame, true);
		initialize();
		this.setTitle(title);
		this.setSize(300,200);
		this.setLocationRelativeTo(null);
		this.setResizable(false);
		this.jTextPaneMessage.setText(message);
	}
	public WarningDialog(Frame frame, String title, String message) {
		super(frame, true);
		initialize();
		this.setTitle(title);
		this.setSize(300,200);
		this.setLocationRelativeTo(null);
		this.setResizable(false);
		this.jTextPaneMessage.setText(message);
	}

	/**
	 * This method initializes this
	 *
	 * @return void
	 */
	private void initialize() {
		this.setSize(300, 200);
		this.setContentPane(getJContentPane());
	}

	/**
	 * This method initializes jContentPane
	 *
	 * @return javax.swing.JPanel
	 */
```

```
	private JPanel getJContentPane() {
		if (jContentPane == null) {
			jContentPane = new JPanel();
			jContentPane.setLayout(new BorderLayout());
			jContentPane.add(getJPanel(), java.awt.BorderLayout.SOUTH);

	jContentPane.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));
			jContentPane.add(getJTextPaneMessage(),
java.awt.BorderLayout.CENTER);
		}
		return jContentPane;
	}

	/**
	 * This method initializes jPanel
	 *
	 * @return javax.swing.JPanel
	 */
	private JPanel getJPanel() {
		if (jPanel == null) {
			jPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));
			jPanel.add(getJButton(), null);
		}
		return jPanel;
	}

	/**
	 * This method initializes jButton
	 *
	 * @return javax.swing.JButton
	 */
	private JButton getJButton() {
		if (jButton == null) {
			jButton = new JButton();
			jButton.setText("OK");
			jButton.addActionListener(new java.awt.event.ActionListener() {
				public void actionPerformed(java.awt.event.ActionEvent e) {
					System.out.println("warning ok"); // TODO Auto-
generated Event stub actionPerformed()
					WarningDialog.this.setVisible(false);

				}
			});
		}
		return jButton;
	}
	/**
	 * This method initializes jTextPaneMessage
	 *
	 * @return javax.swing.JTextPane
	 */
	private JTextPane getJTextPaneMessage() {
		if (jTextPaneMessage == null) {
			jTextPaneMessage = new JTextPane();
			jTextPaneMessage.setEditable(false);
			jTextPaneMessage.setBackground(new Color(240,240,240));


		}
		return jTextPaneMessage;
	}

}
```