ULBRA
CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS
COMUNIDADE EVANGÉLICA LUTERANA "SÃO PAULO"
Credenciado pelo Decreto de 06/07/2000 - D.O.U. nº 130 de 07/07/2000

IVO SÓCRATES MORAES DE OLIVEIRA

GERÊNCIA DE APLICAÇÕES COM A API JMX: MANUAL DE UTILIZAÇÃO

Palmas-TO

2006

ULBRA
CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS
COMUNIDADE EVANGÉLICA LUTERANA "SÃO PAULO"
Credenciado pelo Decreto de 06/07/2000 - D.O.U. nº 130 de 07/07/2000

IVO SÓCRATES MORAES DE OLIVEIRA

GERÊNCIA DE APLICAÇÕES COM A API JMX: MANUAL DE UTILIZAÇÃO

"Trabalho apresentado como requisito parcial da disciplina Estágio Supervisionado do Curso de Sistemas de Informação, supervisionado pela Profª. MSc. Madianita Bogo."

Palmas
2006

Sumário

Lista de Figuras

1 Introdução

O propósito da tecnologia de informação é permitir o bom uso das aplicações para o processamento de dados. Atualmente, existem inúmeras aplicações sem nenhum suporte de gerenciamento e as poucas que possuem gerenciamento sofrem com as dificuldades impostas pela heterogeneidade de *software* e *hardware* existentes no mercado. Este problema estende-se aos meios de interligação, os quais são responsáveis pelas comunicações entre recursos e serviços.

As aplicações que não possuem gerenciamento, em sua maioria, são vulneráveis a falhas por falta de recursos computacionais necessários no ambiente em que são executadas. De forma análoga, temos problemas quanto ao desperdício de recursos, ou seja, em determinados ambientes existem aplicações que possuem grande quantidade de recursos, porém, não são utilizados pelas aplicações (caracterizando, assim, uma má distribuição dos recursos computacionais). As funcionalidades que envolvem o gerenciamento de redes e aplicação podem auxiliar na tomada de decisões, na redução de custos, proporcionando uma maior praticidade na gerência e um melhor desempenho e estabilidade do sistema.

A API JMX (*Java Management Extensions* – Extensões de Gerenciamento Java) é uma tecnologia que apresenta meios para suprir a necessidade de gerenciamento de aplicações e recursos de redes de computadores, em plataformas distintas, baseando-se na JVM (*Java Virtual Machine* - Máquina Virtual Java).

Neste documento será abordado o gerenciamento de aplicações e recursos utilizando a API JMX. Para isso, serão apresentados conceitos iniciais sobre essa API, bem como sobre sua arquitetura, sua instalação, que é baseada no *kit* J2SE, exemplo de uso e teste de funcionamento. Como exemplo, será usada uma aplicação de um Jogo de

Anagramas, que foi retirado do site do Netbeans.org e para o teste será usada uma aplicação de análise combinatória, teste este desenvolvido a partir do conhecimento adquirido através do exemplo do Jogo de Anagramas.

Dentre os conceitos básicos da API JMX, serão abordadas as informações iniciais, as funcionalidades e a sua arquitetura, que proporcionarão subsídios para um melhor entendimento do funcionamento interno da API.

A instalação do J2SE (*Java 2 Standart Edition*), *kit* com o compilador Java, e a IDE de desenvolvimento NetBeans serão abordados neste documento para prover o ambiente de utilização e teste da API JMX.

O exemplo do desenvolvimento do Jogo de Anagramas, que tem por objetivo exemplificar uma aplicação gerenciável pela API JMX, será realizado com a utilização da IDE NetBeans. Essa IDE possibilita a criação de estruturas de métodos e de classes de uma forma simples e rápida, pois fornece uma interface de desenvolvimento através de *wizards* (assistentes), mas, oferece a possibilidade de desenvolvimento, também, em modo texto.

2 JMX (*Java Management Extensions*)

O JMX é uma API padrão introduzida no J2SE 5.0 para gerenciamento e monitoramento de recursos tais como aplicações, dispositivos, serviços e a JMV (*Java Virtual Machine* – Máquina Virtual Java). A tecnologia JMX foi desenvolvida através do JCP (*Java Community Process* – Processo Comunitário Java) e dentre suas várias características destacam-se (SUN, 2006, *online*)[1]:

- O gerenciamento de aplicações Java com baixos investimentos;
- Uma arquitetura de gerenciamento progressivo;
- Integração a soluções existentes de gerenciamento;
- Desenvolvimento das tecnologias padrão existentes de Java;
- Desenvolvimento de novos conceitos relacionados a gerência;
- Definição em módulos da gerência.

Por padrão, a API JMX inclui acesso remoto, permitindo que através de outros computadores contendo o *kit* J2SE instalado, seja possível gerenciar uma aplicação já instrumentada, e, também, um programa de gerenciamento remoto, o JConsole, que permite gerenciar aplicações em tempo de execução (SUN, 2006, *online*)[2].

Quanto à monitoração com o JMX, é possível descobrir em tempo de execução que serviços estão ativos e quais os recursos tais serviços estão consumindo. Já sobre a configuração com o JMX, é possível verificar e modificar valores de parâmetros de configuração, solicitar a inicialização, paralisação ou reinicialização de serviços. Existem várias aplicações que são gerenciáveis pelo JMX como, por exemplo, o JBoss (Servidor de Aplicação) e o Apache Tomcat (Servidor de Páginas *Web*).

2.1 Arquitetura do JMX

A arquitetura da API JMX, que é apresentada na figura 1, é dividida em três níveis: Nível Instrumentação, Nível Agente e Nível Gerente.

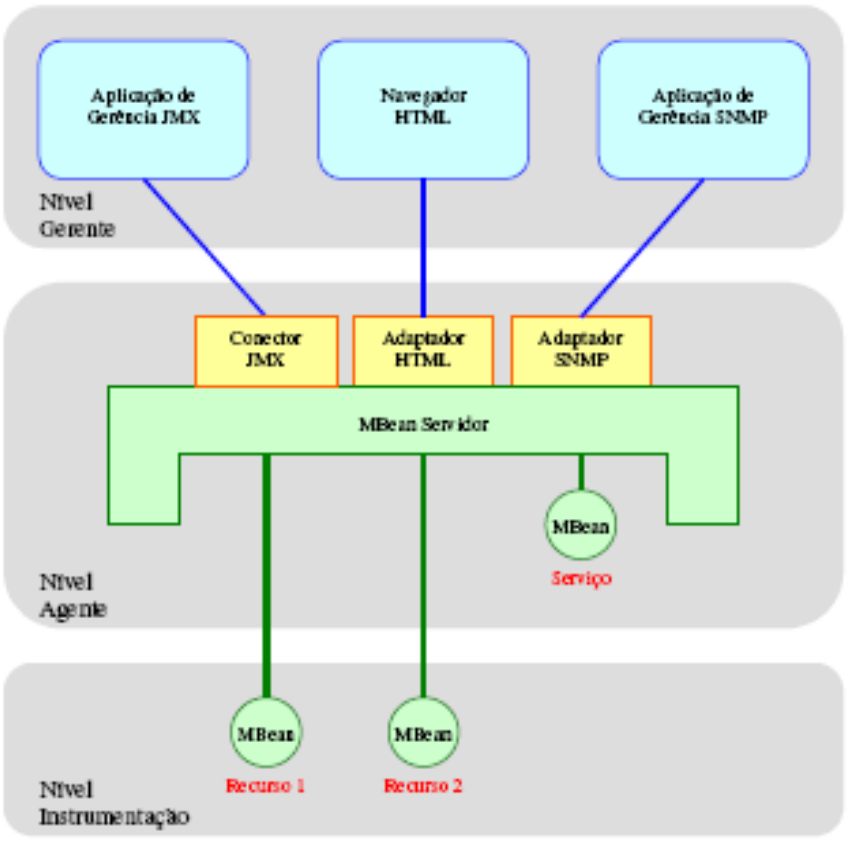

Figura 1: Arquitetura do JMX (FRANCISCO, 2001, p. 3)

O **Nível de Instrumentação** é responsável pela implementação de MBeans (*Management Beans* – Beans de Gerenciamento), que são objetos Java que representam os recursos a serem gerenciados tais como aplicações, dispositivos ou serviços. Além disso, os Beans de Gerenciamento expõem interfaces de gerenciamento compostas de atributos e operações, que podem fazer uso dos recursos de notificações para informar ou atualizar dados referentes aos recursos instrumentados.

Na especificação do JMX é definido o modelo de notificações genérico baseado no modelo de eventos Java, que provê objetos de notificações e a *interface* do transmissor e receptor das notificações enviadas e recebidas (SUN, 2006, *online*)[1]. As notificações poderão ser enviadas pelos MBeans ou pelo servidor de MBeans, que é especificado no Nível Agente do JMX.

O **Nível Agente** tem como componente principal o Servidor MBeans, o qual é responsável por controlar os recursos diretamente e torná-los acessíveis por agentes de gerenciamento remoto. Num agente pode-se ter um ou mais Servidor MBeans, porém, comumente, utiliza-se apenas um por JVM.

O **Nível Gerente** é responsável pela comunicação entre os Agentes e o Gerente. Os adaptadores e conectores de um determinado protocolo padrão fornecidos para a conexão tornam um agente JMX acessível pelas aplicações de gerência remota fora da máquina virtual Java do Agente. As aplicações de gerenciamento possuem a função de configuração e monitoração dos recursos e dispositivos instrumentados (SUN, 2004, *online*).

Dessa forma, ao implementar uma aplicação gerenciável torna-se necessária a implementação de MBeans, Agentes JMX e um Gerente JMX. Podem-se encontrar facilmente Gerentes JMX já implementados tais como o JConsole, MC4J, Spring WLS JMX Console e WLST.

Na seção 3, será implementado um exemplo que, além de mostrar como utilizar o NetBeans, exemplificará os passos necessários para o desenvolvimento de aplicações gerenciáveis.

Para desenvolvimento de aplicações com a utilização do JMX, é necessária apenas a instalação do J2SE que é responsável pela compilação das classes Java. Mas, neste documento será abordado o uso do J2SE juntamente a IDE NetBeans que proporcionará um ambiente de desenvolvimento organizado e prático. A preparação do ambiente é apresentada na seção 2.2.

2.2 Ambiente de Desenvolvimento JMX

Para a preparação do ambiente de desenvolvimento de aplicações gerenciáveis com a utilização da tecnologia JMX, que permitirá a monitoração e o controle de aplicações e dispositivos, são necessárias:

- Instalação do J2SE;
- Instalação do NetBeans;
- Configuração do NetBeans.

Essas etapas, referentes à preparação do ambiente de desenvolvimento de aplicações de gerenciamento remoto com a utilização da tecnologia JMX, serão minuciosamente detalhadas nas subseções a seguir.

A escolha da configuração do ambiente utilizando o Sistema Operacional Windows é dada pelo fato de que essa plataforma apresenta maior número de fontes de pesquisa e exemplos para testes. Os *softwares* instalados oferecem suporte às plataformas Linux, Unix e Solaris.

2.2.1 Instalação do J2SE

O J2SE (*Java 2 Standart Edition*) é um *kit* de ferramentas desenvolvido pela *Sun Microsystems* para desenvolvimento de *softwares* na linguagem Java, que possui como características portabilidade, segurança, estabilidade, eficiência, suporte a sistemas distribuídos e implementação de *multithreaded* (INDRUSIAK, 1996, p. 3-5). O J2SE é inteiramente grátis e possui sua licença baseada na *Java Internal Use License.*

A API JMX está inserida no pacote do J2SE, assim como diversas outras API´s, como pode ser visto na figura 2, que apresenta a plataforma *Java Standart Edition.*

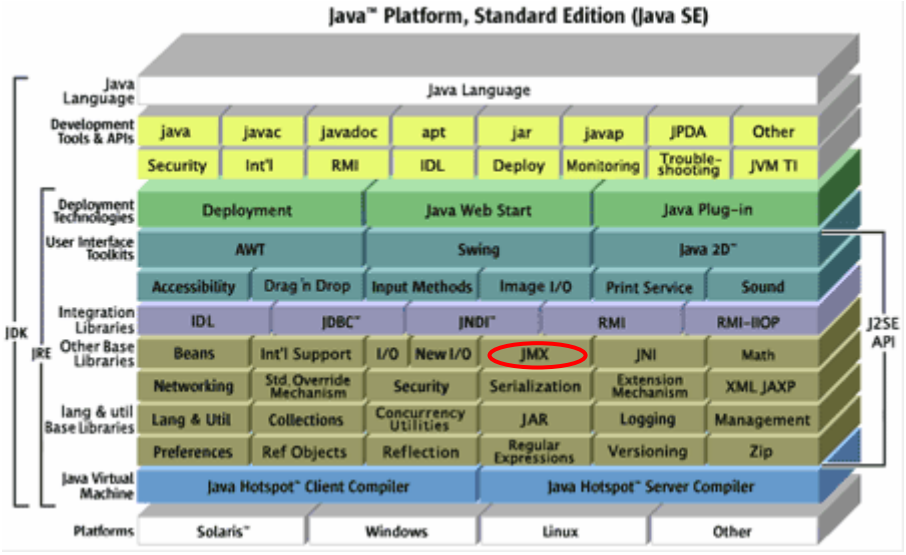

Figura 2: Plataforma *Java Standart Edition* (SUN, 2006, *online*)[3]

A figura 2 apresenta a plataforma *Java Standart Edition*, que será instalada para a implementação de aplicações gerenciáveis. Nela é possível visualizar o nível em que se

encontra a biblioteca da API JMX. A Plataforma Java S*tandart Edition* possui internamente dois *kits*: o primeiro, JDK (J2SE *Development Kit*), fornece a linguagem Java e ferramentas de desenvolvimento; o segundo, JRE (*J2SE Runtime Environment)*, fornece um conjunto de bibliotecas, componentes adicionais e a Máquina Virtual Java. Todos os recursos gerenciados pela API JMX necessitam do *kit* JRE instalado no computador para funcionar normalmente.

A versão *J2SE Development Kit 5.0 Update 8* é a mais recente até o momento de elaboração deste documento. Para instalar o J2SE, primeiramente, é necessário acessar o endereço www.sun.com.br. Em seguida deve-se selecionar a opção "*Downloads*" e, finalmente, transferir uma cópia do programa pela opção "*J2SE(TM) Development Kit 5.0 Update 8*".

Após a transferência do *J2SE(TM) Development Kit 5.0 Update 8*, é necessária a execução do instalador transferido. Após a execução do instalador, aparecerá o termo de licença de uso do programa J2SE, que apresenta permissões e restrições de uso da aplicação, conforme mostra a figura 3. Para prosseguir a instalação, é necessário concordar com o termo de licença apresentado, selecionando a opção "*I accept the terms in the license agreement*" e a opção "*Next*".

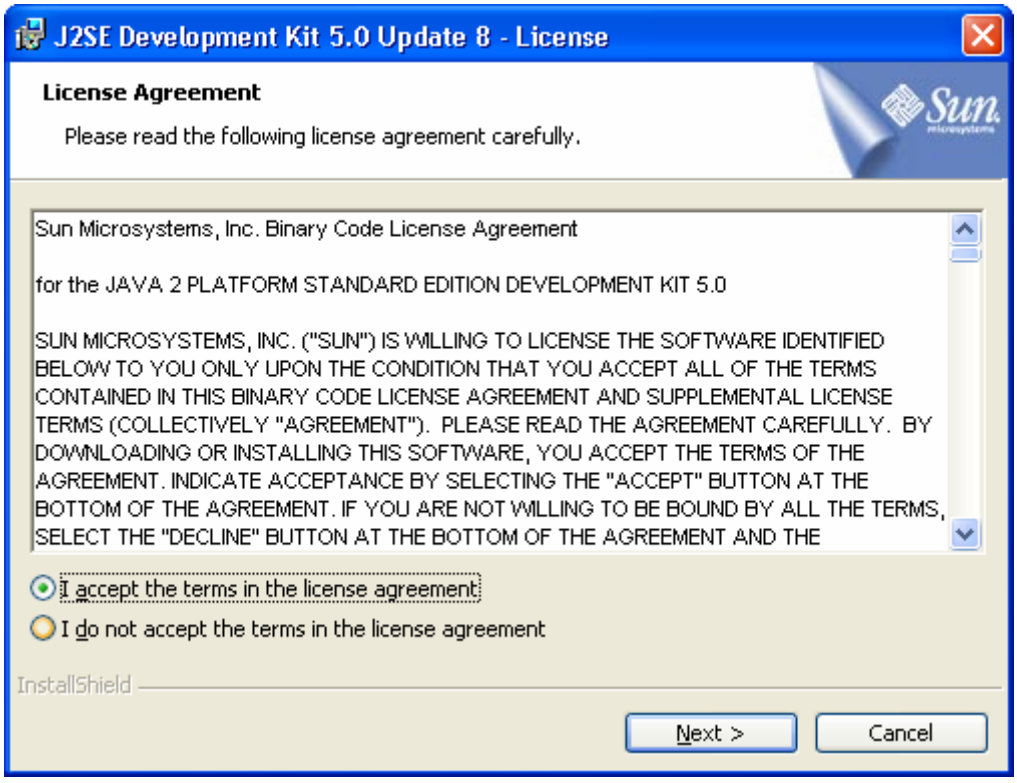

Figura 3: Tela com a licença de uso do J2SE

O próximo passo da instalação do J2SE é determinar que módulos do JDK devam ser instalados. Por padrão, todos os módulos estarão selecionados para a instalação, como apresenta a figura 4. Não haverá prejuízo na utilização da API JMX caso se retire os seguintes módulos: *Demos* e *Source Code*. No entanto, convêm mantê-los, pois poderão servir como auxílio da linguagem Java, provendo recursos para pesquisas que facilitam o entendimento de uso da linguagem. Logo, para instalar todos os módulos e prosseguir com a instalação, é necessário apenas selecionar o botão "*Next*".

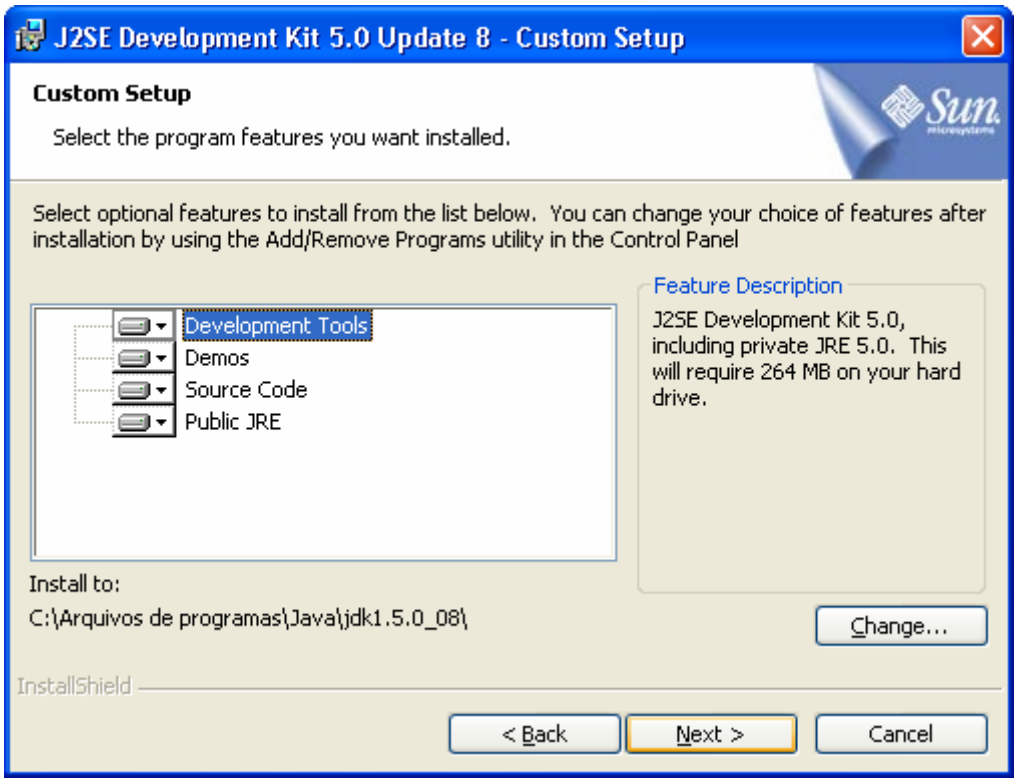

Figura 4: Tela de customização da instalação do J2SE JDK

O próximo passo da instalação é determinar que módulos do JRE devam ser instalados. Por padrão, todos os módulos estarão selecionados para a instalação, assim como apresenta a figura 5. A fim de que não haja problemas de suporte a idiomas, mídias e/ou fontes, torna-se necessário manter a instalação padrão para a qual basta selecionar o botão "*Next*".

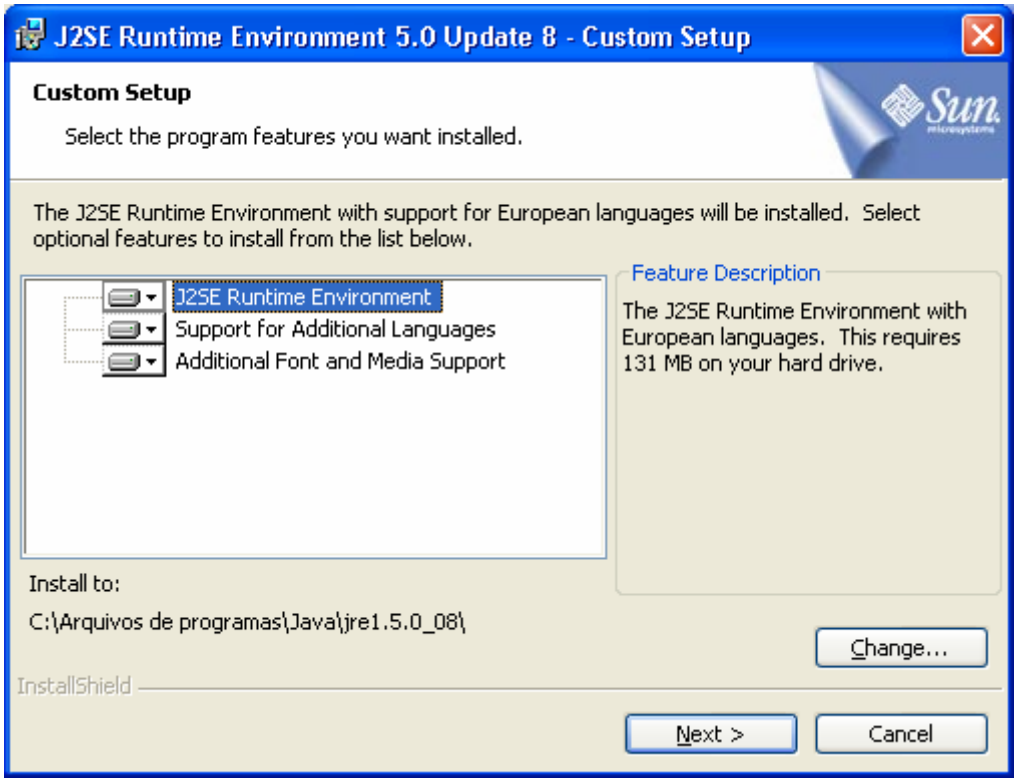

Figura 5: Tela de customização da instalação do J2SE JRE

Antes de terminar a instalação, é obrigatória a seleção de pelo menos um navegador compatível com o *Plug-in* Java. Isso é necessário, pois o Java utilizará o navegador selecionado nessa etapa para executar alguns testes. Logo após, será informado que a instalação foi concluída com sucesso. Deve-se selecionar o botão "*Finish*" para finalizar o processo de instalação.

2.2.2 Instalação do NetBeans

A IDE NetBeans é um ambiente de desenvolvimento livre e sem restrições de uso, completamente escrita em Java, que permite escrever, compilar, verificar erros e instalar programas em diversas linguagens de programação (NETBEANS, 2006, *online*)[1]. A última versão homologada até o período de elaboração deste documento é a 5.0.

Para instalar a IDE NetBeans 5.0, é necessário acessar o endereço http://www.netbeans.org/, selecionar a opção "*Downloads*" e, logo após, a opção "NetBeans IDE, *Mobility Pack and Profiler 5.0 downloads*". Em seguida, é preciso

informar a plataforma que será instalada e selecionar a opção “NetBeans IDE 5.0 *Installer*” para realizar a transferência de uma cópia do programa.

Em seguida, é necessária a execução do instalador transferido, após a qual aparecerá uma tela de boas vindas ao assistente de instalação do NetBeans. Para iniciar a instalação de fato deve-se selecionar o botão “*Next*”.

Em seguida, aparecerá o termo de licença de uso do NetBeans IDE 5.0, sendo que para continuar a instalação é necessário concordar com os termos da licença de uso apresentados. Para isso, deve-se selecionar a opção “*I accept the terms in the license agreement*” e o botão “*Next*”. A figura 6 apresenta a tela com a licença de uso do NetBeans IDE 5.0 com a opção “*I accept the terms in the license agreement*” selecionada.

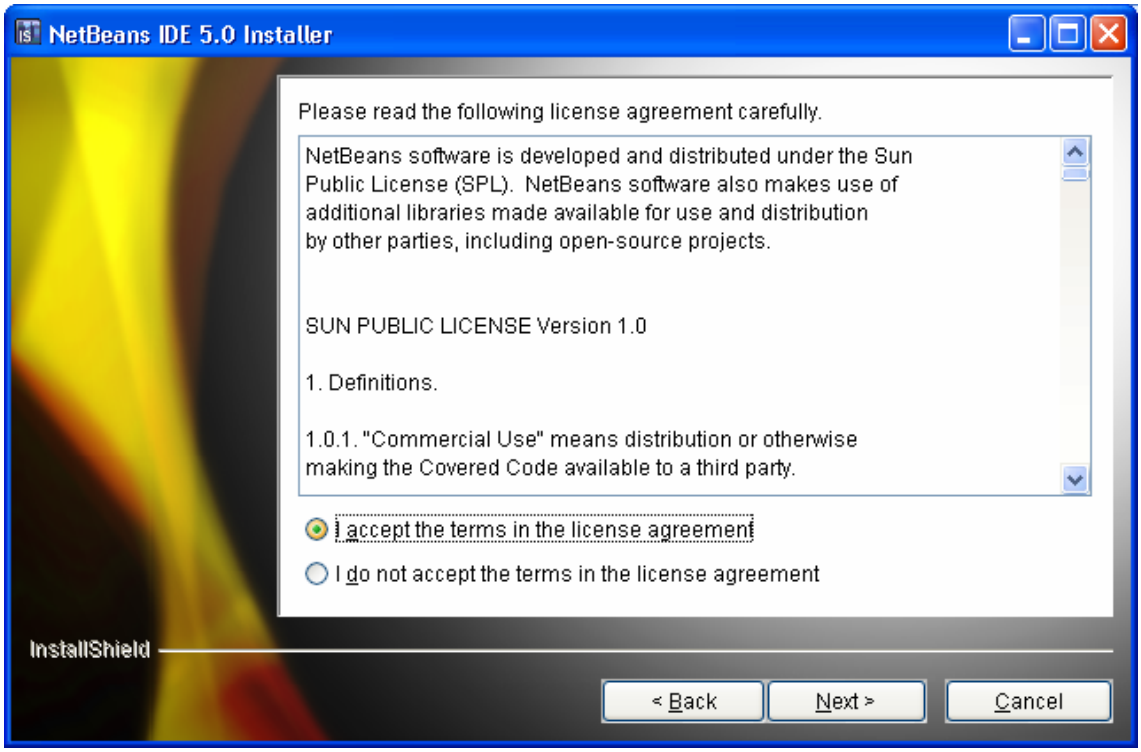

Figura 6: Tela com a licença de uso do NetBeans

Logo após, é necessário indicar o local de instalação do NetBeans no computador. Por padrão, ele vem com [DiretorioRaiz]:\[DiretorioDeProgramas]\netbeans-5.0, semelhante ao caminho apresentado na figura 7. Para confirmar a opção do diretório e instalar o NetBeans no local indicado, deve-se selecionar o botão “*Next*”.

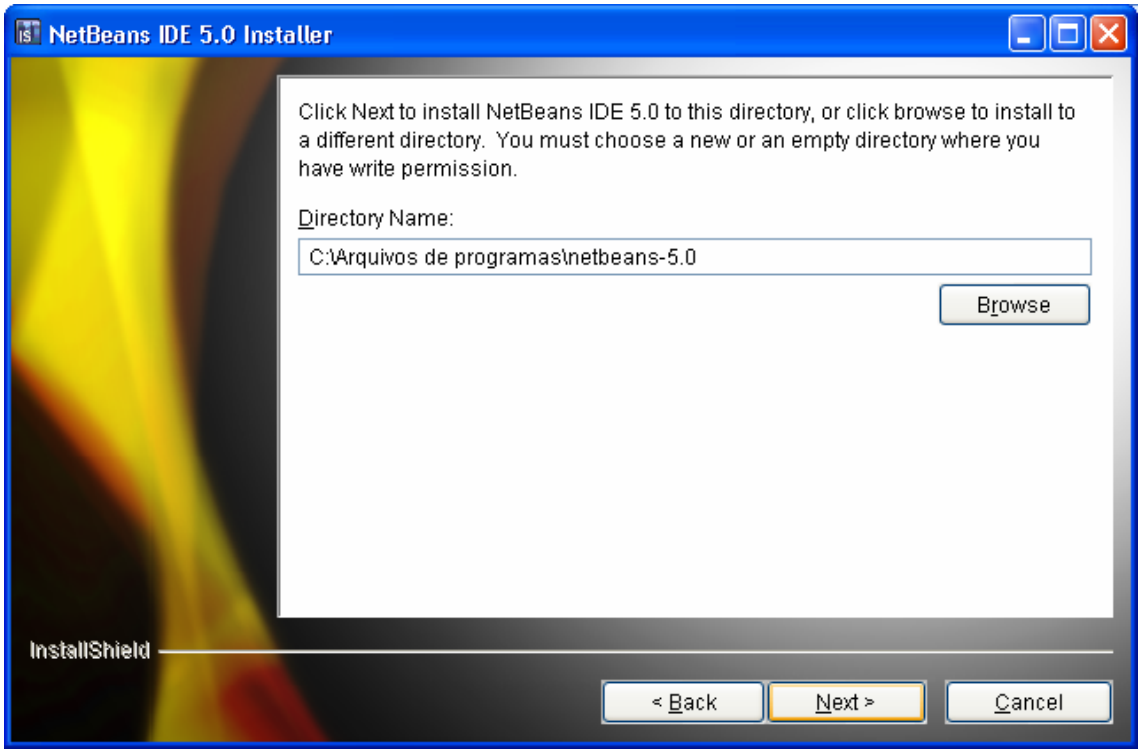

Figura 7: Tela de escolha de diretório de instalação do NetBeans

Em seguida, é necessário selecionar a versão do J2SE JDK que se deseja utilizar como compilador dos programas que irão rodar no NetBeans. Para isso deve-se indicar o local onde o Java está instalado. Por padrão, ele já irá listar as versões existentes no computador, de modo semelhante à lista apresentada na figura 8, sendo que se deve escolher a versão mais recente. Para confirmar a opção selecionada deve-se selecionar o botão "*Next*".

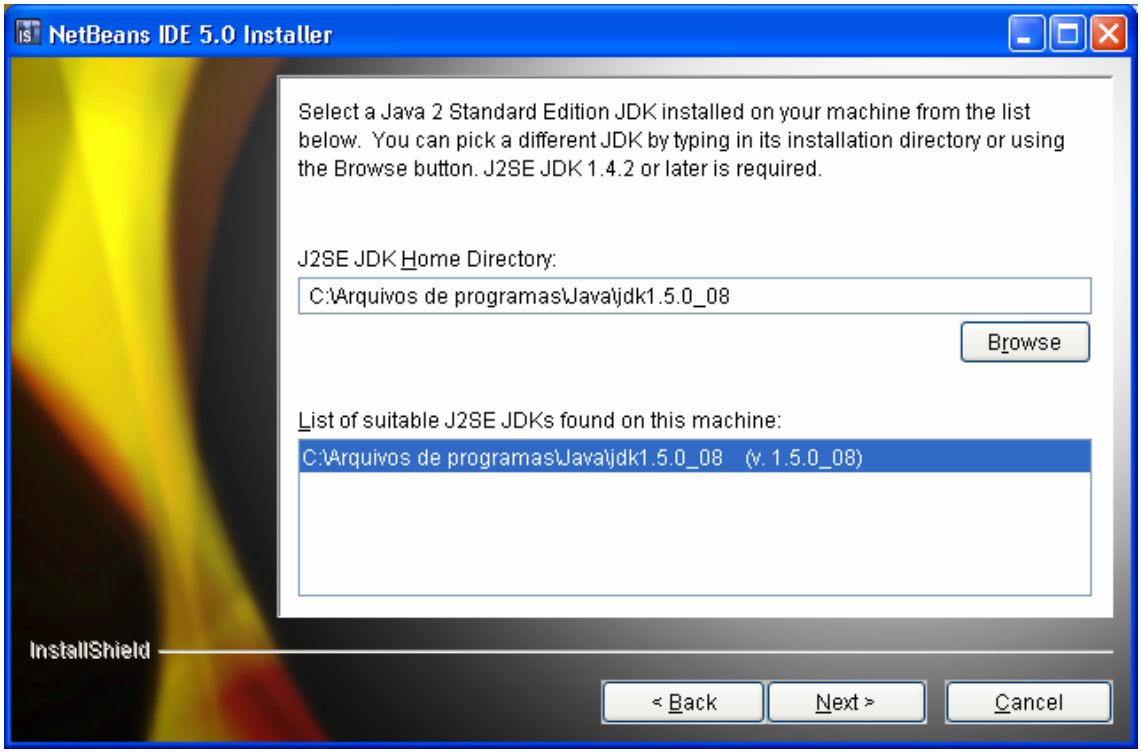

Figura 8: Tela de escolha do compilador JDK

Aparecerá uma tela informando o local selecionado para instalação e o espaço total em disco que o NetBeans ocupará. Para prosseguir com a instalação é necessário selecionar o botão "*Next*".

Posteriormente, deverá aparecer uma tela do assistente informando a conclusão da instalação. Por fim deve-se selecionar o botão "*Finish*".

2.2.3 Configuração do NetBeans

O NetBeans necessita de um pacote de atualização para fornecer um ambiente de desenvolvimento de aplicações de gerenciamento remoto utilizando a API JMX. Para a atualização, faz-se necessário que o computador esteja conectado com a Internet, uma vez que ela será transferida de um servidor externo pertencente ao netbeans.org. Em seguida, com o NetBeans aberto, deve-se escolher a opção "*Update Center*", que se encontra no menu "*Tools*", como exibe a figura 9.

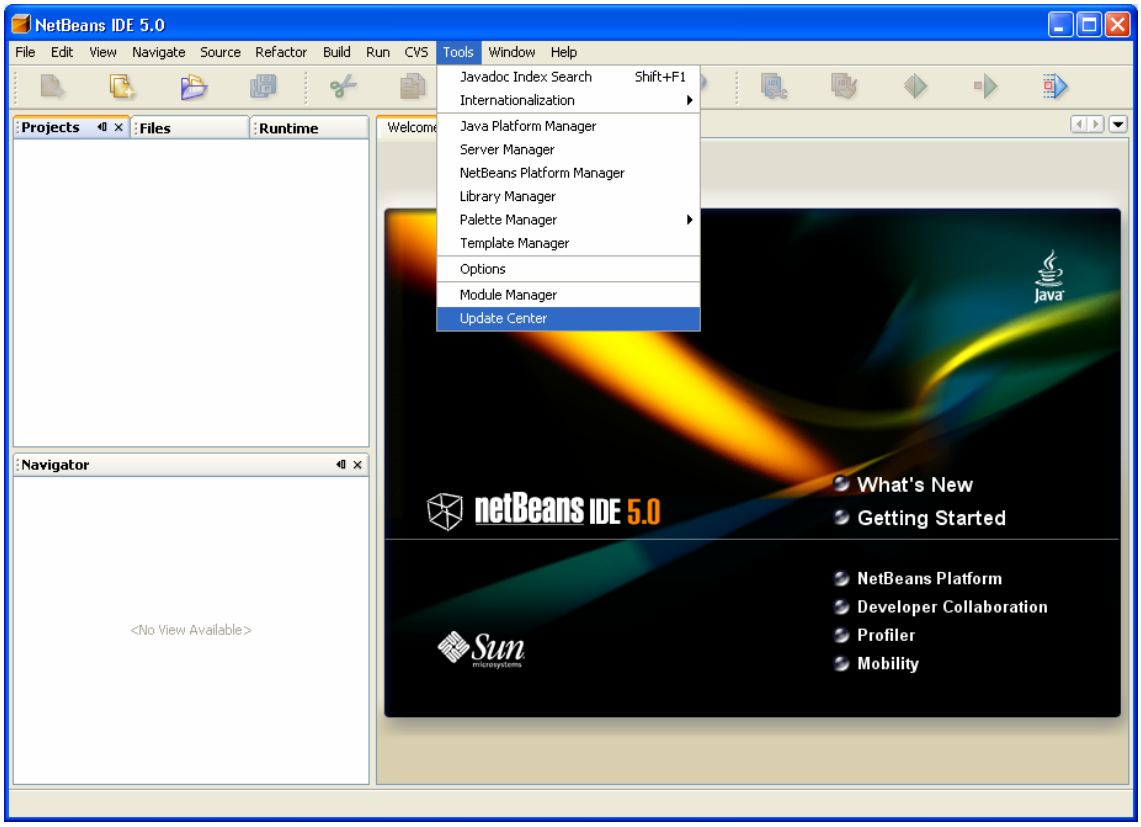

Figura 9: Tela de inicial do NetBeans com menu "*Tools*" expandido

Posteriormente, aparecerá a tela de Seleção de Módulos. Nela, deve-se desmarcar as opções "NetBeans *Hotfix Update Center*" e "*Third-party Update Center*" e manter apenas a opção "NetBeans *Update Center*" marcada, pois o pacote de atualização que será instalado encontra-se no Módulo "NetBeans *Update Center*". A figura 10 apresenta a seleção do módulo. Para prosseguir com a atualização deve-se selecionar o botão "*Next*".

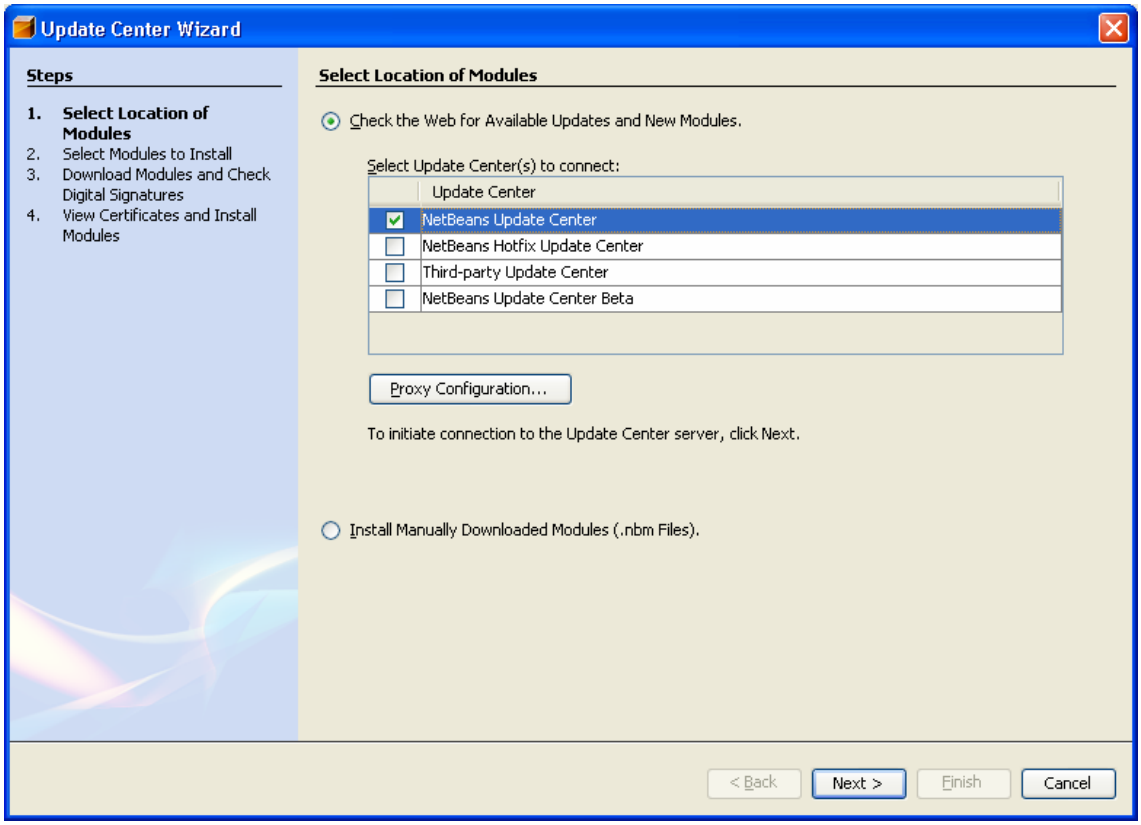

Figura 10: Tela de seleção de módulos de conexão para atualização do NetBeans

Para escolher o módulo de atualização “JMX” deve-se expandir o caminho “NetBeans *Update Center*” e “*Features*”, além de selecionar a opção “JMX”, que se encontra no campo “*Available Updates and New Module:*”.

Logo após, deve-se selecionar o botão “*Add*”. Nessa etapa, o módulo “JMX” aparecerá no campo “*Include in Install:*”, assim como exibe a figura 11, que apresenta a “*Select Modules to Install”* (Seleção de Módulo para Instalação). Deve-se selecionar o botão “*Next*” para confirmar a opção e prosseguir com a atualização.

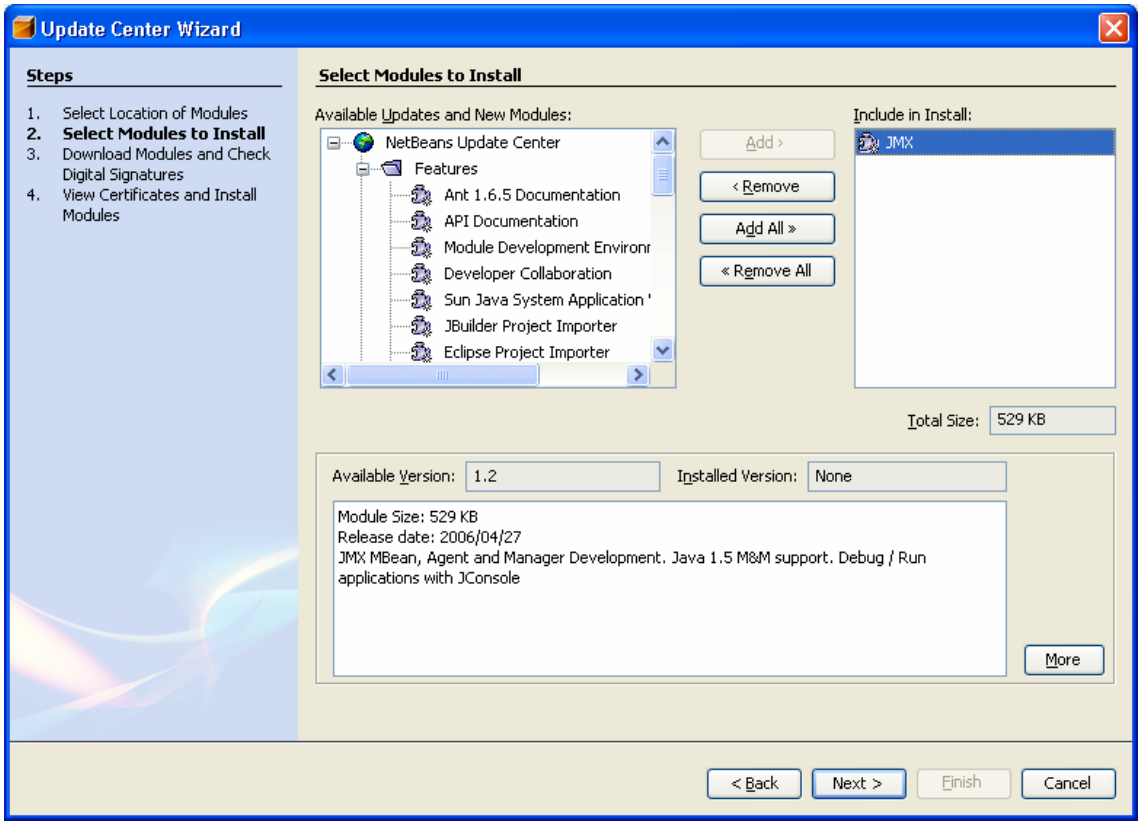

Figura 11: Tela de seleção de módulo para instalação do NetBeans

Para prosseguir com a atualização é necessário concordar com o termo de uso do módulo, para o qual basta selecionar o botão "*Accept*". Logo após, o NetBeans fará o *download* do módulo JMX para o computador.

Quando terminar a transferência da cópia do módulo aparecerá à tela *View Certificates and Install Modules* (Visualização de Certificados e Instalação do Módulo), semelhante à tela que a figura 12 apresenta. Em seguida, deve-se selecionar a opção da coluna "*Global*" para que todos os usuários do computador tenham acesso a esse módulo.

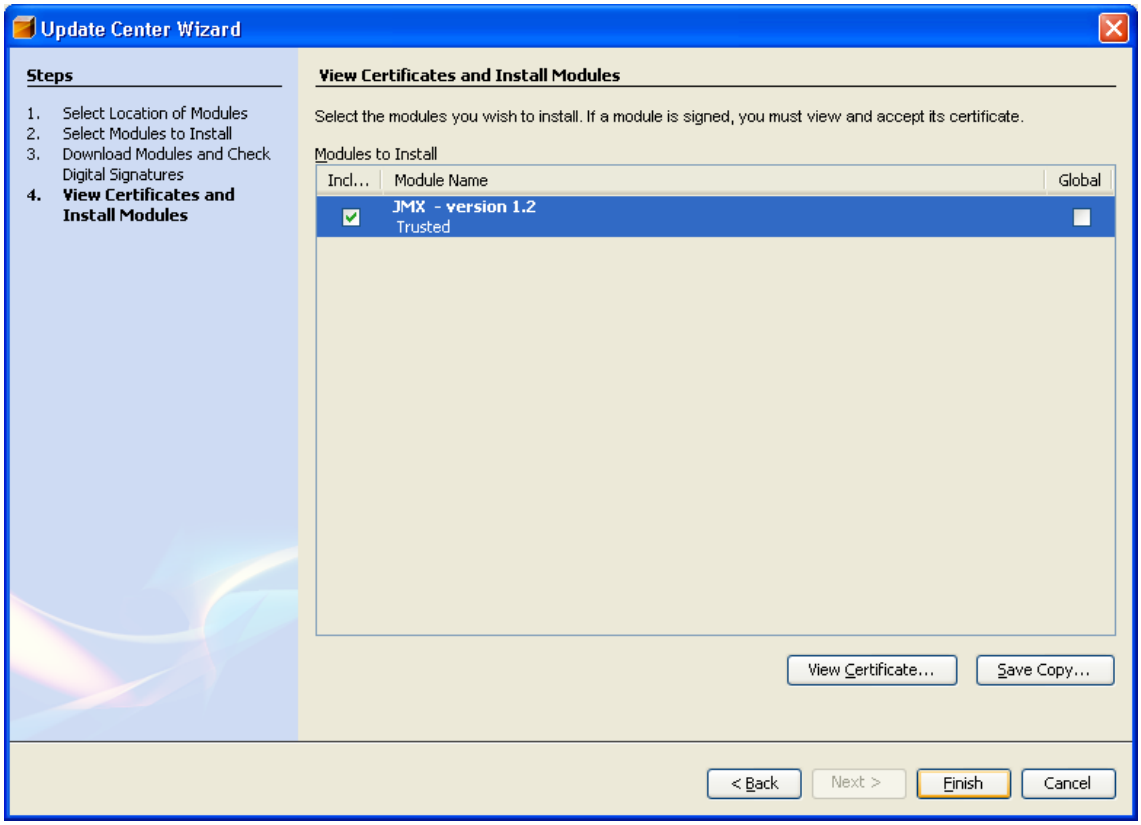

Figura 12: Tela de visualização de certificados e instalação do módulo no NetBeans

Posteriormente, aparecerá uma tela que questionará a seleção da opção de instalação do módulo JMX de forma global. Deve-se selecionar o botão "*Yes*" para confirmar a instalação de forma global.

Por fim, deve-se selecionar o botão "*Finish*" para concluir o uso do assistente de instalação do módulo JMX no NetBeans.

Concluída as etapas citadas nas subseções 2.2.1, 2.2.2 e 2.2.3, o ambiente de desenvolvimento de aplicações com a utilização da API JMX estará preparado para uso na realização do exemplo e teste.

3 Exemplo de Utilização do JMX no NetBeans

O módulo NetBeans JMX permite o desenvolvimento rápido de aplicações de gerência, a inclusão de gerência às aplicações existentes, o desenvolvimento de aplicações gerenciáveis e implementação de um monitor de estado da máquina virtual Java (JVM) (NETBEANS, 2006, *online*)[2].

O NetBeans disponibiliza um exemplo de projeto para o uso do JMX, um jogo de anagramas, que se apresenta como um objeto de teste de desempenho para a API JMX. O exemplo fornece classes do jogo implementadas e interfaces para o gerenciamento.

O Jogo de Anagramas, representado pela classe `Anagrams.java`, consiste na apresentação um conjunto de palavras misturadas no campo *Scrambled Word* (Palavras Misturadas) que deverá ser reapresentado pelo usuário de forma significativa, ou seja, no formato de uma palavra com o mesmo número de letras e usando as mesmas letras oferecidas. Por exemplo, se as letras fornecidas são “obtrasat”, o usuário deverá encontrar a palavra “abstrato” e digitá-la no campo *Your Guess* (Sua Suposição). O Jogo de Anagramas fornece as seguintes opções ao usuário: *Guess* (Suposição), para confirmar uma suposição e *New Word* (Nova Palavra), a fim de que a aplicação forneça um novo conjunto de letras para suposição.

O exemplo aborda a geração de recursos que propiciam a gerência do Jogo de Anagramas, de forma a permitir a monitoração do desempenho de um usuário na tentativa de solucionar os problemas apresentados pelo Jogo cujo código pode ser equiparado a qualquer aplicação Java comum, que necessite ser gerenciada. Para isso, basta apenas analisar os parâmetros a serem gerenciados e, posteriormente, realizar o desenvolvimento das classes para gerir a aplicação.

O desenvolvimento do projeto será descrito nas seções seguintes. Os passos apresentados tiveram por base o projeto disponível em (NETBEANS, 2006, *online*)[2].

3.1 Criando um Projeto

Para a execução do exemplo faz-se necessária a criação de um projeto que facilitará o uso dos testes e sua organização. Para a criação de um projeto no NetBeans deve-se escolher a opção "*New Project*" no menu "*File*" ou utilizar a tecla de atalho (Ctrl+Shift+N). A figura 13 apresenta a tela inicial de execução do NetBeans, com o menu "*File*" expandido.

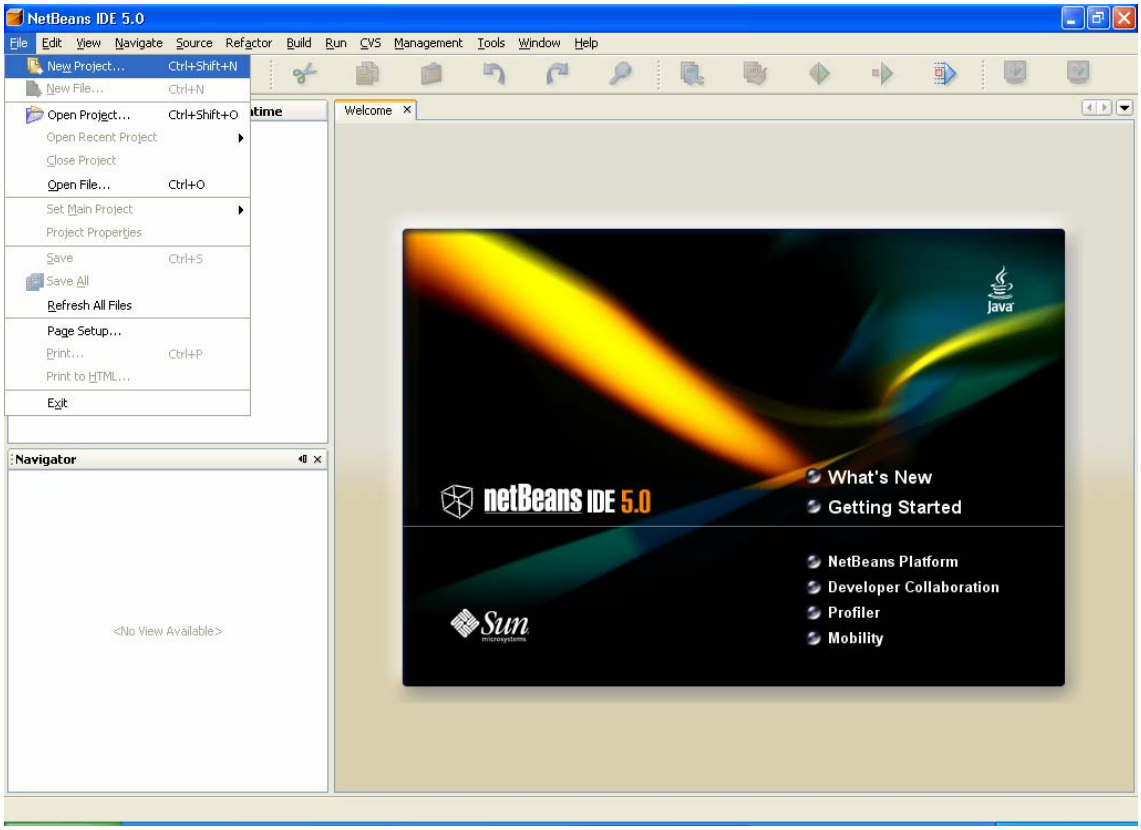

Figura 13: Tela inicial de execução do NetBeans, com o menu "*File*" expandido

Para a criação do projeto JMX é necessário expandir o diretório "*Samples*" no campo "*Categories*". Dentre os vários novos subdiretórios que aparecerá deve-se selecionar o subdiretório "*Management*", e no campo "*Projects*" deve-se selecionar o Projeto "*Anagram Game Managed with JMX*", como apresenta a figura 14, que exibe a tela *Choose Project* (Seleção de Projeto) com o *Anagram Game Managed with JMX* (Jogo de

Anagrama Gerenciável com JMX) selecionado. Por fim será necessário selecionar o botão "*Next*" para prosseguir com a criação do projeto.

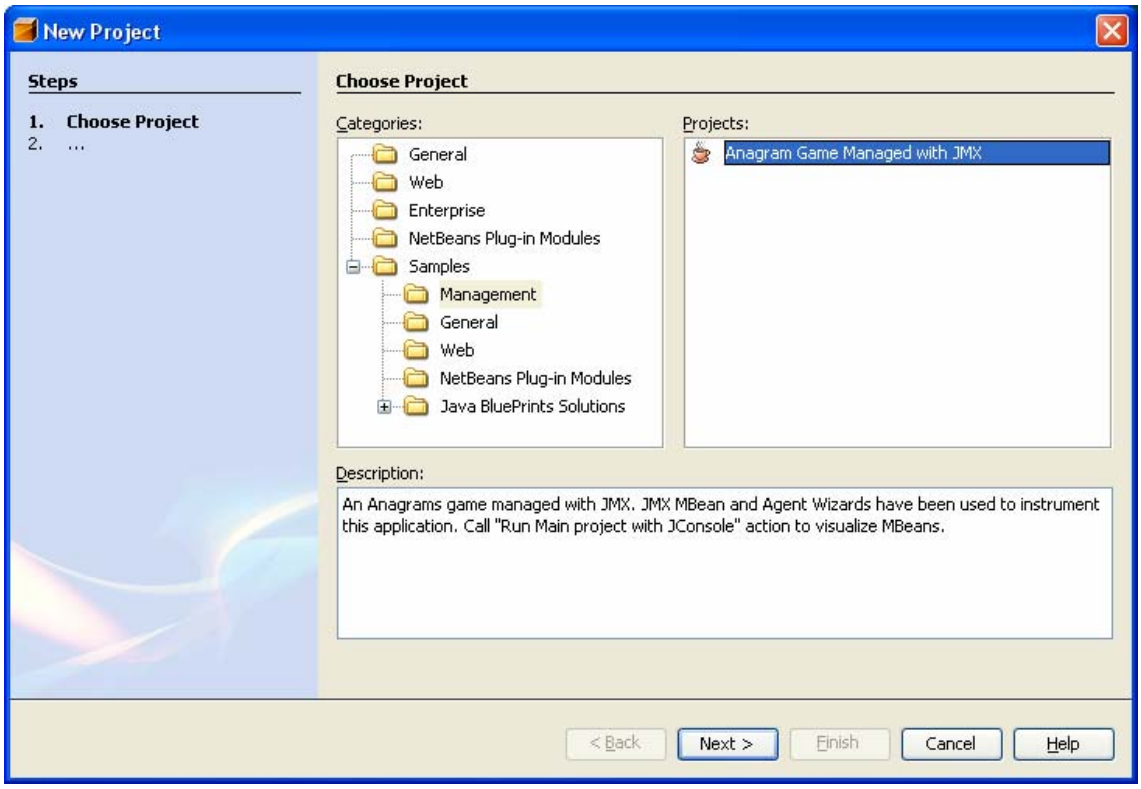

Figura 14: Tela de Seleção de Projeto com o "*Anagram Game Managed with JMX*" selecionado

Para determinar o nome do projeto e o local onde será armazenado, basta alterar respectivamente as informações nos campos "*Project Name*" e "*Project Location*" e selecionar o botão "*Finish*". Para o exemplo deve-se colocar as seguintes informações:

- *Project Name*: AnagramaGerenciavel
- *Project Location*: C:\JMX

Ao selecionar o botão "*Finish*" é criado um diretório como o nome do projeto e dentro do diretório alguns códigos-fonte de classes Java, que representam o modelo do Jogo de Anagramas, por serem padrões serão listados ao final da documentação na seção 5 em anexo.

Pode-se implementar o exemplo aproveitando todos os códigos gerados e fazendo apenas alteração nos mesmos. Porém, para melhor entendimento, só será aproveitado o código principal do projeto a classe "Anagrams.java" que representa a classe *Main*

(principal) do Jogo de Anagramas e a classe “WordLibrary.java” que define as informações para classe principal.

Inicialmente a classe “Anagrams.java” implementa a tela principal do Jogo de Anagramas. Como esta classe foi gerada pelo NetBeans com o propósito de se realizar o gerenciamento com a utilização da API JMX esta será criada com alguns códigos desnecessários (sujeiras) em relação ao código necessários para o teste. Esses códigos desnecessários podem ser excluídos para a realização do teste.

O código da classe “WordLibrary.java” não possui códigos desnecessários (sujeiras) e estará preparado para a realização do teste. Logo, pode-se concluir que qualquer aplicação Java poderá ser gerenciada, necessitando apenas a instrumentação dos recursos que se deseja gerenciar.

3.2 Criando um MBean para a Aplicação

Nesta seção será exemplificada a criação de um JMX MBean que instrumentará recursos da aplicação e que permitirá a monitoração do tempo gasto pelo usuário para resolver o problema de um novo anagrama.

Para iniciar a criação do MBean, deve-se escolher a opção “*New File*” no menu “*File*” ou utilizar a tecla de atalho (Ctrl + N). A figura 15 apresenta a tela de execução do NetBeans, com o menu “*File*” expandido.

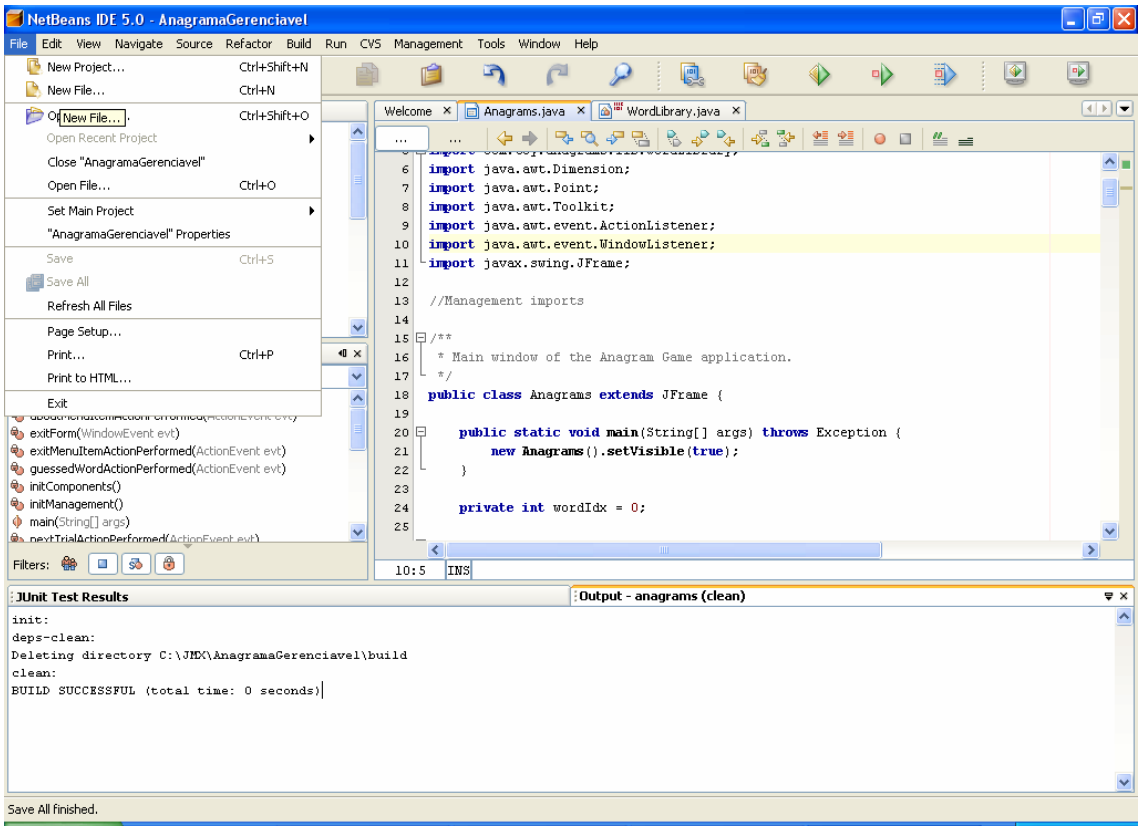

Figura 15: Tela de execução do NetBeans, com o menu “File” expandido

Para a criação do MBean é necessário selecionar o diretório “*Management*” no campo “*Categories*”, e no campo “*File Types*” deve-se selecionar “*JMX MBean*”, como apresenta a figura 16, que exibe a tela *Choose File Type* (Seleção de Tipos de Arquivo) com o *JMX MBean* selecionado. Por fim será necessário selecionar o botão “*Next*” para prosseguir com a criação do MBean.

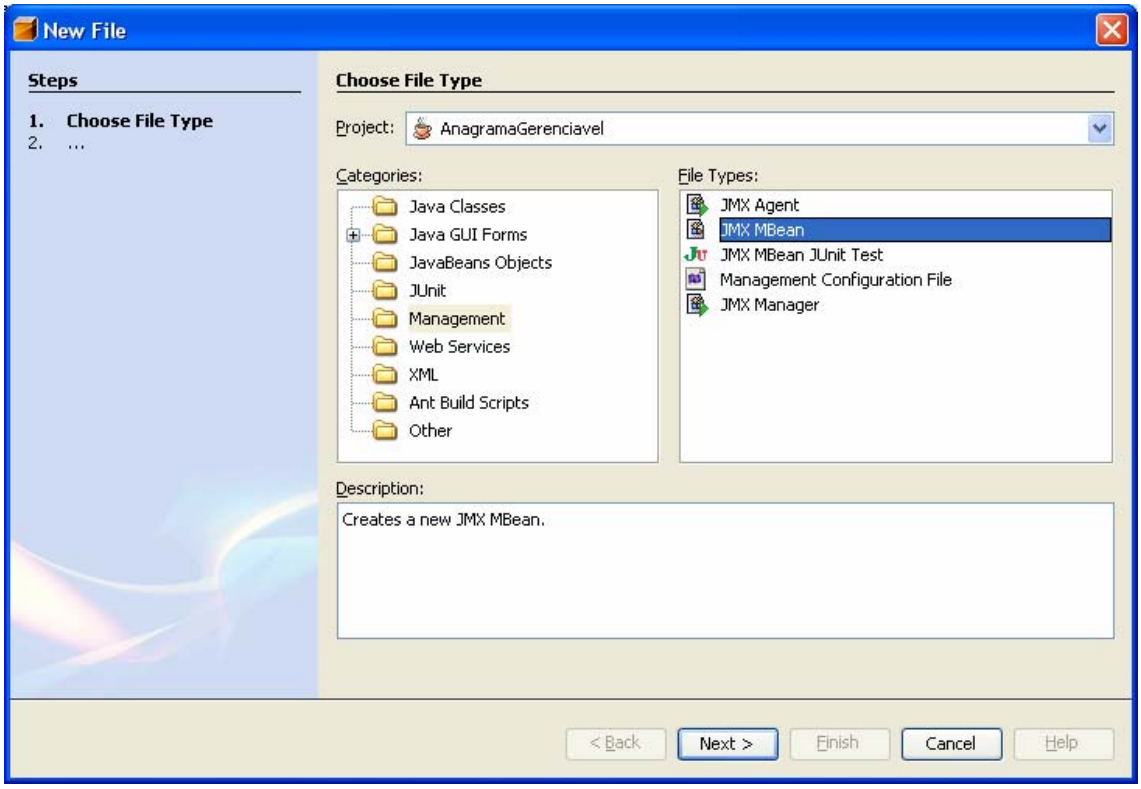

Figura 16: Tela de Seleção de Tipo de Arquivo com o *JMX MBean* selecionado

Para determinar o nome da classe, o local e em qual pacote será armazenada, basta alterar respectivamente as informações "*Class Name*", "*Location*" e "*Package*" e selecionar o botão "*Next*". Para o exemplo deve-se colocar as seguintes informações:

- *Class Name*: MBeanStatusAnagrama
- *Location*: AnagramaGerenciavel
- *Package*: com.toy.anagrams.mbeans

No painel *Type and Options* (Tipo e Opções) deve-se selecionar "*Standard MBean*" como um tipo de MBean. E no campo "*Description*" deve-se adicionar uma descrição, algo do tipo "Monitorando e Gerenciando um Jogo de Anagramas", assim como apresenta a figura 17, que exibe o painel *Type and Options* (Tipo e Opções) do assistente do NetBeans. Para prosseguir com a criação do MBean é necessário selecionar o botão "*Next*".

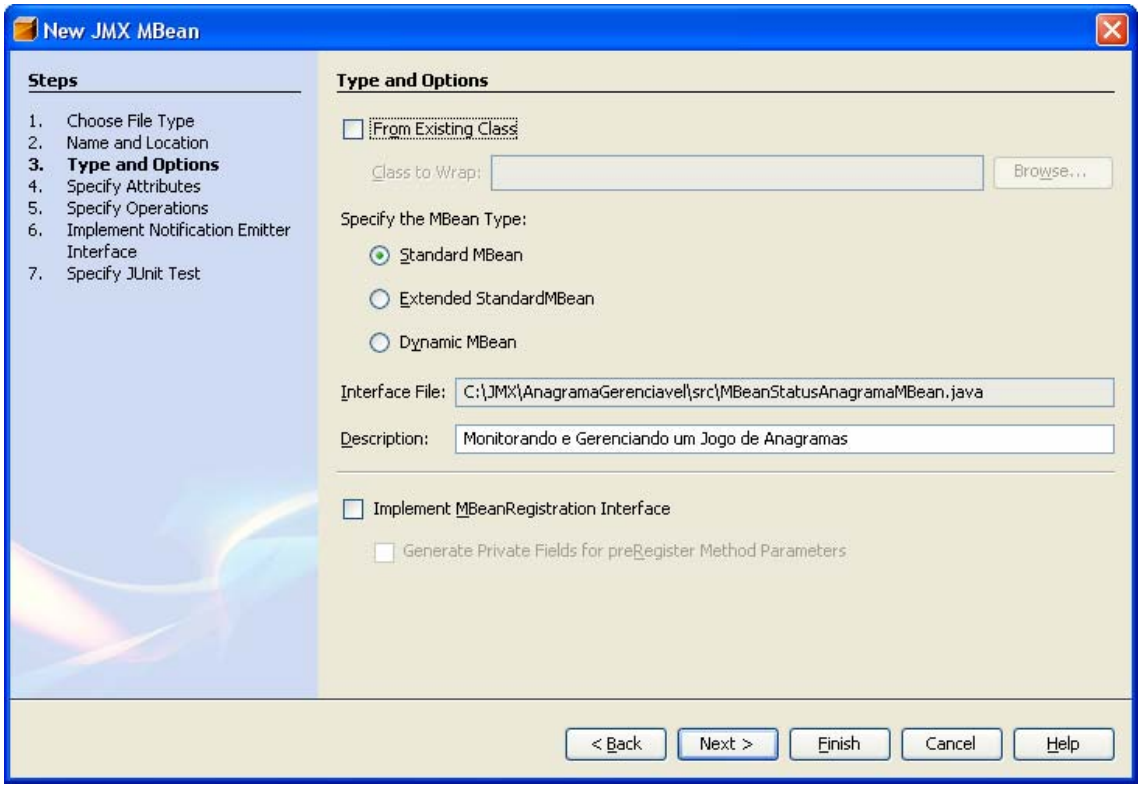

Figura 17: Tela de definição de Tipos e Opções do MBeans

No painel *Specify Attributes* (Especificação de Atributos) será especificado um atributo para o MBean que guardará o tempo gasto na análise da resolução de um anagrama. Para criar o atributo deve-se selecionar o botão "*Add Attribute*". Para o exemplo deve-se colocar as seguintes informações nos respectivos campos:

- *Attribute Name*: `TempoGastoPensando`
- *Type*: `int`
- *Access*: `ReadOnly`
- *Description*: `Tempo Gasto para Resolver`

A figura 18 apresenta a tela *Specify Attributes* (Especificação de Atributos) com os campos preenchidos com os dados listados anteriormente. Após definir as informações do atributo deve-se selecionar o botão "*Next*".

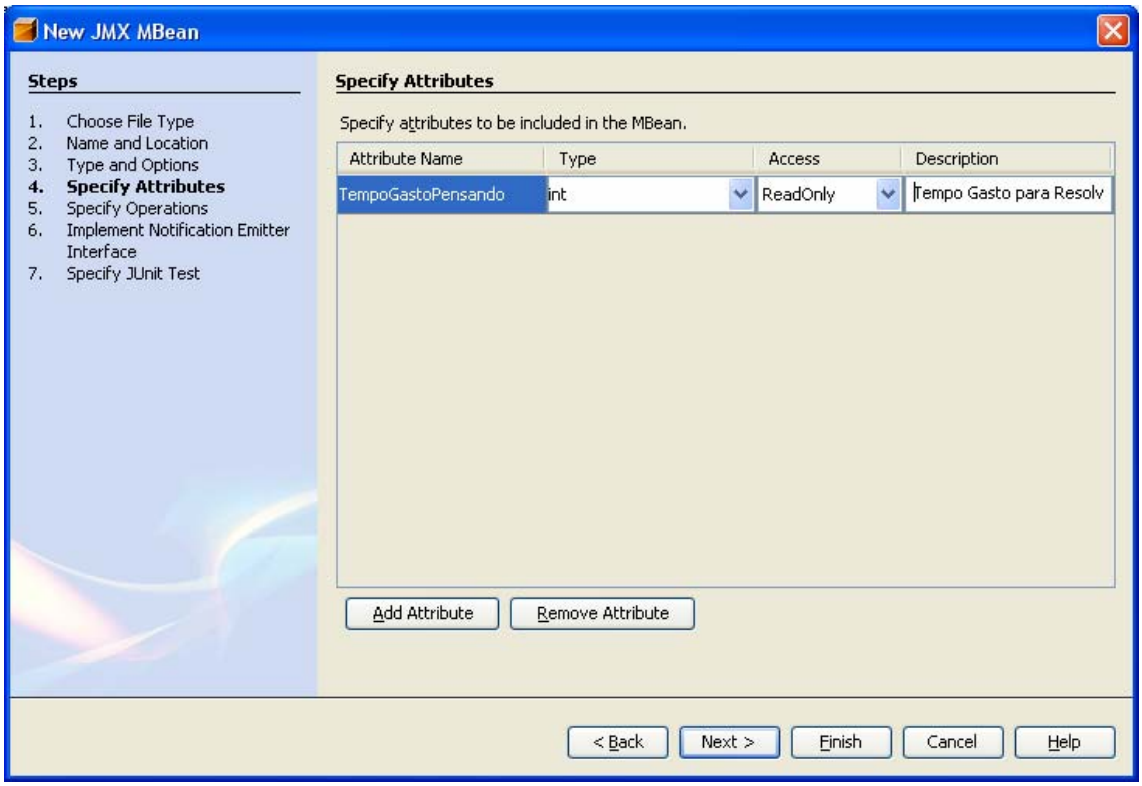

Figura 18: Tela de especificação de atributos com os campos preenchidos

No painel *Specify Operations* (Especificação de Operações) será especificado uma operação (método) que restaura o estado do MBean. Para criar a operação deve-se selecionar o botão "*Add Operation*". Para o exemplo deve-se colocar as seguintes informações nos respectivos campos:

- *Operation Name*: `zerarTodos`
- *Returns*: `void`
- *Parameters*:
- *Exceptions*:
- *Description*: `Restaura o Estado do MBean`

A figura 19 apresenta a tela *Specify Operations* (Especificação de Operações) com os campos preenchidos com os dados listados anteriormente. Após definir as informações da operação deve-se selecionar o botão "*Next*".

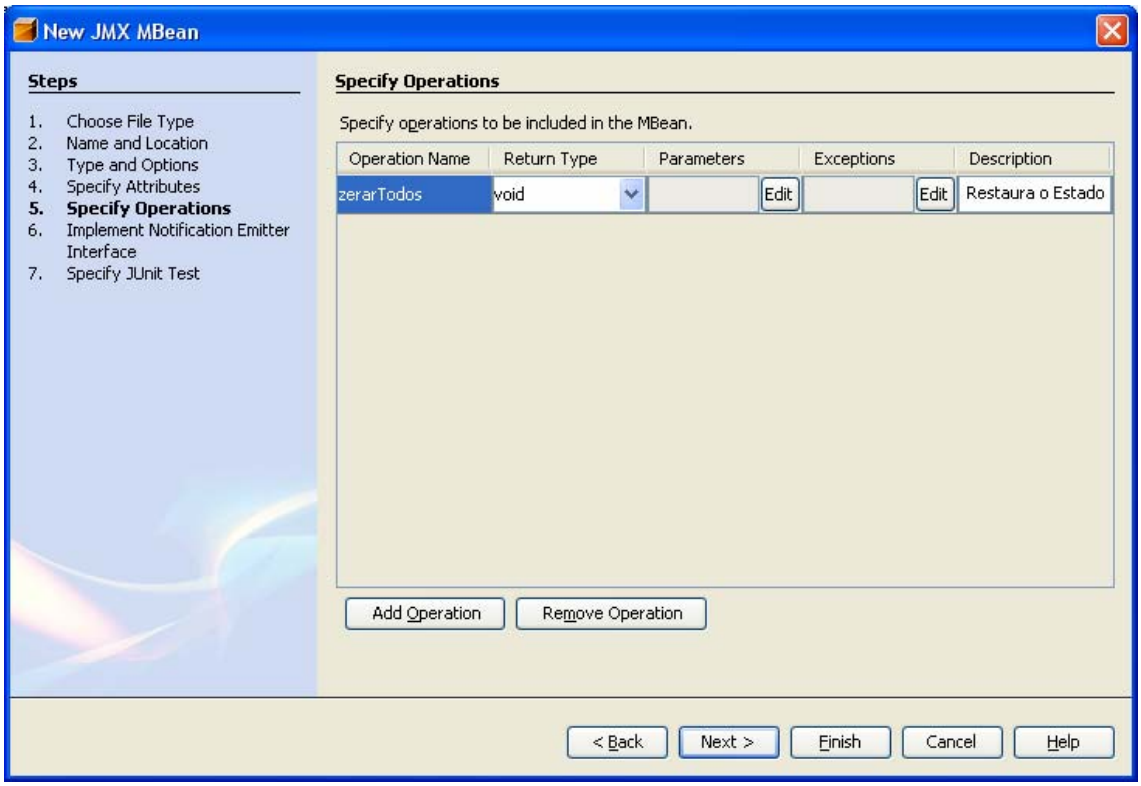

Figura 19: Tela de especificação de operações com os campos preenchidos

No Painel "*Implement Notification Emitter Interface*" (Implementação da Interface do Emissor de Notificação), será necessário ativar cada uma das opções citadas a seguir para a realização dos testes. Estas opções permitirão, respectivamente, criar notificações, delegar difusão da notificação e privar acessos à notificação:

- *Implement NotificationEmitter Interface*
- *Generate Delegation to Broadcaster*
- *Generate Private Seq Number and Accessor*

Após ativar as opções, o botão "*Add Notification*" será habilitado. Para criar uma nova notificação é necessário selecionar este botão. Para a execução do exemplo deve-se colocar as seguintes informações nos respectivos campos:

- *Notification Class*: javax.management.AttributeChange
- *Description*: Anagrama foi Resolvido
- *Notification* Type: ATTRIBUTE_CHANGE

A figura 20 apresenta a tela "*Implement Notification Emitter Interface*" (Implementação de Interface do Emissor de Notificação) com as opções ativadas e os campos preenchidos com os dados listados anteriormente.

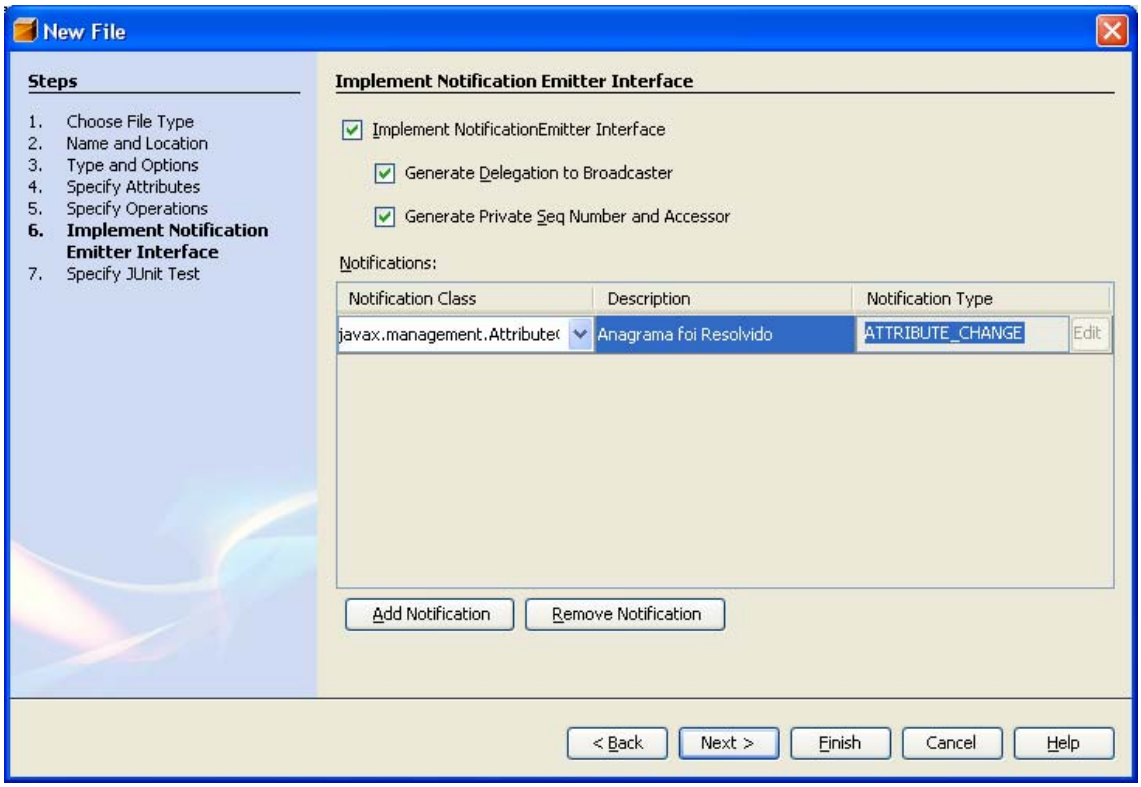

Figura 20: Tela de criação de interface para emissão de notificação com os campos preenchidos

Após o preenchimento das informações da Interface do Emissor de Notificação o MBean estará concluído, mas é possível especificar um JUnit (Java Unit Tests) de teste, ou seja, um ambiente de teste de módulos de códigos Java, escolhendo a opção "*Next*". Porém a criação de um JUnit de teste será abordada minuciosamente na seção 2.3.5 e 2.3.6, sendo necessário apenas selecionar o botão "*Finish*" para finalizar a criação do MBean.

Assim como na criação do projeto, é criado um código Java de implementação do MBean. As figuras 21 e 22, apresentam os códigos gerados na execução da criação do MBean. Foram retirados apenas os comentários.

```
1   package com.toy.anagrams.mbeans;
2   public interface MBeanStatusAnagramaMBean
3   {
4      public int getTempoGastoPensando();
```

```
5       public void zerarTodos();
6   }
```

Figura 21: Interface gerada na implementação do MBean

"MBeanStatusAnagramaMBean.java"

```
1   package com.toy.anagrams.mbeans;
2   import javax.management.*;
3
4   public class MBeanStatusAnagrama implements MBeanStatusAnagramaMBean,
5   NotificationEmitter {
6
7      private int tempoGastoPensando;
8
9      public MBeanStatusAnagrama() {
10     }
11
12     public int getTempoGastoPensando() {
13        return tempoGastoPensando;
14     }
15
16     public void zerarTodos() {
17     }
18
19     public void addNotificationListener(NotificationListener listener,
20  NotificationFilter filter, Object handback) throws IllegalArgumentException {
21        broadcaster.addNotificationListener(listener, filter, handback);
22     }
23
24     public MBeanNotificationInfo[] getNotificationInfo() {
25        return new MBeanNotificationInfo[] {
26           new MBeanNotificationInfo(new String[] {
27              AttributeChangeNotification.ATTRIBUTE_CHANGE},
28              javax.management.AttributeChangeNotification.class.getName(),
29              "Anagrama foi Resolvido")
30           };
31     }
32
33     public void removeNotificationListener(NotificationListener listener) throws
34  ListenerNotFoundException {
35        broadcaster.removeNotificationListener(listener);
36     }
37
38     public void removeNotificationListener(NotificationListener listener,
39  NotificationFilter filter, Object handback) throws ListenerNotFoundException {
40        broadcaster.removeNotificationListener(listener, filter, handback);
41     }
42
43     private synchronized long getNextSeqNumber() {
44        return seqNumber++;
45     }
46
47     private long seqNumber;
48
49     private final NotificationBroadcasterSupport broadcaster = new
```

```
50  NotificationBroadcasterSupport();
51
52  }
```

Figura 22: Classe gerada na implementação do MBean "MBeanStatusAnagrama.java"

Para a criação de classes de MBeans, Agentes JMX e Aplicações de Gerenciamento JMX faz-se necessário o uso de interfaces. Esta prática é comum no desenvolvimento de códigos orientada a objeto, que corresponde a exposição dos métodos e atributos abstratos ao demais objetos permitindo identificar funções que podem ser realizadas pelo objeto. A figura 21 representa o código fonte da interface do MBean gerado nesta seção e a figura 22 representa o código do MBean propriamente dito.

3.3 Configurando o MBean para a Aplicação

Na seção anterior foi criada a estrutura do MBean, que apresenta os códigos-fonte dos atributos e operações (métodos). Nesta seção é apresentado o passo para a adição do atributo NumAnagramasResolvidos ao MBean gerado previamente.

O atributo NumAnagramasResolvidos permitirá contar o número de anagramas que foram resolvidos. Este pode ser implementado utilizando o assistente do NetBeans, não sendo necessária a alteração manual dos códigos e tornando a inserção de atributos e operações (métodos) uma tarefa fácil e prática.

Para criar um novo atributo, seguindo o assistente, deve-se explorar a área do campo "*Project*", expandir o caminho "AnagramaGerenciavel" + "*Source Package*" + "*com.toy.anagrams.mbeans*" e selecionar o MBean "MBeanStatusAnagrama", criado anteriormente. A seguir, deve-se selecionar, no menu "*Management*", a opção "*Add MBeans Attributes...*".

A figura 23 ilustra a tela apresentada antes da seleção da opção "*Add MBeans Attributes...*", com o menu "*Management*" expandido.

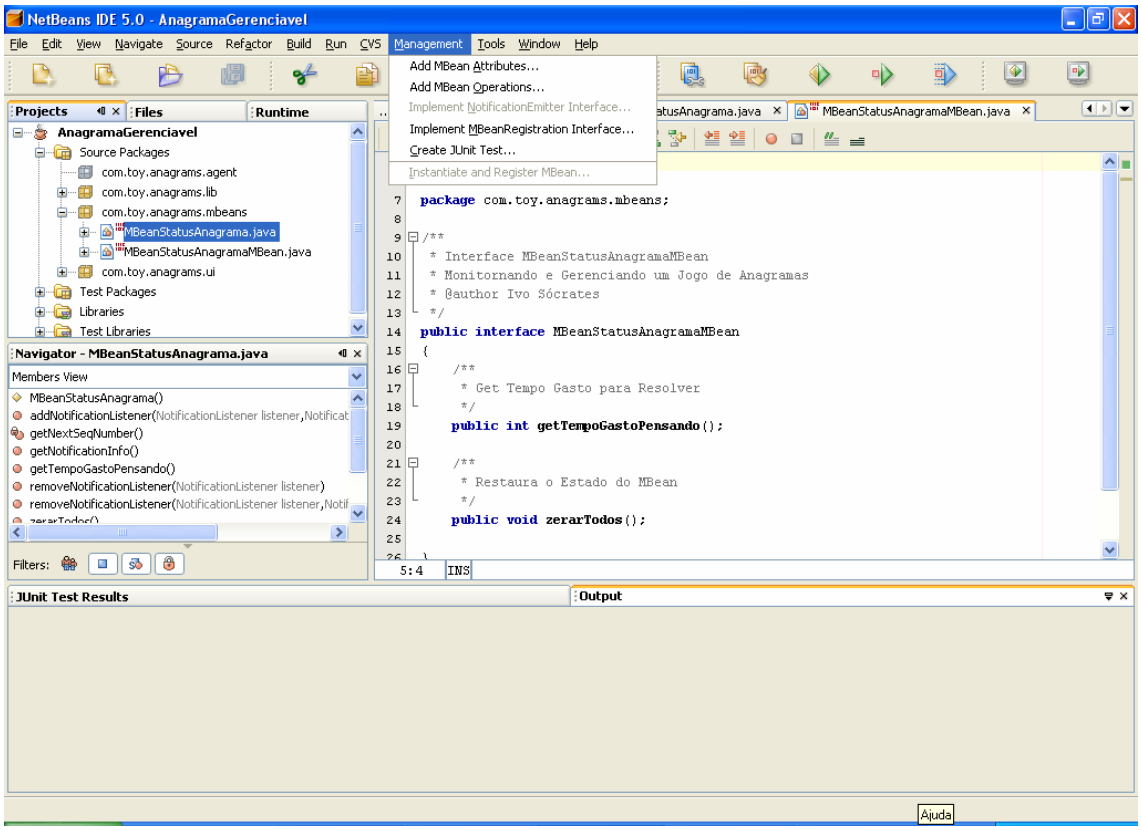

Figura 23: Tela de execução do NetBeans, com o menu "*Management*" expandido

Após a seleção da opção "*Add MBeans Attributes...*" é apresentada a tela *Add MBeans Attributes* (Adição de Atributos ao MBean), onde é necessário adicionar um novo atributo ao MBeans para a realização do teste.

Para criar um novo atributo, deve-se selecionar o botão "*Add Attribute*" no painel *Add MBeans Attributes*. Para o exemplo deve-se colocar as seguintes informações nos respectivos campos:

- *Attribute Name*: NumAnagramasResolvidos
- *Type*: int
- *Access*: ReadOnly
- *Description*: Número de Anagramas Resolvidos

A figura 24 apresenta a tela *Add MBeans Attributes* (Adição de Atributos ao MBean) com os campos preenchidos com os dados listados anteriormente. Após definir as informações do atributo, deve-se selecionar o botão "*OK*" para adicionar um novo atributo.

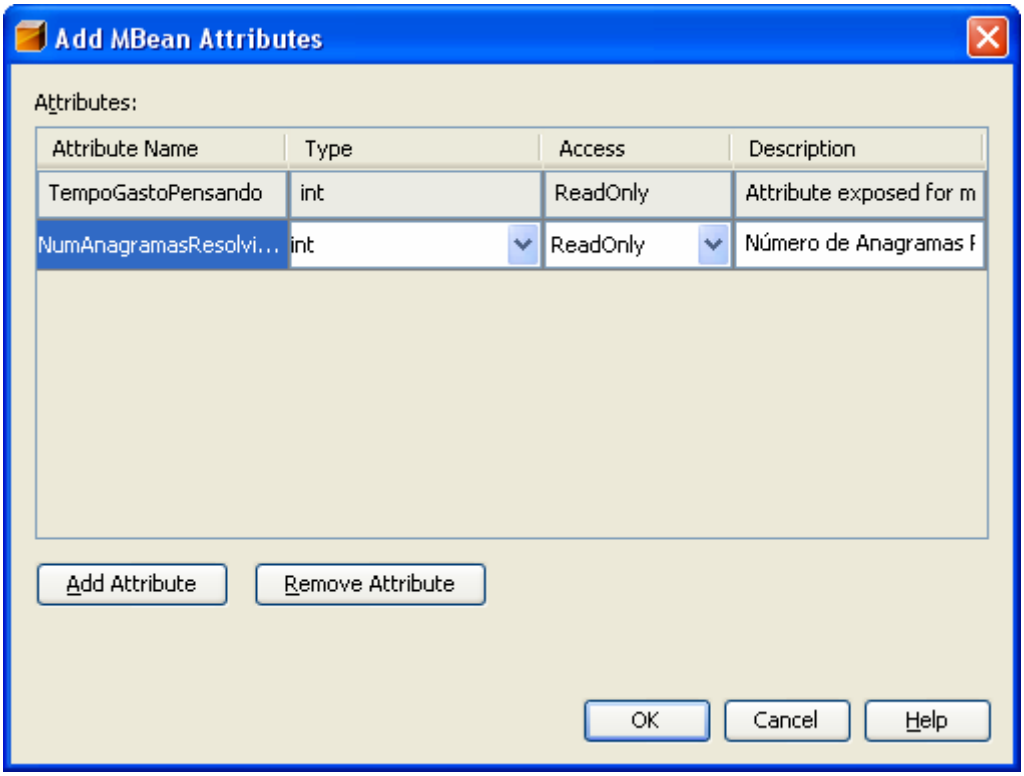

Figura 24: Tela de adição de atributos ao MBean, com algumas informações declaradas

O método "getNumAnagramasResolvidos" será adicionado na extremidade final da classe "MBeanStatusAnagrama.java" e em sua interface, ou seja na classe "MBeanStatusAnagramaMBean.java", como apresentam as figuras 25 e 26, respectivamente, com os códigos-fonte das classes citadas.

```
1  package com.toy.anagrams.mbeans;
2  public interface MBeanStatusAnagramaMBean
3  {
4     public int getTempoGastoPensando();
5     public void zerarTodos();
6     public int getNumAnagramasResolvidos();
7  }
```

Figura 25: Interface do MBean "MBeanStatusAnagramaMBean.java" alterada

```
1   package com.toy.anagrams.mbeans;
2   import javax.management.*;
3
4   public class MBeanStatusAnagrama implements MBeanStatusAnagramaMBean,
5   NotificationEmitter {
6      private int numAnagramasResolvidos;
7
8      private int tempoGastoPensando;
9
```

```
10      public MBeanStatusAnagrama() {
11      }
12
13      public int getTempoGastoPensando() {
14          return tempoGastoPensando;
15      }
16
17      public void zerarTodos() {
18      }
19
20      public void addNotificationListener(NotificationListener listener,
21  NotificationFilter filter, Object handback) throws IllegalArgumentException {
22          broadcaster.addNotificationListener(listener, filter, handback);
23      }
24
25      public MBeanNotificationInfo[] getNotificationInfo() {
26          return new MBeanNotificationInfo[] {
27              new MBeanNotificationInfo(new String[] {
28                  AttributeChangeNotification.ATTRIBUTE_CHANGE},
29                  javax.management.AttributeChangeNotification.class.getName(),
30                  "Anagrama foi Resolvido")
31              };
32      }
33
34      public void removeNotificationListener(NotificationListener listener) throws
35  ListenerNotFoundException {
36          broadcaster.removeNotificationListener(listener);
37      }
38
39      public void removeNotificationListener(NotificationListener listener,
40  NotificationFilter filter, Object handback) throws ListenerNotFoundException {
41          broadcaster.removeNotificationListener(listener, filter, handback);
42      }
43
44      private synchronized long getNextSeqNumber() {
45          return seqNumber++;
46      }
47
48      private long seqNumber;
49
50      private final NotificationBroadcasterSupport broadcaster = new
51  NotificationBroadcasterSupport();
52
53      public int getNumAnagramasResolvidos() {
54          return numAnagramasResolvidos;
55      }
56
57  }
```

Figura 26: Classe do MBean “MBeanStatusAnagrama.java” alterada

O MBean é atualizado com um novo atributo. Na próxima seção serão feitas algumas alterações no MBean. Estas alterações proporcionarão um funcionamento completo ao MBean.

3.4 Implementando o MBean para a Aplicação

Nesta seção serão inseridos elementos que proporcionam gerenciamento ao MBean (MBeanStatusAnagrama). Inicialmente, deve-se observar que os atributos "NumAnagramasResolvidos" e "TempoGastoPensando" não necessitam de qualquer atualização. Porém o método "zerarTodos" necessita ser implementado, pois está vazio. Quando o método "zerarTodos" for chamado os contadores serão ajustados para 0 (zero). Para a realização do exemplo deve-se inserir os códigos apresentados na figura 27 no corpo do método "zerarTodos" da classe "MBeanStatusAnagrama":

```
1   tempoGastoPensando = 0;
2   numAnagramasResolvidos = 0;
```

Figura 27: Código com atribuição de valores a ser inserido no corpo do método "zerarTodos"

Para a realização do exemplo, além de inserir atribuição de valores no corpo do método "zerarTodos", deve-se inserir alguns métodos na classe "MBeanStatusAnagrama". Isso para que a classe "*Anagrams*" notifique o MBean que um novo anagrama foi proposto para o usuário ou que um anagrama foi resolvido.

O código apresentado na figura 28 servirá para criar e enviar uma notificação quando um anagrama for resolvido este deve ser inserido antes do último símbolo de fechamento de uma classe.

```
1       /**
2        * Os métodos expõe os componentes do Anagrams UI para fornecer dados a
3    gerência.
4        */
5
6       private long startTime;
7       /*
8        * Um novo Anagrama é proposto ao usuário. O usuário começa pensar...
9       */
10      public void startThinking() {
11          startTime = System.currentTimeMillis();
12      }
13      /*
14       * Um Anagrama foi resolvido.
15       */
16      public void stopThinking() {
```

```
17
18          //Atualiza o número de anagramas resolvidos
19          numAnagramasResolvidos++;
20
21          // Calcula o tempo gasto pensando (em segundos).
22          long stopTime = System.currentTimeMillis();
23
24          tempoGastoPensando = (int) (stopTime - startTime) / 1000 ;
25
26          //Cria uma notificação JMX para notificar a definição do Anagrama.
27          Notification notification = new
28            Notification(AttributeChangeNotification.ATTRIBUTE_CHANGE,
29                              this,
30                              getNextSeqNumber(),
31                              "Anagrama Resolvido!");
32
33          // Envia a notificação JMX.
34          broadcaster.sendNotification(notification);
35      }
```

Figura 28: Código com métodos que expõe os componentes do Anagrama a ser inserido no corpo da classe “MBeanStatusAnagrama”

Deve ser observado que uma notificação do JMX do tipo ATTRIBUTE_CHANGE será enviada cada vez que um anagrama for resolvido. Após a realização das alterações abordadas nesta seção, o código fonte da classe “MBeanStatusAnagrama.java” deverá ser semelhante ao que é apresentado na figura 29, porém com alguns comentários.

```
1   package com.toy.anagrams.mbeans;
2   import javax.management.*;
3
4   public class MBeanStatusAnagrama implements MBeanStatusAnagramaMBean,
5   NotificationEmitter {
6
7       private int numAnagramasResolvidos;
8
9       private int tempoGastoPensando;
10
11      public MBeanStatusAnagrama() {
12      }
13
14      public int getTempoGastoPensando() {
15          return tempoGastoPensando;
16      }
17
18      public void zerarTodos() {
19          tempoGastoPensando = 0;
20          numAnagramasResolvidos = 0;
21      }
22
23      public void addNotificationListener(NotificationListener listener,
24  NotificationFilter filter, Object handback) throws IllegalArgumentException {
25          broadcaster.addNotificationListener(listener, filter, handback);
```

```
26      }
27
28      public MBeanNotificationInfo[] getNotificationInfo() {
29          return new MBeanNotificationInfo[] {
30              new MBeanNotificationInfo(new String[] {
31                  AttributeChangeNotification.ATTRIBUTE_CHANGE},
32                  javax.management.AttributeChangeNotification.class.getName(),
33                  "Anagrama foi Resolvido")
34              };
35      }
36
37      public void removeNotificationListener(NotificationListener listener) throws
38  ListenerNotFoundException {
39          broadcaster.removeNotificationListener(listener);
40      }
41
42      public void removeNotificationListener(NotificationListener listener,
43  NotificationFilter filter, Object handback) throws ListenerNotFoundException {
44          broadcaster.removeNotificationListener(listener, filter, handback);
45      }
46
47      private synchronized long getNextSeqNumber() {
48          return seqNumber++;
49      }
50
51      private long seqNumber;
52
53      private final NotificationBroadcasterSupport broadcaster = new
54  NotificationBroadcasterSupport();
55
56      public int getNumAnagramasResolvidos() {
57          return numAnagramasResolvidos;
58      }
59
60      private long startTime;
61
62      public void startThinking() {
63          startTime = System.currentTimeMillis();
64      }
65
66      public void stopThinking() {
67
68          numAnagramasResolvidos++;
69
70          long stopTime = System.currentTimeMillis();
71
72          tempoGastoPensando = (int) (stopTime - startTime) / 1000 ;
73
74          Notification notification = new
75            Notification(AttributeChangeNotification.ATTRIBUTE_CHANGE,
76                         this,
77                         getNextSeqNumber(),
78                         "Anagrama Resolvido!");
79
80          broadcaster.sendNotification(notification);
81      }
```

```
82  }
```

Figura 29: Código completo da classe “MBeanStatusAnagrama” após alterações

Nas seções anteriores foram explicadas a criação de um Projeto de um Jogo de Anagramas e a implementação de um MBean padrão, que enfocou duas partes: a criação de atributos e de operações (métodos) da gerência e a criação de operações (métodos) para permitir atualizações no estado do MBean.

Na seção seguinte é apresentada a criação de JUnit de teste para verificar o funcionamento dos atributos e operações (métodos) do MBean. Esse teste permite saber se o MBean criado está funcionando corretamente.

3.5 Criando um JUnit de Teste para o MBeans

O JUnit é um ambiente de teste de módulos de códigos Java que serve para automatizar a criação de casos de teste, evitando testes duplicados e permitindo escrever testes com valores de longo prazo, ou seja, que possam ser reutilizáveis (VAZ, 2003, p. 16-17).

Nesta seção são abordados os métodos de criação de um JUnit de teste para um MBean, que realizará testes baseados nos atributos e nas operações (métodos) do MBean “MBeanStatusAnagrama”.

Para o gerenciamento de uma aplicação não se faz necessária a criação do JUnit, porém, para garantir que o MBean esteja em perfeito funcionamento deve-se gerar o JUnit de teste. A prática de teste em MBean é importante para reduzir erros na instrumentação dos recursos que se pretende gerenciar.

Para criar um JUnit, seguindo o assistente, deve-se explorar a área do campo “*Project*”, expandir o caminho “AnagramaGerenciavel” + “*Source Package*” + “*com.toy.anagrams.mbeans*” e selecionar o MBean “MBeanStatusAnagrama”, criado anteriormente. A seguir, deve-se selecionar, no menu “*Management*”, a opção “*Create JUnit Test*”. A figura 30 ilustra a tela apresentada antes da seleção da opção “*Create JUnit Test*”, com o menu “*Management*” expandido.

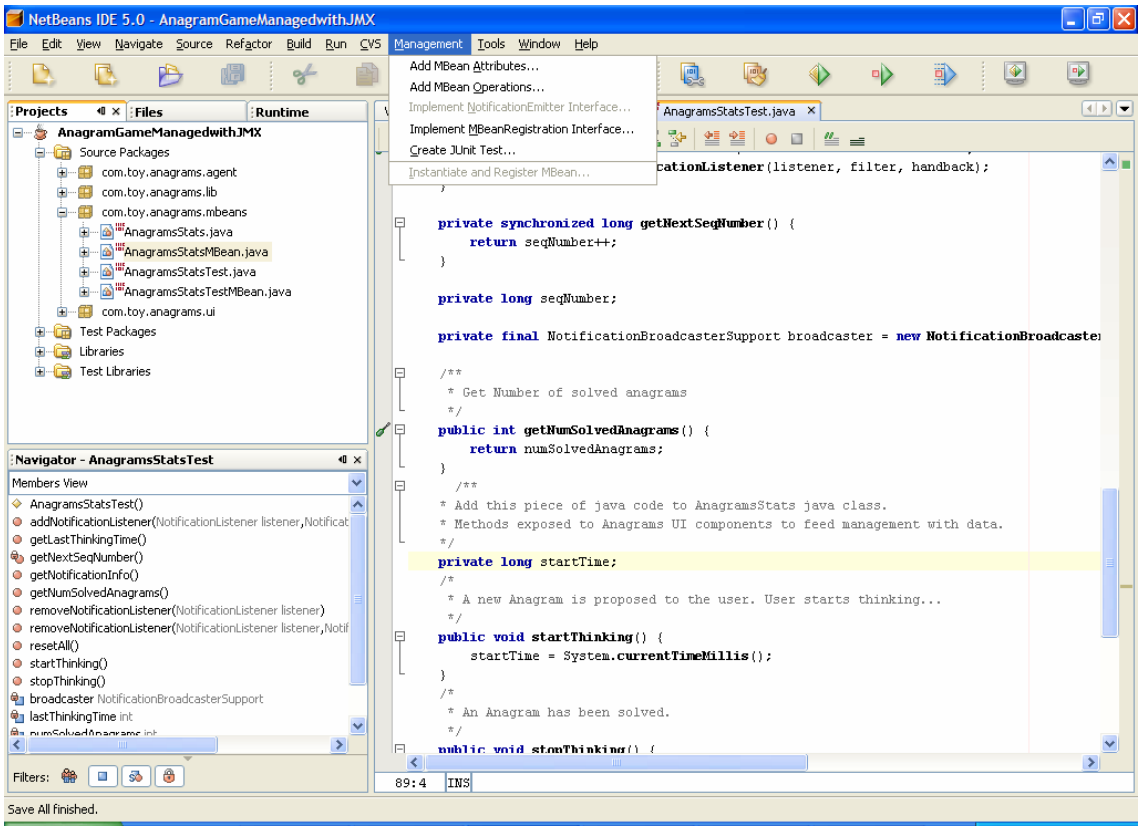

Figura 30: Tela de execução do NetBeans, com o menu "*Management*" expandido

Após a seleção da opção "*Create JUnit Test*" é apresentada a tela *Create JUnit Test* (Criação de um JUnit de Teste), onde será proposta uma nova classe de teste JUnit para testar o MBeans.

No campo *Class to Test* (Classe para Teste) deverá aparecer a seguinte informação: com.toy.anagrams.mbeans.MBeanStatusAnagrama. No campo *Created to Test Class* (Classe Criada para Testar) deverá conter a seguinte informação: com.toy.anagrams.mbeans.MBeanStatusAnagramaTest. No campo *Location* (Local) deverá estar selecionada a opção "*Test Packages*" e no campo *MBeans Constructor* (Construtor do MBeans) a opção "MBeanStatusAnagrama()" deverá estar selecionada.

A figura 31 apresenta a tela *Create JUnit Test* (Criação de um JUnit de Teste) com as opções ativadas. Caso alguma destas opções não esteja ativada deve-se selecioná-las para a criação do JUnit e escolher a opção "*OK*".

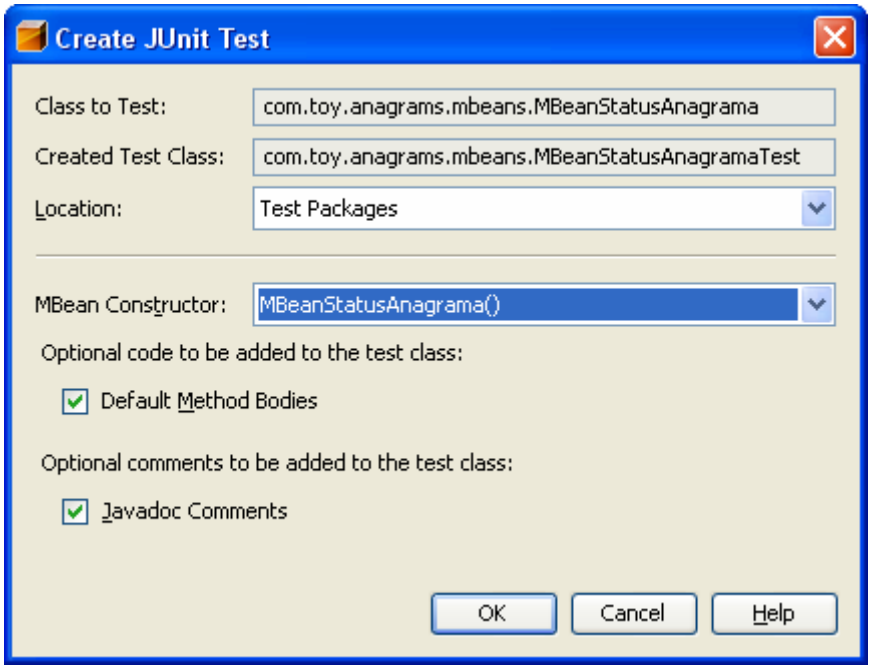

Figura 31: Tela do assistente de criação de um JUnit de teste

Após selecionar o botão "*OK*" o NetBeans abrirá no editor de código com a classe de teste JUnit "MBeanStatusAnagramaTest" que foi gerada. A figura 32 apresenta o código-fonte da classe de teste JUnit criado.

```
1    package com.toy.anagrams.mbeans;
2    import junit.framework.*;
3    import javax.management.*;
4    import java.lang.management.ManagementFactory;
5
6    public class MBeanStatusAnagramaTest extends TestCase {
7
8        public MBeanStatusAnagramaTest(String testName) {
9            super(testName);
10       }
11
12       public void testNumAnagramasResolvidosAttribute() throws Exception {
13           System.out.println("testNumAnagramasResolvidosAttribute");
14           Integer val = (Integer) getAttribute("NumAnagramasResolvidos");
15           fail("The test case is empty.");
16       }
17
18       public void testTempoGastoPensandoAttribute() throws Exception {
19           System.out.println("testTempoGastoPensandoAttribute");
20           Integer val = (Integer) getAttribute("TempoGastoPensando");
21           fail("The test case is empty.");
22       }
23
24       public void testZerarTodosOperation() throws Exception {
25           System.out.println("testzerarTodosOperation");
```

```
26          String[] signature = new String[] {
27          };
28          Object[] params = new Object[] {
29          };
30          invoke("zerarTodos",params,signature);
31          fail("The test case is empty.");
32      }
33
34      public Object createMBean() {
35          return new MBeanStatusAnagrama();
36      }
37
38      protected void setUp() throws Exception {
39          jmxInit();
40      }
41
42      protected void tearDown() throws Exception {
43           jmxTerminate();
44      }
45
46      public static Test suite() {
47          TestSuite suite = new TestSuite(MBeanStatusAnagramaTest.class);
48          return suite;
49      }
50
51      private Object getAttribute(String attName) throws Exception {
52          return getMBeanServer().getAttribute(mbeanName, attName);
53      }
54
55      private void setAttribute(Attribute att) throws Exception {
56          getMBeanServer().setAttribute(mbeanName, att);
57      }
58
59      private Object invoke(String opName, Object params[],
60              String signature[]) throws Exception {
61          return getMBeanServer().invoke(mbeanName, opName, params,
62  signature);
63      }
64
65      private void jmxInit() throws Exception {
66          mbeanName = new ObjectName(":type=MBeanStatusAnagrama");
67          Object mbean = createMBean();
68          server = ManagementFactory.getPlatformMBeanServer();
69          server.registerMBean(mbean, mbeanName);
70      }
71
72      private void jmxTerminate() throws Exception {
73          server.unregisterMBean(mbeanName);
74      }
75
76      private MBeanServer getMBeanServer() {
77          return server;
78      }
79
80      private MBeanServer server;
81      private ObjectName mbeanName;
```

```
82
83    }
```

Figura 32: Classe gerada na criação do teste JUnit “MBeanStatusAnagramaTest.java”

3.6 Configurando o JUnit para Testar o MBeans

Nesta seção é abordada a configuração do JUnit, que é baseada na alteração da classe JUnit “MBeanStatusAnagramaTest.java”. Deve-se adicionar elementos essenciais que permitirão o teste do MBean, gerado anteriormente; e também será abordada a execução do teste do JUnit.

Inicialmente deve-se adicionar uma declaração simples em cada caso de teste. Para cada atributo do MBean, uma operação (método) no formato test[NomeAtributo]Attribute() será gerada. Nos métodos, os atributos do MBean serão obtidos e analisados.

Para a realização do teste faz-se necessária a alteração do código da classe do JUnit “MBeanStatusAnagramaTest.java”. Deve-se realizar a substituição manual do código no método testTempoGastoPensandoAttribute(). Substitui-se a chamada “fail("The test case is empty");” por “assertTrue("O tempo gasto analisado deve ser igual a 0", val == 0);”; e no método testNumAnagramasResolvidosAttribute(), substitui-se a chamada “fail("The test case is empty");” por “assertTrue("O número de Anagramas resolvidos deve ser igual a 0", val == 0);”.

Para cada operação (método) do MBean, uma operação (método) no formato test[NomeOperacao]Operation será gerada no JUnit. Neste método, a operação será chamada. Para a realização do teste fazem-se necessárias alterações no código da classe do JUnit “MBeanStatusAnagramaTest.java”. Deve-se realizar a substituição manual do código no método testZerarTodosOperation() e faz-se necessária a alteração da chamada “fail("The test case is empty");” para “assertTrue("Método invocado com sucesso!", true);”.

Após a realização das alterações, o código fonte da classe “MBeanStatusAnagramaTest.java” deverá ser semelhante ao que é apresentado na figura 33, porém sem alguns comentários.

```
1     package com.toy.anagrams.mbeans;
2     import junit.framework.*;
```

```
3    import javax.management.*;
4    import java.lang.management.ManagementFactory;
5
6    public class MBeanStatusAnagramaTest extends TestCase {
7
8       public MBeanStatusAnagramaTest(String testName) {
9          super(testName);
10      }
11
12      public void testNumAnagramasResolvidosAttribute() throws Exception {
13         System.out.println("testNumAnagramasResolvidosAttribute");
14
15         Integer val = (Integer) getAttribute("NumAnagramasResolvidos");
16
17         assertTrue("O número de Anagramas resolvidos deve ser igual a 0", val
18   == 0);
19      }
20
21      public void testTempoGastoPensandoAttribute() throws Exception {
22         System.out.println("testTempoGastoPensandoAttribute");
23
24         Integer val = (Integer) getAttribute("TempoGastoPensando");
25
26         assertTrue("O tempo gasto analisado deve ser igual a 0", val == 0);
27      }
28
29      public void testZerarTodosOperation() throws Exception {
30         System.out.println("testzerarTodosOperation");
31
32         String[] signature = new String[] {
33         };
34
35         Object[] params = new Object[] {
36         };
37
38         invoke("zerarTodos",params,signature);
39         assertTrue("Método invocado com sucesso!", true);
40
41      }
42
43      public Object createMBean() {
44         return new MBeanStatusAnagrama();
45      }
46
47      protected void setUp() throws Exception {
48         jmxInit();
49      }
50
51      protected void tearDown() throws Exception {
52         jmxTerminate();
53      }
54
55      public static Test suite() {
56         TestSuite suite = new TestSuite(MBeanStatusAnagramaTest.class);
57         return suite;
58      }
```

```
59
60      private Object getAttribute(String attName) throws Exception {
61          return getMBeanServer().getAttribute(mbeanName, attName);
62      }
63
64      private void setAttribute(Attribute att) throws Exception {
65          getMBeanServer().setAttribute(mbeanName, att);
66      }
67
68      private Object invoke(String opName, Object params[],
69              String signature[]) throws Exception {
70          return getMBeanServer().invoke(mbeanName, opName, params,
71  signature);
72      }
73
74      private void jmxInit() throws Exception {
75          mbeanName = new ObjectName(":type=MBeanStatusAnagrama");
76          Object mbean = createMBean();
77          server = ManagementFactory.getPlatformMBeanServer();
78          server.registerMBean(mbean, mbeanName);
79      }
80
81      private void jmxTerminate() throws Exception {
82          server.unregisterMBean(mbeanName);
83      }
84
85      private MBeanServer getMBeanServer() {
86          return server;
87      }
88
89      private MBeanServer server;
90      private ObjectName mbeanName;
91
92  }
```

Figura 33: Códigos da classe de teste JUnit "MBeanStatusAnagramaTest.java" após alterações

Para a execução do teste JUnit no MBean, deve-se explorar a área do campo "*Project*", expandir o caminho "AnagramaGerenciavel" + "*Test Package*" + "*com.toy.anagrams.mbeans*" e selecionar a classe "MBeanStatusAnagramaTest.java", criada anteriormente. A seguir, deve-se selecionar a opção "*Run "MBeanStatusAnagramaTest.java"*", localizada em "*Run File*" no menu "*Run*" ou utilizar a tecla de atalho (Shift + F6). A figura 34 apresenta a tela do NetBeans, com o menu "*Run*" expandido e a opção "*Run "MBeanStatusAnagramaTest.java"*" selecionada.

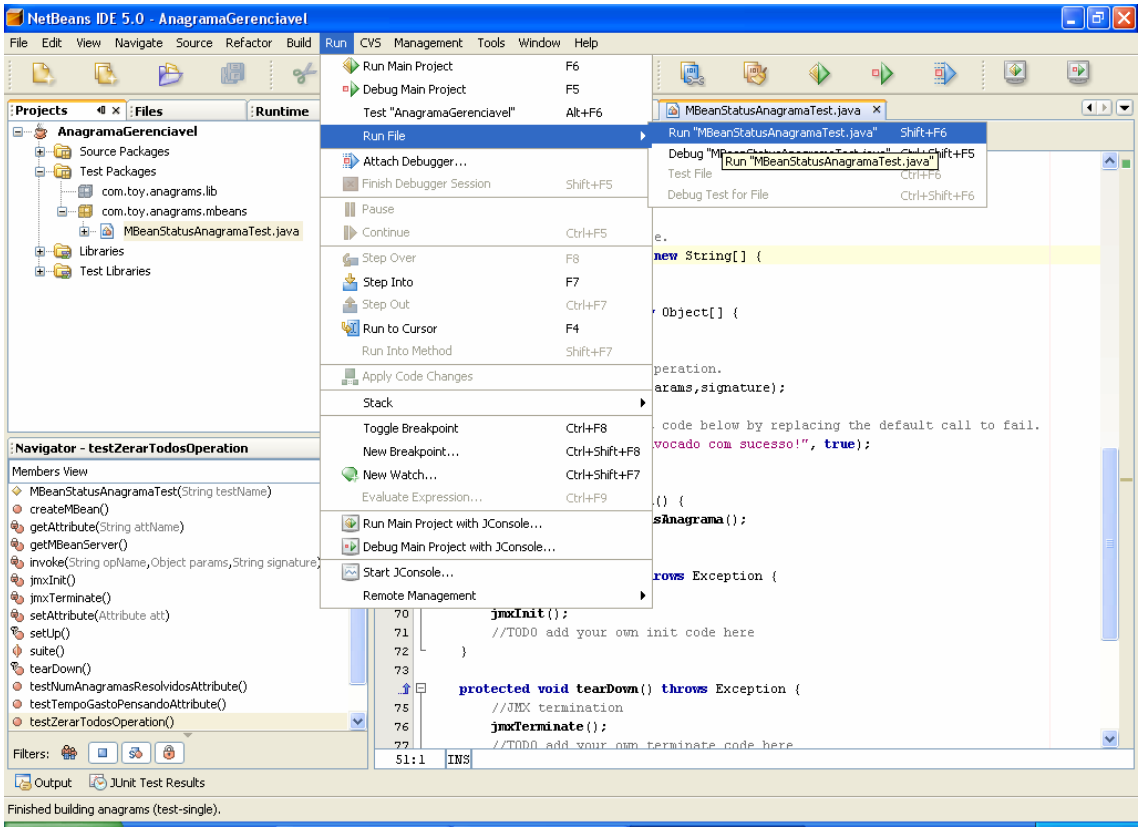

Figura 34: Tela de execução do NetBeans, com o menu "*Run*" expandido e a opção "*Run "MBeanStatusAnagramaTest.java"*" selecionada

Logo após, são realizados testes para verificar o funcionamento do MBean. Os resultados dos testes são apresentados na janela "*JUnit Test Results*", assim como apresenta a figura 35, que exibe a tela de execução do NetBeans, com os resultados no campo "*JUnit Test Results*". O resultado do teste deverá apresentar a informação "*passed*" em todos os itens, caso contrário deve-se revisar os passos descritos na criação do MBean "MBeanStatusAnagrama" para verificar erros na implementação.

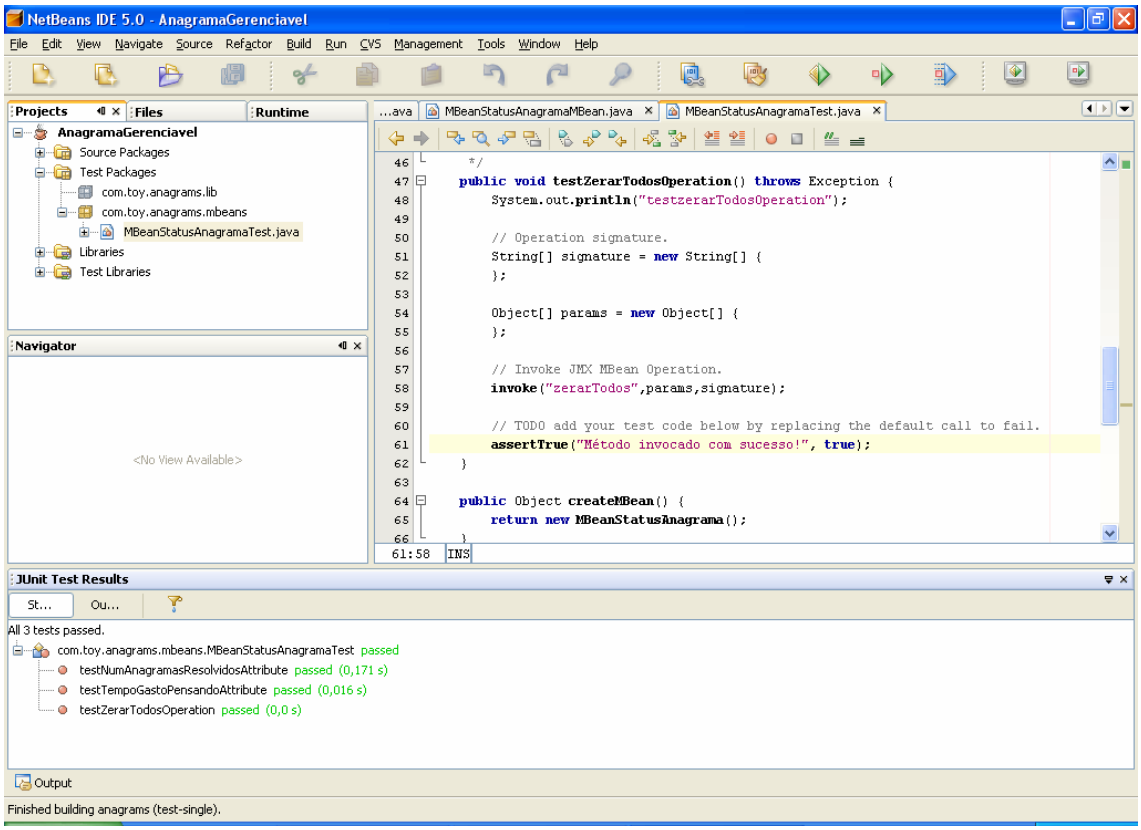

Figura 35: Tela de execução do NetBeans, com os resultados no campo "*JUnit Test Results*"

Após os passos de atualização e execução de um teste JUnit JMX, que permite a confirmação do funcionamento do MBean, deve-se criar um recipiente para guardar e invocar os recursos oferecidos pelos MBeans. Este recipiente será o Agente JMX, que será abordado nas próximas seções.

3.7 Gerando um Agente JMX

Nesta seção será apresentado o desenvolvimento de um Agente JMX, que será responsável por controlar diretamente recursos e torná-los acessíveis por agentes de gerenciamento remoto. Por padrão um Agente JMX poderá ser uma classe executável, que contenha um método principal em que abrigará a classe gerenciável, ou uma classe não executável, que será utilizada apenas para atribuir gerenciamento a classes pré-existentes. No teste será abordado um agente não executável, uma vez que a classe principal "Anagrams.java" já

existe. Assim, serão atribuídos, apenas, meios para que ela seja gerenciável. Um Agente JMX pode oferecer:

- Método para criar e registrar Mbeans.
- Acesso ao Servidor de MBeans.
- Simplicidade ao projeto modelado, pois permite fácil acesso a um agente inicializado. Um agente inicializado é um agente que possui todos os MBeans registrados.

Para iniciar a criação do Agente JMX, deve-se escolher a opção "*New File*" no menu "*File*" ou utilizar a tecla de atalho (Ctrl + N). A figura 36 apresenta a tela de execução do NetBeans, com o menu "*File*" expandido.

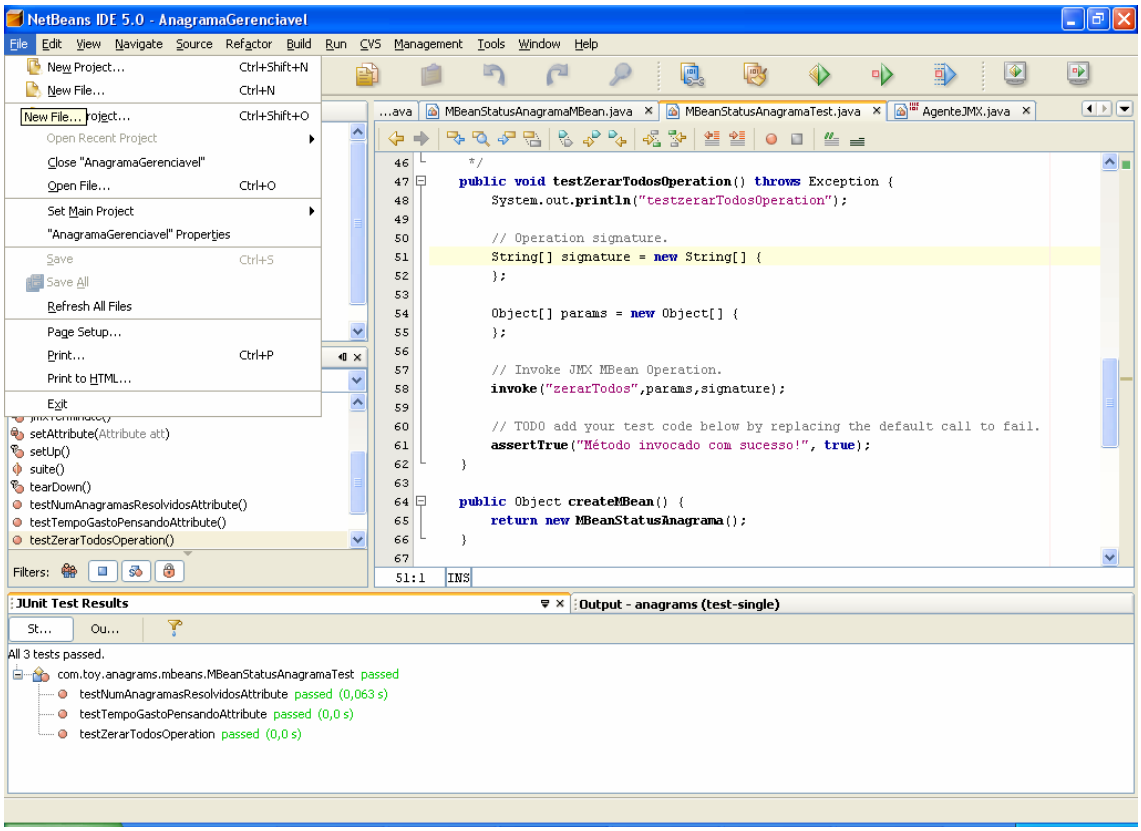

Figura 36: Tela de execução do NetBeans, com o menu "*File*" expandido

Para a criação do Agente JMX é necessário selecionar o diretório "*Management*" no campo "*Categories*", e no campo "*File Types*" deve-se selecionar "*JMX Agent*", como apresenta a figura 37, que exibe a tela *Choose File Type* (Seleção de Tipos de Arquivo)

com o *JMX Agent* selecionado. Por fim será necessário selecionar o botão "*Next*" para prosseguir com a criação do MBean.

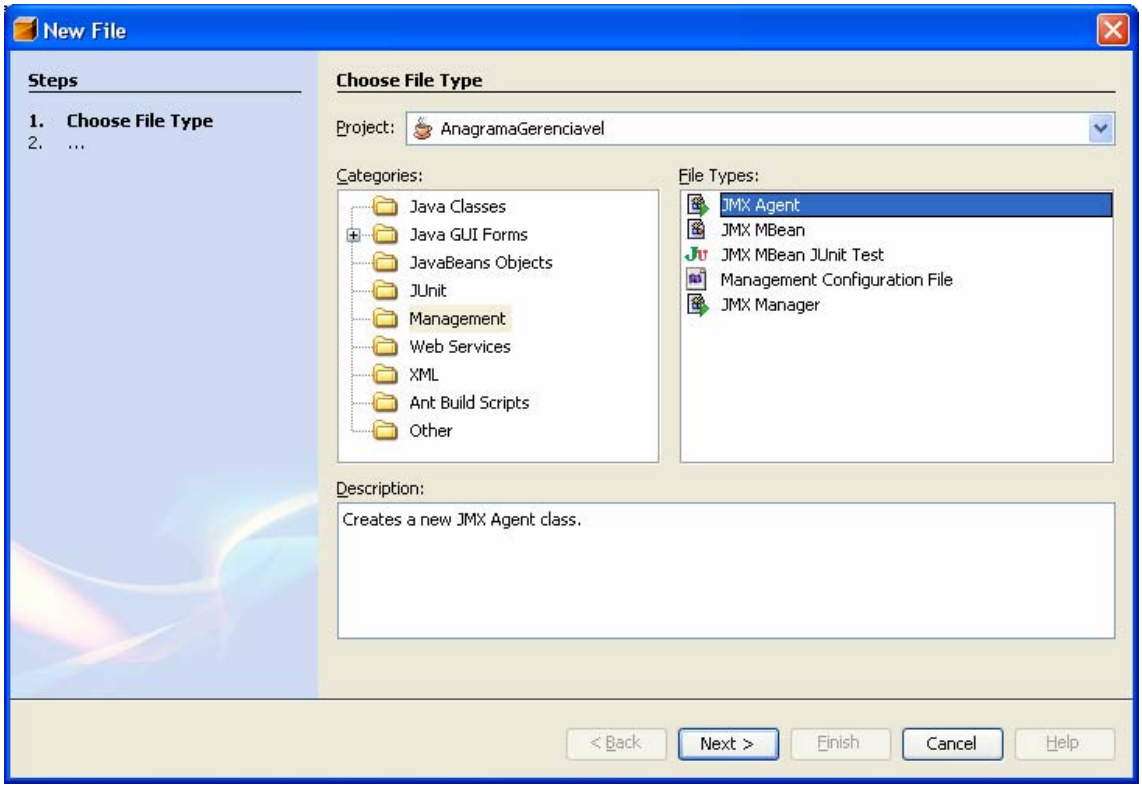

Figura 37: Tela de Seleção de Tipo de Arquivo com o *JMX Agent* selecionado

Para determinar o nome da classe, o local e em qual pacote será armazenado, basta alterar respectivamente as informações "*Class Name*", "*Location*" e "*Package*" e selecionar o botão "*Next*". Para o exemplo deve-se colocar as seguintes informações:

- *Class Name*: AgenteJMX
- *Location*: Source Packages
- *Package*: com.toy.anagrams.agent

Deve-se desmarcar a opção "*Create Main Method*" (Criar Método Principal), pois no exemplo será realizada a instrumentação de uma aplicação que já possui um método principal. A figura 38 apresenta a tela "*Name and Location*" (Nome e Localização) com os campos preenchidos com os dados listados anteriormente e a opção "*Create Main Method*" desativada. Por fim será necessário selecionar o botão "*Finish*" para criar o Agente JMX.

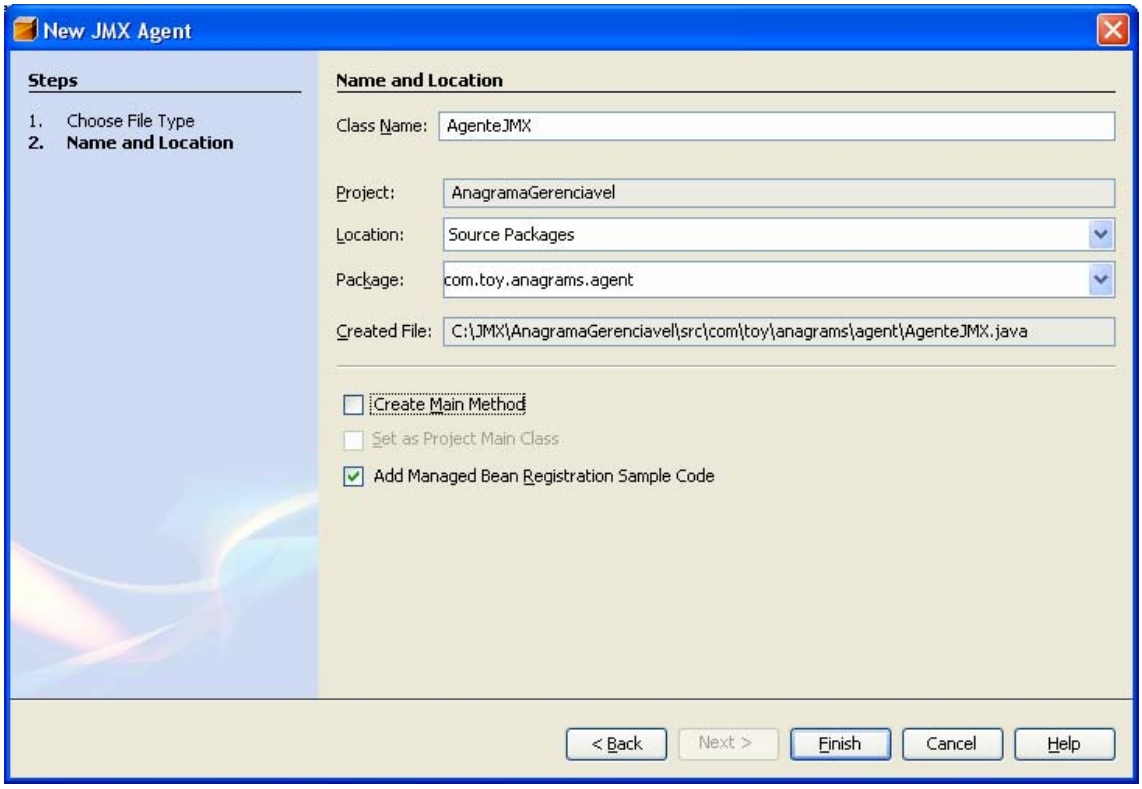

Figura 38: Tela de adição de nome e localização do Agente JMX, com os campos preenchidos e a opção "*Create Main Method*" desativada

A classe AgenteJMX será criada no projeto e aberta no painel do editor de código. O código fonte da classe “AgenteJMX.java” deverá ser semelhante ao que é apresentado na figura 39, porém sem os comentários.

```
1    package com.toy.anagrams.agent;
2
3    import javax.management.ObjectName;
4    import javax.management.MBeanServer;
5    import java.lang.management.ManagementFactory;
6
7    public class AgenteJMX {
8
9       public void init() throws Exception {
10      }
11
12      public synchronized static AgenteJMX getDefault() throws Exception {
13         if(singleton == null) {
14            singleton = new AgenteJMX();
15            singleton.init();
16         }
17         return singleton;
18      }
19
```

```
20      public MBeanServer getMBeanServer() {
21          return mbs;
22      }
23
24      private final MBeanServer mbs =
25  ManagementFactory.getPlatformMBeanServer();
26
27      private static AgenteJMX singleton;
28  }
```

Figura 39: Classe gerada na criação do Agente JMX “AgenteJMX.java”

Dentre os comentários gerados juntos com a classe do Agente JMX é importante manter o comentário que há dentro do método init(), pois neste comentário é exemplificado a forma de registro de um MBean manualmente. A figura 40 apresenta o comentário retirado do método init() da classe “AgenteJMX.java”.

```
1   /* *** SAMPLE REGISTRATION EXAMPLE *** */
2         /*
3         // Instantiate CounterMonitor MBean
4         javax.management.monitor.CounterMonitor mbean =
5            new javax.management.monitor.CounterMonitor();
6         ObjectName mbeanName = new ObjectName (":type=CounterMonitor");
7         //Register the CounterMonitor MBean
8         getMBeanServer().registerMBean(mbean, mbeanName);
9          */
```

Figura 40: Comentário retirado do método init() da classe “AgenteJMX.java”

Após a criação do Agente JMX, faz-se necessária inclusão de pelo menos um MBeans ao agente, uma vez que, o principal propósito da criação de Agentes JMX é o de propiciar acesso a recursos instrumentados pelo MBeans para agentes de gerenciamento remoto; na próxima seção serão abordados os passos necessários para a inclusão de MBeans em um Agente JMX.

3.8 Adicionando o MBean no Agente

Nesta seção será adicionado um MBean “MBeanStatusAnagrama” em um Agente JMX “AgenteJMX”. Adicionar um MBean a um agente envolve três etapas: Instanciação do MBean, Chamada ao MBean e registro do MBean no Servidor de MBeans.

Para aperfeiçoar e padronizar o desenvolvimentos de aplicações utilizando a tecnologia JMX, a *Sun Microsystem* confeccionou um documento chamado *Best Practices*,

neste documento constam as melhores práticas de desenvolvimento de aplicações utilizando a tecnologia JMX, para prosseguir com a criação do exemplo é importante observar alguns itens referentes a nomeação de MBeans, deve-se verificar que:

- Ao nomear um MBean, deve-se utilizar o tipo chave. O valor desta chave deve ser a classe do MBean (no exemplo deve-se utilizar MBeanStatusAnagrama);
- No exemplo de um único MBean (um MBean que tenha uma única instância dentro de sua aplicação), ele possuirá a chave única que é suficiente para a nomeação;
- Não se deve colocar nomes de domínios grandes demais. Poderá ser feito reutilização do nome de domínio padrões. Não se deve especificar um domínio antes do ObjectName: é uma referência implícita para nomeação de domínio padrão.

A utilização destas *Best Practices* (Melhores Práticas) traz a formalidade necessária para a implementação de MBeans. O ObjectName que está sendo criado para o exemplo é `:type=MBeanStatusAnagrama`.

Para instanciar e Registrar o MBean no Agente JMX deve-se explorar a área do campo "*Project*", expandir o caminho "AnagramaGerenciavel" + "*Source Package*" + "*com.toy.anagrams.agent*" e selecionar a classe "AgenteJMX.java. A seguir, deve-se selecionar a opção "*Instantiate and Register MBean"*, no menu "*Management*".

Aparecerá a tela *Instantiate and Register MBean* (Instanciar e Registrar um MBean). Para Registrar um MBean existente deve-se selecionar a opção "Register Existing MBean" e selecionar o botão "*Browse...*". A figura 41 apresenta a tela *Instantiate and Register MBean* (Instanciar e Registrar um MBean).

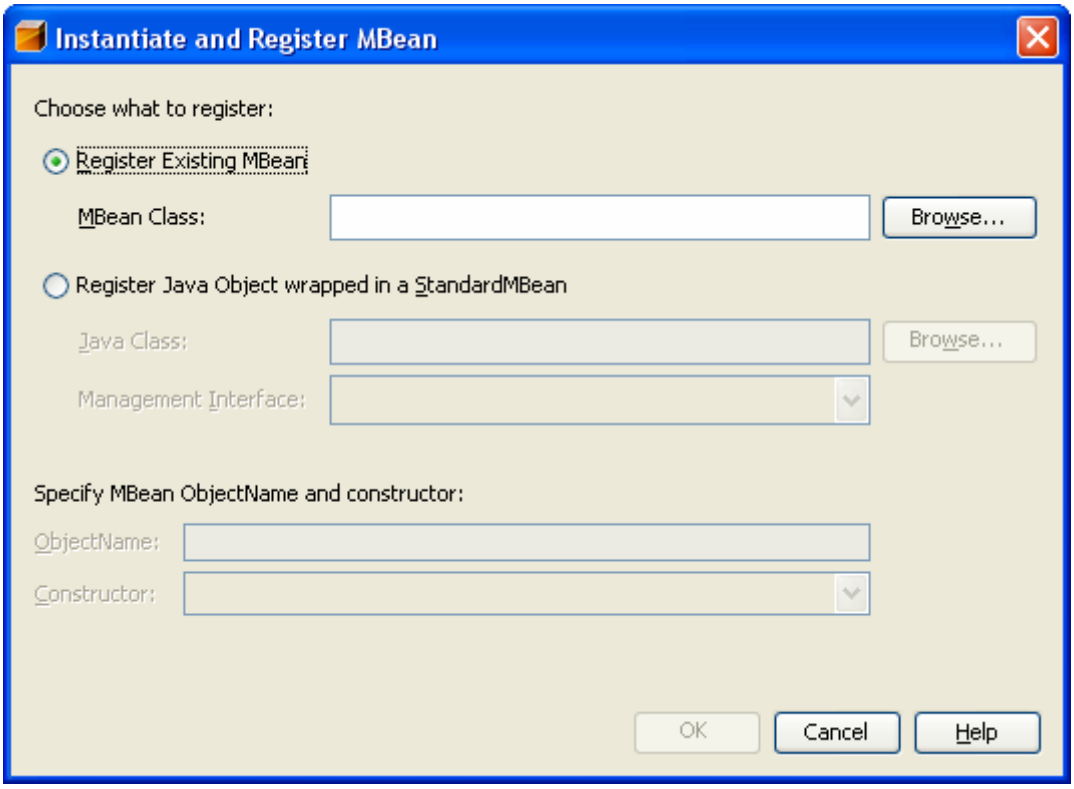

Figura 41: Tela de Instanciação e Registro de um MBean

Após selecionar a opção “Browse...” aparecerá a tela *Select Class* (Selecione a Classe), com a árvore do projeto atual indicada, deve-se expandir o caminho “*Sources*” + “*Source Packages*” + “*com.toy.anagrams.mbeans*” e selecionar o MBean “MBeanStatusAnagrama”. E em seguida deve-se selecionar o botão “*OK*”. A figura 42 apresenta a tela *Select Class* (Selecione a Classe), com o “MBeanStatusAnagrama” selecionado.

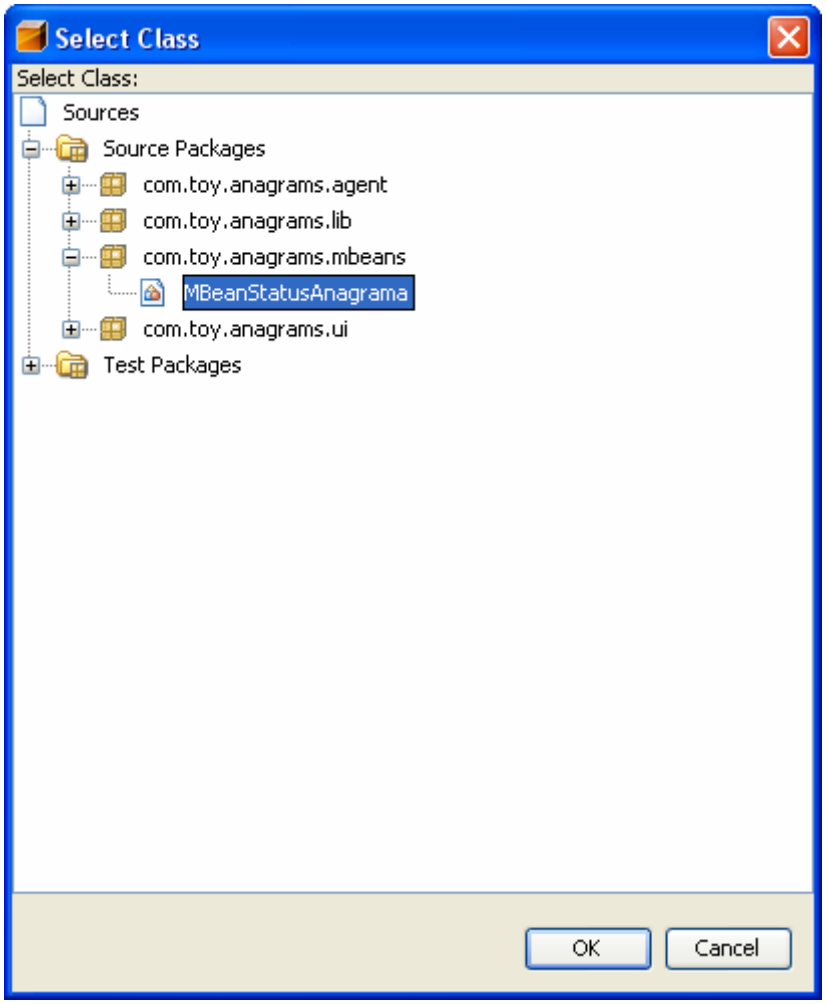

Figura 42: Tela de seleção da classe, com o "MBeanStatusAnagrama" selecionado

Após selecionar o botão "*OK*" a caixa de diálogo *Instantiate and Register MBean* (Instanciar e Registrar um MBean) terá os dados atualizados com o MBean selecionado. Os campos *ObjectName* e *Construtor*, respectivamente, serão preenchidos automaticamente com os dados "mbeans.anagrams.toy.com:type=MBeanStatusAnagrama" e "MBeanStatusAnagrama()".

Deve-se manter os dados preenchidos automaticamente, assim como apresenta a figura 43, com os campos *MBean Class*, *ObjectName* e *Construtor* preenchidos, e selecionar o botão "OK".

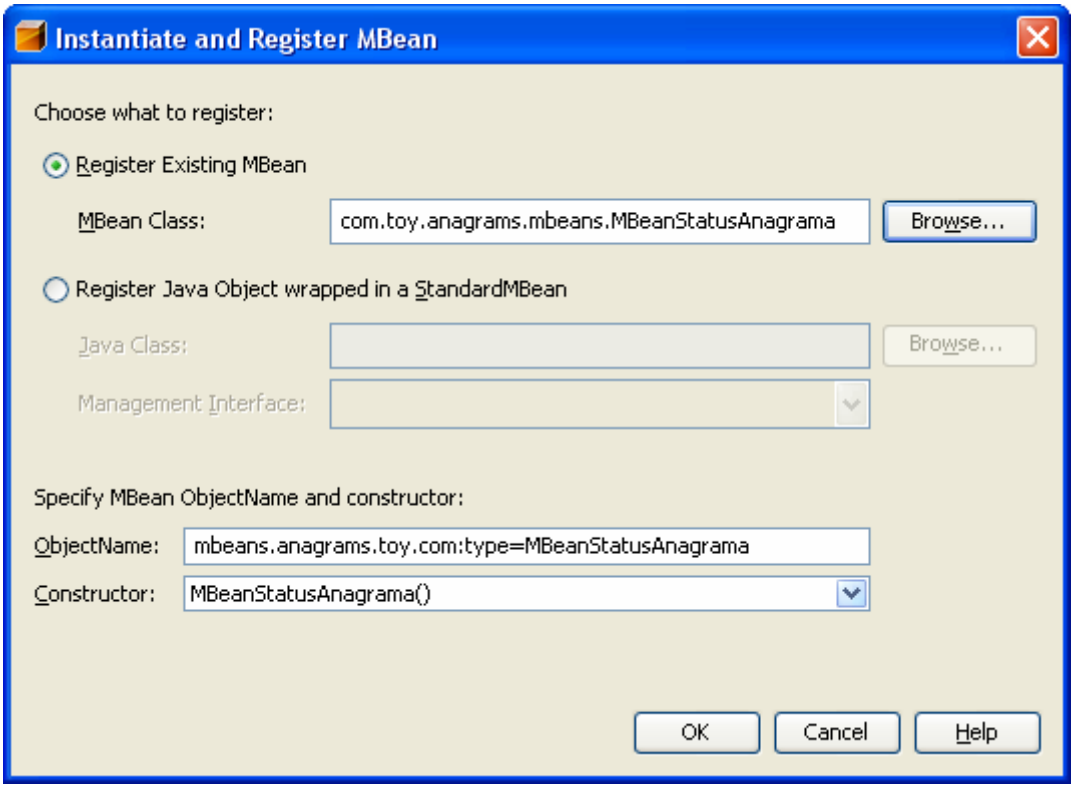

Figura 43: Tela de Instanciação e Registro de um MBean, com os campos preenchidos

Após selecionar o botão "*OK*" o método init() do AgenteJMX será atualizado com o código de registro do MBean. A figura 44 apresenta todo o código da classe do AgenteJMX após a instanciação e registro do MBean.

```
1   package com.toy.anagrams.agent;
2
3   import javax.management.ObjectName;
4   import javax.management.MBeanServer;
5   import java.lang.management.ManagementFactory;
6
7   public class AgenteJMX {
8
9      public void init() throws Exception {
10        /* *** SAMPLE REGISTRATION EXAMPLE *** */
11        /*
12              // Instantiate CounterMonitor MBean
13              javax.management.monitor.CounterMonitor mbean =
14                 new javax.management.monitor.CounterMonitor();
15              ObjectName mbeanName = new ObjectName
16  (":type=CounterMonitor");
17              //Register the CounterMonitor MBean
18              getMBeanServer().registerMBean(mbean, mbeanName);
19               */
20        getMBeanServer().registerMBean(
21           new com.toy.anagrams.mbeans.MBeanStatusAnagrama(),
22           new
23  ObjectName("mbeans.anagrams.toy.com:type=MBeanStatusAnagrama"));
```

```
24      }
25
26      public synchronized static AgenteJMX getDefault() throws Exception {
27         if(singleton == null) {
28            singleton = new AgenteJMX();
29            singleton.init();
30         }
31         return singleton;
32      }
33
34      public MBeanServer getMBeanServer() {
35         return mbs;
36      }
37
38      private final MBeanServer mbs =
39   ManagementFactory.getPlatformMBeanServer();
40
41      private static AgenteJMX singleton;
42   }
```

Figura 44: Classe do Agente JMX "AgenteJMX.java" após a instanciação e registro do MBean

Para a realização do exemplo faz-se necessária a atribuição de acesso a interface do MBean "MBeanStatusAnagrama" para a aplicação. Não é obrigatória a realização desta atribuição na criação de aplicações gerenciáveis, porém pode ser útil quando a aplicação necessitar enviar dados para um MBean.

Os métodos startThinking e stopThinking são usados pela aplicação para notificar o MBean que alguns eventos ocorreram. Por sua vez, o MBean atualiza seu estado. Para tornar o MBean "MBeanStatusAnagrama" acessível à classe "Anagrams.java", é necessário modificar o método init() e adicionar no MBean uma permissão de acesso para o agente.

Na classe do Agente JMX deve-se adicionar o trecho de código apresentado na figura 45 na linha abaixo do último *import* da classe. O código apresentado na figura 45 importa a classe MBeanStatusAnagrama.

```
1   import com.toy.anagrams.mbeans.MBeanStatusAnagrama;
2   // Importa a classe MBean
```

Figura 45: Trecho de código que importa a classe MBeanStatusAnagrama

Deve-se também adicionar o trecho de código apresentado na figura 46 na linha abaixo da instanciação da classe. O código apresentado na figura 46 permite acesso à classe MBeanStatusAnagrama.

```
1     //Acessa o MBean
2     private MBeanStatusAnagrama mbean;
3
4     public MBeanStatusAnagrama getMBeanStatusAnagrama(){
5        return mbean;
6     }
```

Figura 46: Trecho de código que permite acesso à classe MBeanStatusAnagrama

Deve-se substituir todo o código do método Init() da classe MBeanStatusAnagrama pelo apresentado na figura 47, que instancia um MBean e atualiza a declaração do registro.

```
 1        mbean = new MBeanStatusAnagrama();
 2        // Atualiza o método Init()
 3        getMBeanServer().registerMBean(mbean,
 4           new
 5  ObjectName("mbeans.anagrams.toy.com:type=MBeanStatusAnagrama"));
 6        /* *** SAMPLE REGISTRATION EXAMPLE *** */
 7        /*
 8              // Instantiate CounterMonitor MBean
 9              javax.management.monitor.CounterMonitor mbean =
10                 new javax.management.monitor.CounterMonitor();
11              ObjectName mbeanName = new ObjectName
12  (":type=CounterMonitor");
13              //Register the CounterMonitor MBean
14              getMBeanServer().registerMBean(mbean, mbeanName);
15        */
```

Figura 47: Trecho de código do método Init() que instancia um MBean e atualiza a declaração do registro

Após a alteração do código o Agente JMX estará instrumentado, pois nesta etapa foi realizada a instanciação e o registro do MBean, bem com a liberação de acesso ao MBean. A figura 48 apresenta todo o código da classe do AgenteJMX após alterações.

```
1     package com.toy.anagrams.agent;
2
3     import javax.management.ObjectName;
4     import javax.management.MBeanServer;
5     import java.lang.management.ManagementFactory;
6     import com.toy.anagrams.mbeans.MBeanStatusAnagrama;
7
8     public class AgenteJMX {
9        private MBeanStatusAnagrama mbean;
10
11       public MBeanStatusAnagrama getMBeanStatusAnagrama(){
12          return mbean;
13       }
```

```
14
15      public void init() throws Exception {
16          mbean = new MBeanStatusAnagrama();
17          getMBeanServer().registerMBean(mbean,
18              new
19  ObjectName("mbeans.anagrams.toy.com:type=MBeanStatusAnagrama"));
20      }
21
22      public synchronized static AgenteJMX getDefault() throws Exception {
23          if(singleton == null) {
24              singleton = new AgenteJMX();
25              singleton.init();
26          }
27          return singleton;
28      }
29
30      public MBeanServer getMBeanServer() {
31          return mbs;
32      }
33
34      private final MBeanServer mbs =
35  ManagementFactory.getPlatformMBeanServer();
36
37      private static AgenteJMX singleton;
38  }
```

Figura 48: Classe do Agente JMX “AgenteJMX.java” após alterações

Falta apenas criar a conexão do Jogo de Anagramas com o gerenciamento JMX para possibilitar a realização da monitoração utilizando o JConsole.

3.9 Conectando a Aplicação Gerenciável ao Gerenciamento JMX

Nesta seção será abordada a forma de acesso ao MBean e a forma de implementação do gerenciamento. Para isso, deve-se explorar a área do campo “*Project*”, expandir o caminho “AnagramaGerenciavel” + “*Source Packages*” + “*com.toy.anagrams.ui*” e clicar duas vezes sobre a classe “Anagrams.java”. A seguir, a classe será visualizada no projeto da IDE, assim como apresenta a figura 49, com a tela de execução do NetBeans e com a classe “Anagrams.java” aberta.

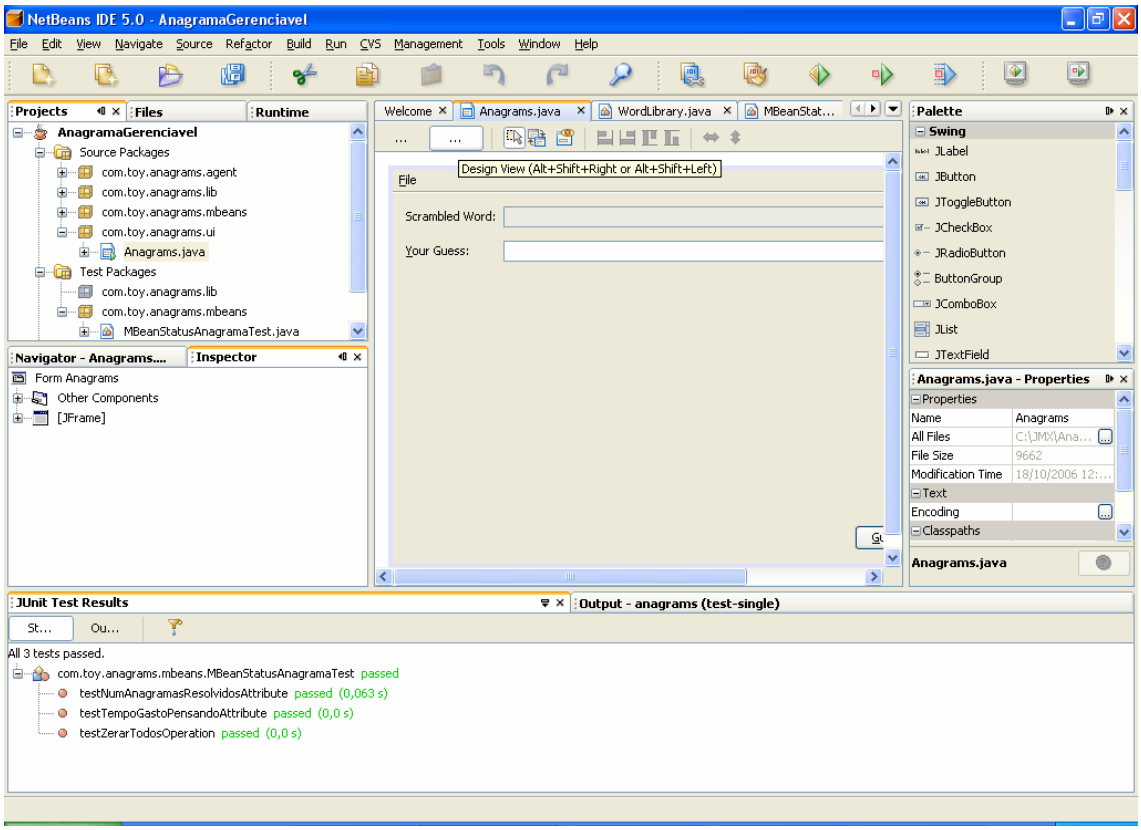

Figura 49: Tela de execução do NetBeans, com a classe “Anagrams.java” aberta

Será necessário abrir a classe no editor de código para alterar a classe “Anagrams.java”. Para isso deve-se selecionar a aba “*Source*” no topo do Editor de Códigos ou utilizar a tecla de atalho (Alt + Shift + Right). É necessário adicionar no construtor da classe Anagrams uma chamada ao método do initManagement, para realizar esta operação basta copiar o seguinte trecho de código `initManagement();` para o final do construtor da classe Anagrams.

O método initManagement() da classe **“**Anagrams.java**”** deverá ser implementado para: acessar o Agente JMX, posteriormente, registrar o MBean, em seguida, acessar o MBean e por fim, chamar a função `startThinking()` implementada na classe “`MBeanStatusAnagrama.java`”. Isso para que seja calculado o tempo absoluto do início da execução aplicação até a primeira resposta correta do anagrama proposto ao jogador, permitindo assim o cálculo completo do desempenho do jogador em relação ao tempo gasto nas resoluções do jogo.

Para atribuir o funcionamento necessário ao método initManagement deve-se adicionar o código apresentado na figura 50 na classe “Anagrams.java”. Este código criará um Agente JMX e executará uma função do MBean.

```
1    private void initManagement() throws Exception {
2          try{
3             //Criar o Agente
4             com.toy.anagrams.agent.AgenteJMX agente =
5                   com.toy.anagrams.agent.AgenteJMX.getDefault();
6
7      //Quando o jogo dos Anagrams é exibido pela primeira vez,
8      //deve-se notificar o MBean que um novo anagrama foi proposto.
9             agente.getMBeanStatusAnagrama().startThinking();
10         } catch(Exception e ){e.printStackTrace();}
11   }
```

Figura 50: Método initManagement() da classe "Anagrams.java"

Para obter dados da experiência do usuário na resolução do anagrama será necessário alterar o código de dois métodos: o nextTrialActionPerformed e guessedWordActionPerformed. Para cada vez que um novo anagrama é proposto ao usuário, deve-se calcular o tempo de início, isto será realizado no método nextTrialActionPerformed; e cada vez que um anagrama for suposto corretamente, é necessário notificar ao MBean o tempo gasto pensando na suposição do anagrama. Isto será realizado no método guessedWordActionPerformed.

Para alterar o método nextTrialActionPerformed deve-se substituí-lo com o código apresentado na figura 51 e para alterar o método guessedWordActionPerformed deve-se substituí-lo com o código apresentado na figura 52.

```
1    private void nextTrialActionPerformed(java.awt.event.ActionEvent evt) {
2          wordIdx = (wordIdx + 1) % WordLibrary.getSize();
3
4          feedbackLabel.setText(" ");
5          scrambledWord.setText(WordLibrary.getScrambledWord(wordIdx));
6          guessedWord.setText("");
7          getRootPane().setDefaultButton(guessButton);
8
9          guessedWord.requestFocusInWindow();
10
11         // Notify MBean to store start time.
12         try {
13            com.toy.anagrams.agent.AgenteJMX.getDefault().
14                  getMBeanStatusAnagrama().startThinking();
15         }catch(Exception e) {e.printStackTrace();}
16   }
```

Figura 51: Método initManagement() da classe "Anagrams.java"

```
1      private void guessedWordActionPerformed(java.awt.event.ActionEvent evt) {
```

```
2           if (WordLibrary.isCorrect(wordIdx, guessedWord.getText())){
3
4               feedbackLabel.setText("Correct! Try a new word!");
5               getRootPane().setDefaultButton(nextTrial);
6
7               //Notify MBean that the user has resolved Anagram.
8               try{
9                   com.toy.anagrams.agent.AgenteJMX.getDefault().
10                      getMBeanStatusAnagrama().stopThinking();
11              }catch(Exception e) {e.printStackTrace();}
12
13          } else {
14              feedbackLabel.setText("Incorrect! Try again!");
15              guessedWord.setText("");
16          }
17
18          guessedWord.requestFocusInWindow();
19      }
```

Figura 52: Método initManagement() da classe "Anagrams.java"

Ao finalizar a substituição dos métodos será concluída a ligação da camada JMX com a camada de aplicação, após esta etapa a aplicação estará preparada para ser monitorada por uma aplicação de gerenciamento.

3.10 Gerenciamento da Aplicação via JConsole

O JConsole é uma ferramenta de gerenciamento e monitoramento de aplicações e dispositivos em tempo de execução, desenvolvida pela *Sun Microsystem* e oferecida gratuitamente no *kit* J2SE. Através do JConsole é possível gerenciar recursos instrumentados através de MBeans contidos na arquitetura da tecnologia JMX e a Máquina Virtual Java (JVM).

Dentro da arquitetura do JMX, o JConsole representa a aplicação de gerência no Nível de Gerente, que faz conexão com o Agente JMX através de adaptadores e conexões, permitindo ligar a aplicação de gerência com os recursos e os serviços instrumentados.

O uso do JConsole não é obrigatório para o gerenciamento utilizando a API JMX, podem-se desenvolver ferramentas personalizadas que substituem o seu uso ou, ainda, podem-se utilizar outras ferramentas já implementadas. Porém, dependendo da aplicação que será gerenciada, não é vantajosa a implementação de uma aplicação de gerência e monitoração, pois o desenvolvimento de uma aplicação com características semelhantes a do JConsole é complexa. As vantagens de se utilizar o JConsole é que este é distribuído

com o pacote Java gratuitamente, é de fácil utilização, possui interface gráfica bem definida e é estável.

O JConsole pode-se conectar a um Agente JMX em execução de 3 (três) maneiras distintas:

- Localmente: conecta-se a processo de uma aplicação Java em execução no sistema local, o qual será executado com o mesmo nome do usuário. O JConsole conecta-se na plataforma do servidor MBean usando um conector do RMI com uma autenticação que usa a permissão de acesso do sistema de arquivo.
- Remotamente: conecta-se a um agente de JMX que usa um conector do RMI. Deve-se passar o nome do *host* ou IP do Agente JMX, a porta que está esperando a conexão, o usuário e a senha com permissão de acesso.
- Modo Avançado: conecta-se a um agente de JMX com o URL especificado. Normalmente se conecta a um agente de JMX que possui um conector implementado de forma personalizada, com exceção do conector do RMI que é padrão.

O JConsole possui as seguintes funcionalidades implementadas:

- *Memory* (Memória): a qual exibe informações sobre o uso de memória pelas aplicações Java.
- *Threads* (Processos em Execução): a qual exibe informações sobre os Processos de aplicações Java.
- *Classes*: exibe informações sobre o carregamento das classes Java.
- *MBeans*: exibe informações sobre dos MBeans dispostos ao JConsole.
- *VM* (Máquina Virtual): exibe informações sobre a Máquina Virtual Java (JVM), trazendo informações de recursos de hardware importantes.

A seguir são apresentados alguns exemplos de uso de gerenciamento JMX utilizando o JConsole baseado na implementação de gerenciamento do Jogo de Anagramas.

3.10.1 Monitoração da Aplicação Gerenciável

Após conectar a aplicação gerenciável, deve-se compilar e executar o projeto "AnagramaGerenciavel" para compilação das classes geradas em todas as seções anteriores e a realização do gerenciamento do Jogo de Anagramas utilizando o JConsole.

Para a compilação e execução do projeto "AnagramaGerenciavel" deve-se selecionar a opção "*Run Main Project with JConsole...*", no menu "*Run*", assim como apresenta a figura 53, com a tela de execução do NetBeans e com o menu "*Run*" expandido.

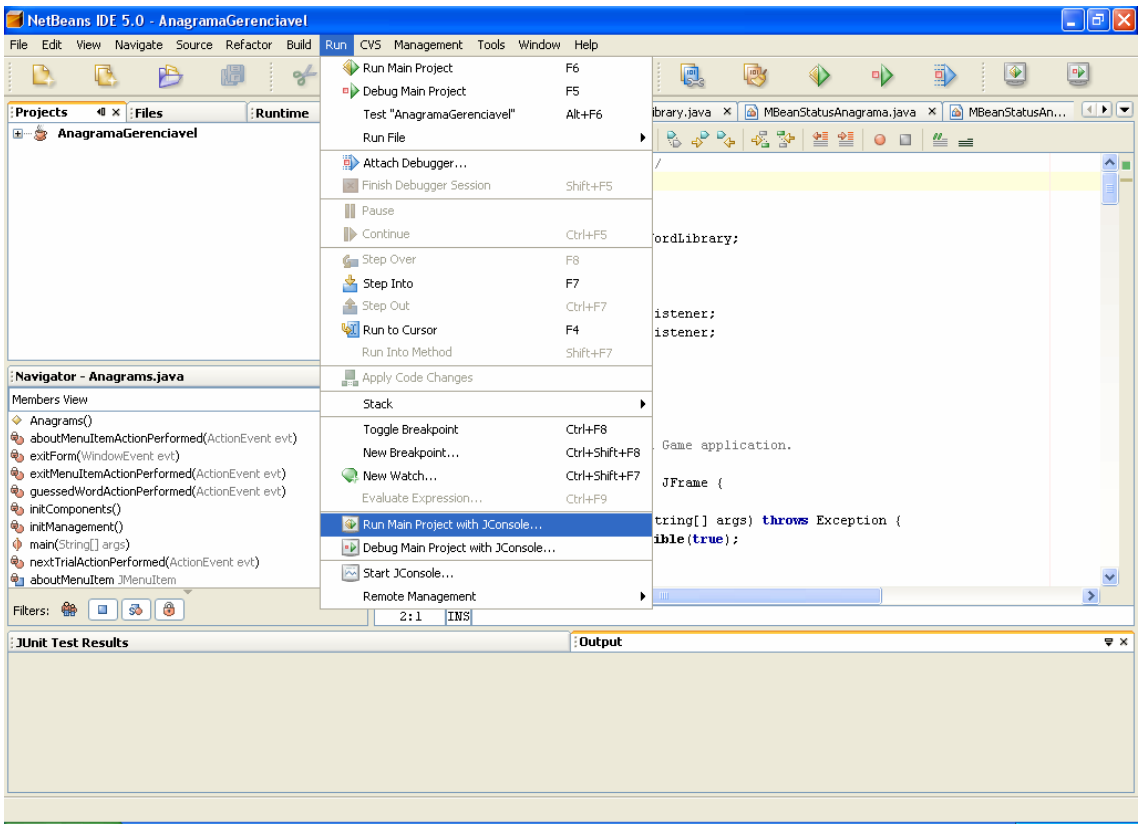

Figura 53: Tela de execução do NetBeans, com o menu "*Run*" expandido

O projeto será compilado e executado com o JConsole. Depois, deverão aparecer duas novas telas, a tela do Jogo de Anagramas (*Anagrams*) e a tela da Aplicação de Gerenciamento (JConsole). A figura 54 apresenta a tela do Jogo de Anagramas (*Anagrams*) e a figura 55 apresenta a tela da Aplicação de Gerenciamento (JConsole).

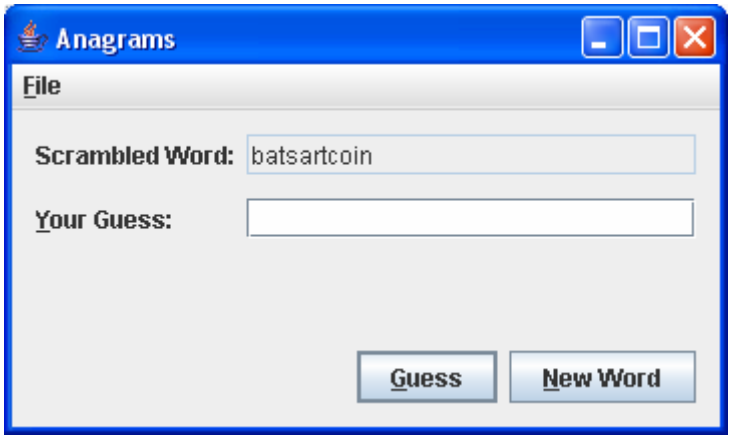

Figura 54: Tela do Jogo de Anagramas (*Anagrams*)

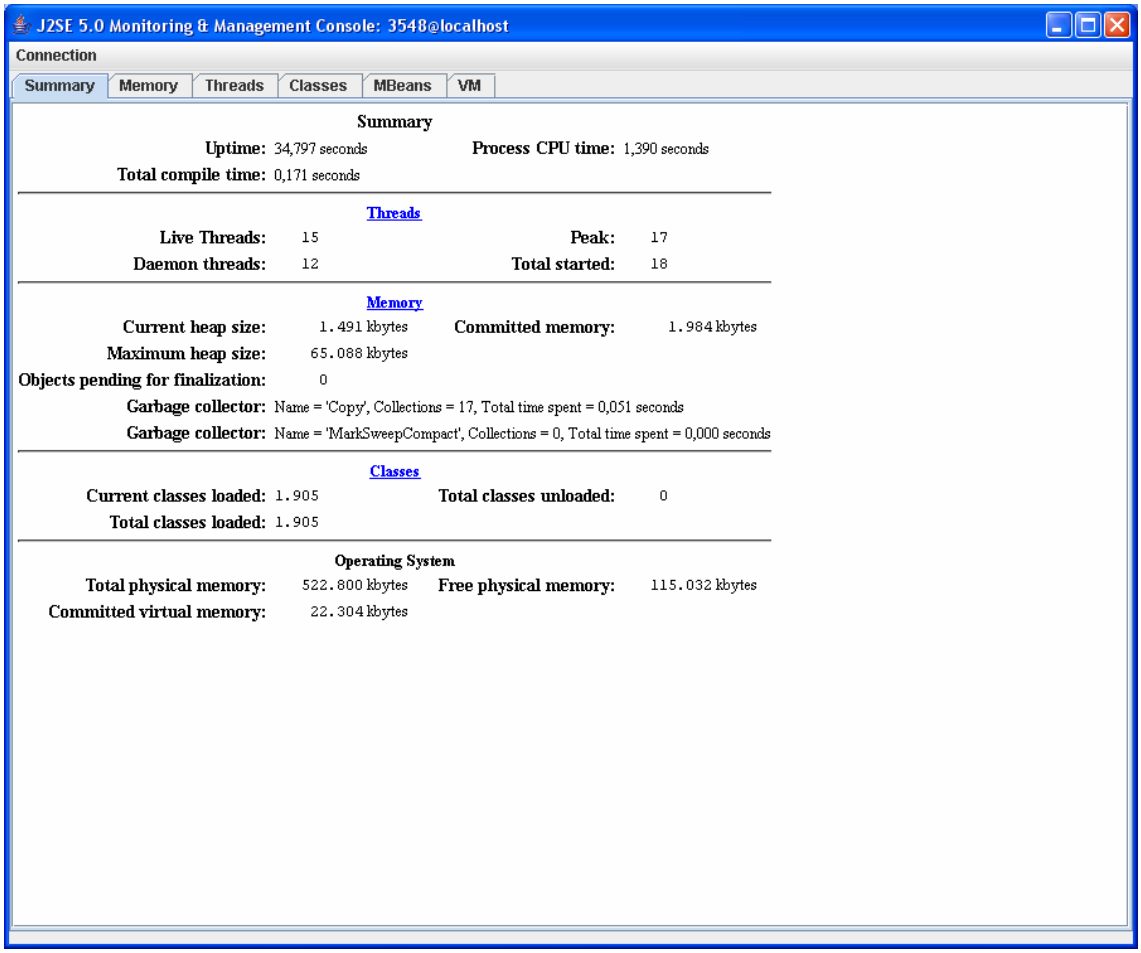

Figura 55: Tela da Aplicação de Gerenciamento (JConsole)

Para a realização do exemplo proposto deve-se configurar primeiramente a aplicação de gerenciamento e, somente após a configuração do ambiente da aplicação de gerenciamento, deve-se começar a jogar o Jogo de Anagramas.

Para configurar o ambiente de gerenciamento e monitorar o desempenho do jogador, visualizando as informações definidas na instrumentação do Jogo na criação do MBean “MBeanStatusAnagrama”, deve-se selecionar a aba “MBeans” e no campo MBeans expandir o caminho “*Tree*” + “*mbeans.anagrams.toy.com*” + “MBeanStatusAnagrama”.

A figura 56 apresenta a tela do JConsole na aba “MBeans” com o “MBeanStatusAnagrama” selecionado.

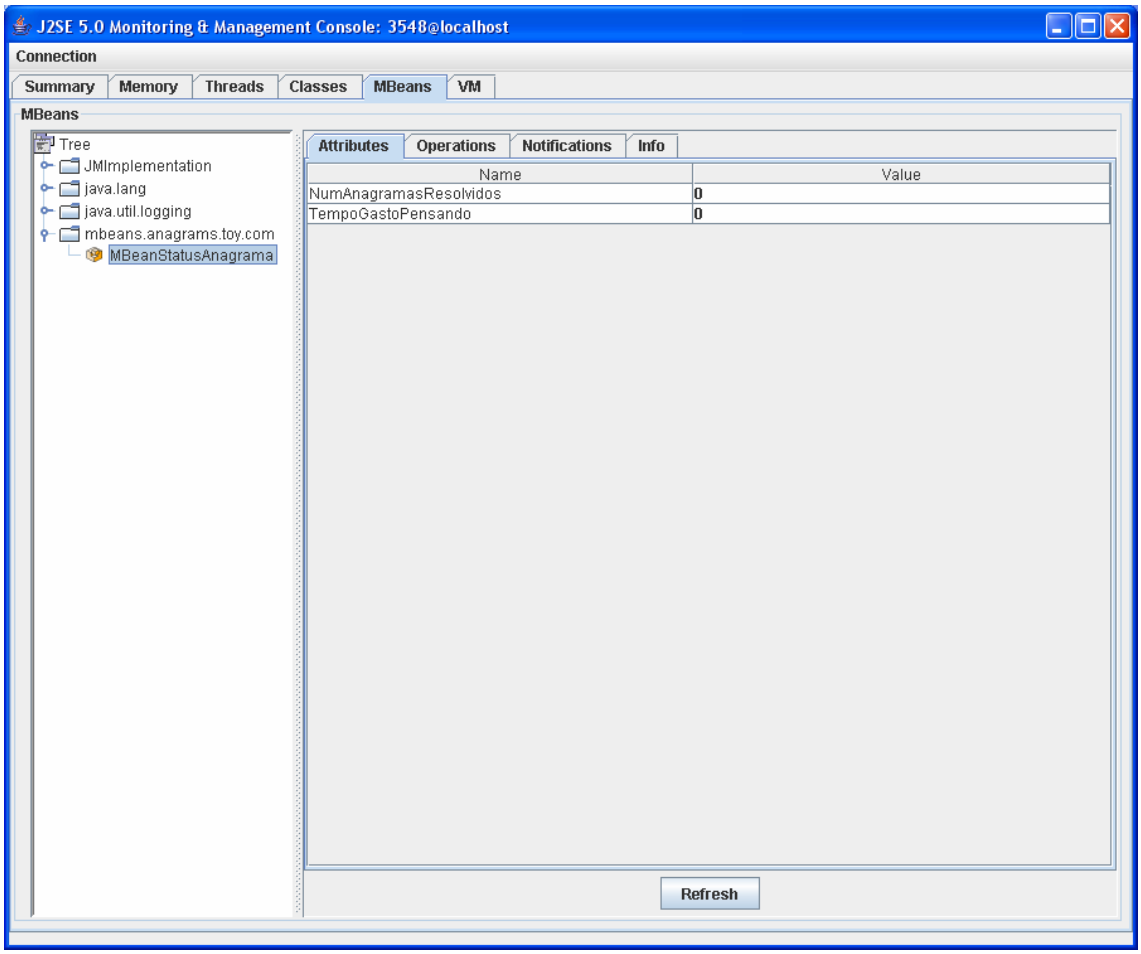

Figura 56: Tela do JConsole na aba “MBeans” com o “MBeanStatusAnagrama” selecionado

A seguir, deve-se selecionar a aba “*Notifications*” (Notificações) e selecionar o botão “*Subscribe*” (Subscreva), para que o JConsole receba e guarde uma notificação nova cada vez que um anagrama for resolvido. Logo após deve-se selecionar a aba “*Attributes*” e clicar duas vezes nos valores da coluna “*value*” com os nomes de atributos NumAnagramasResolvidos e TempoGastoPensando, para exibir os gráficos da análise. Em seguida, pode-se iniciar o jogo.

A cada resposta correta no Jogo de Anagramas, o gerenciamento JMX irá contar o número de Anagramas resolvidos através do atributo NumAnagramasResolvidos e calculará o tempo gasto pelo jogador para responder corretamente o anagrama proposto através do atributo TempoGastoPensando. Ambos os atributos (NumAnagramasResolvidos e TempoGastoPensando) foram declarados no “MBeanStatusAnagrama”.

A figura 57 apresenta a tela do JConsole na aba “MBeans” com o “MBeanStatusAnagrama” selecionado e com os gráficos dos atributos dispostos.

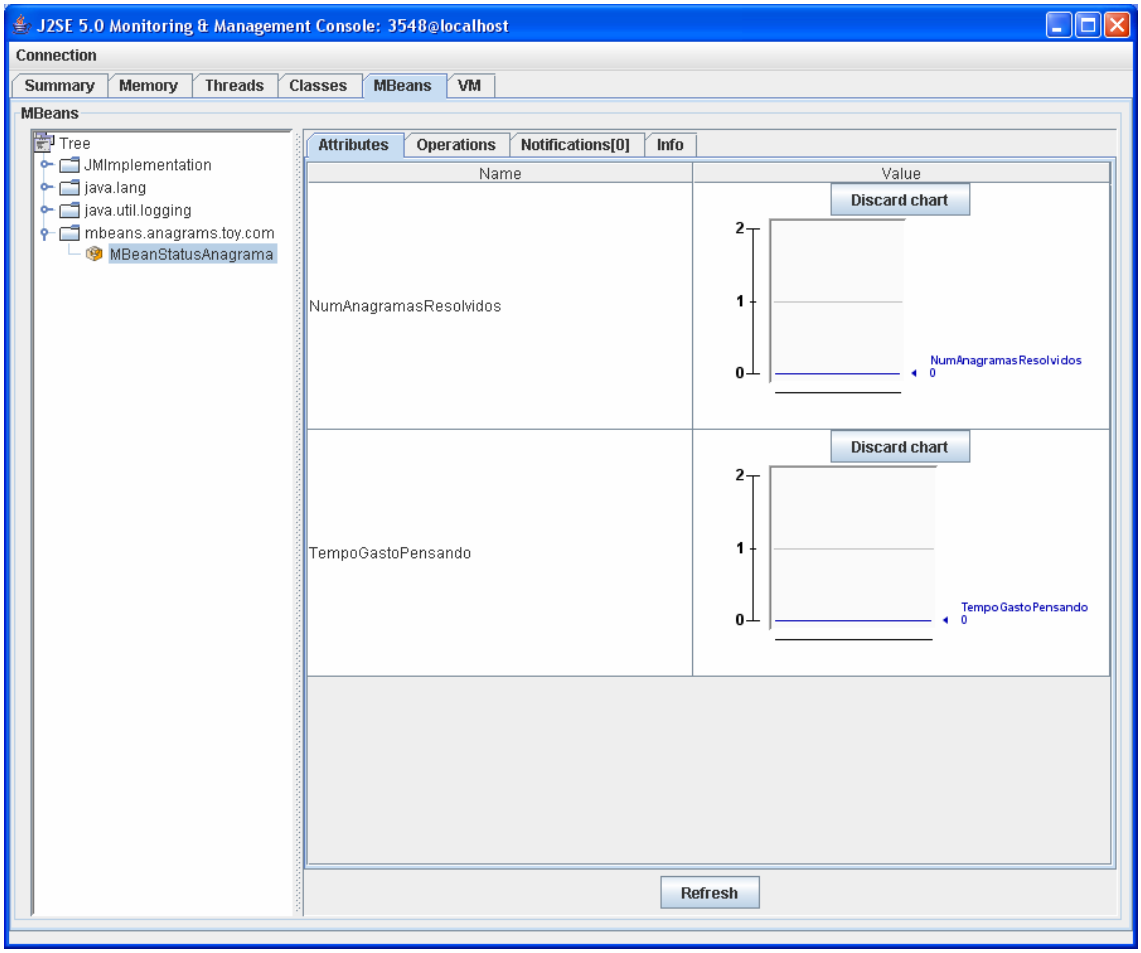

Figura 57: Tela do JConsole na aba “Attributes” com os atributos expostos no formato de gráfico

3.10.1.1 Simulação do uso do Jogo para a Monitoração de Notificações

A seguir é apresentada a simulação de uma jogada e a análise realizada pelo JMX, após a realização da preparação do JConsole. Após o início do jogo, para realizar o teste de

gerenciamento, deve-se tentar resolver o anagrama. Para isso, faz-se necessário analisar as palavras apresentadas no campo *Scrambled Word* (Palavras Misturadas) e responder no campo *Your Guess* (Sua Suposição) na tela do Jogo de Anagramas (*Anagrams*). A resposta deverá ser baseada na construção de uma palavra bem formada no idioma inglês utilizando as mesmas palavras apresentadas no campo *Scrambled Word*.

Para realizar a análise do exemplo deve-se inserir a palavra "*abstraction*" no campo *Your Guess* (Sua Suposição) e selecionar a opção "*Guess*" (Suposição). Deve-se observar que será computada a resposta certa e o tempo gasto pra a resolver o anagrama, que será representado no gráfico do JConsole.

Deve-se selecionar a opção "New Word" (Nova Palavra) para uma nova jogada. Para continuar a realização da análise do exemplo deve-se inserir a palavra "*ambiguous*" no campo *Your Guess* (Sua Suposição) e selecionar a opção "*Guess*" (Suposição). O gráfico representará o número de anagramas resolvidos e o tempo gasto para resolver, ambos os atributos serão alterados. Continuando com análise, deve-se inserir a palavra "*arithmetic*" no campo *Your Guess* (Sua Suposição) e selecionar a opção "*Guess*" (Suposição).

A figura 58 apresenta os resultados dos testes no gráfico, porém os valores apresentados no atributo Tempo Gasto Pensando são relativos e deverá ser desconsiderado em relação a uma nova simulação, uma vez que se, realizada novamente, Tempo Gasto Pensando deverá apresentar outro valor.

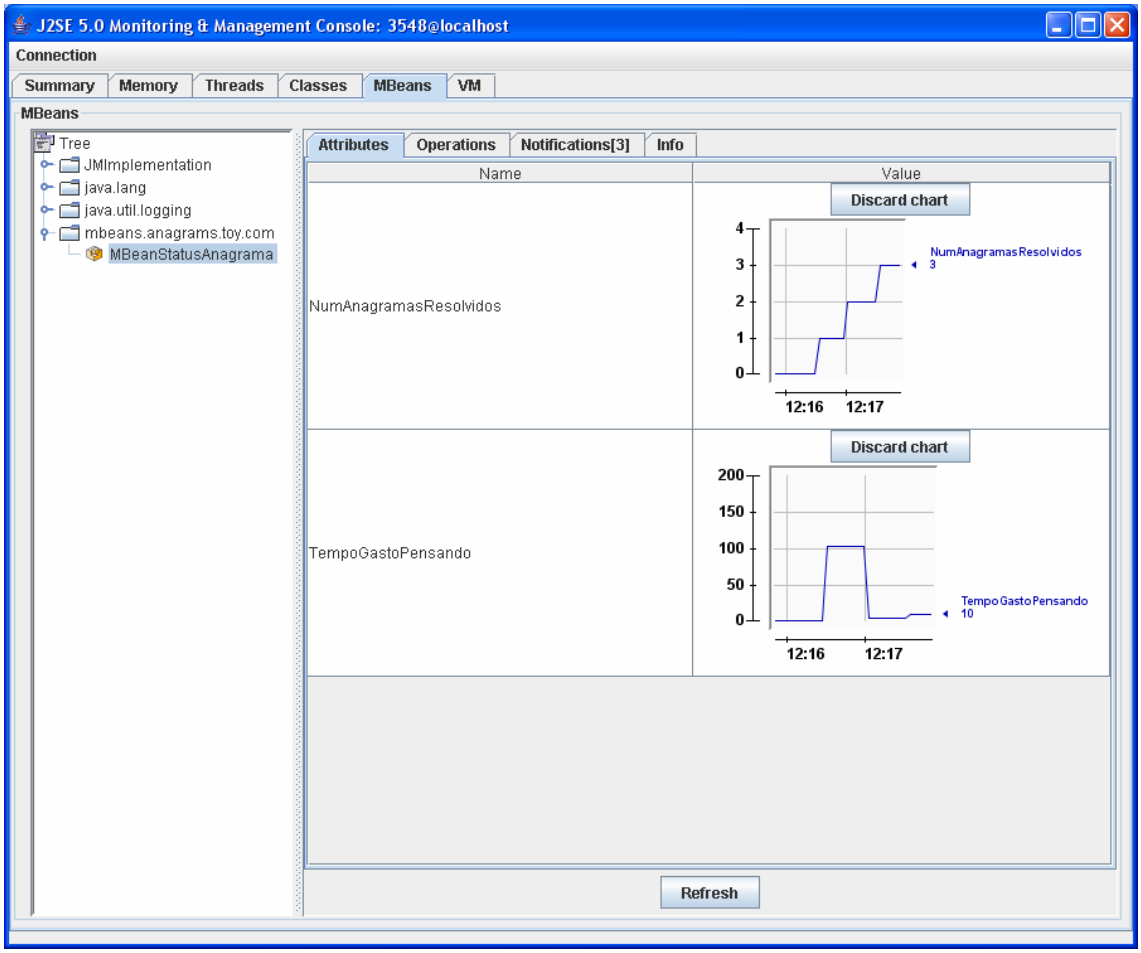

Figura 58: Tela do JConsole na aba "*Attributes*" com os atributos expostos no formato de gráfico representando o resultado da análise das jogadas

3.10.1.2 Simulação da Monitoração do Recurso de Memória

Para verificar graficamente o total e a quantidade disponível de memória no computador em que a aplicação está sendo executada, deve-se selecionar a aba "MBeans" na Tela da Aplicação de Gerenciamento (JConsole) e expandir o caminho "*Tree*" + "*java.lang*" + "*OperatingSystem*" no campo MBeans. Em seguida, para exibir o gráfico, deve-se clicar duas vezes nos valores da coluna "*Value*" e nas seguintes linhas "*FreePhysicalMemorySize*" (Tamanho da Memória Física Livre) e "*TotalPhysicalMemorySize*" (Tamanho Total da Memória Física). A figura 59 apresenta os gráficos com consumo de memória do computador em que está executando a aplicação.

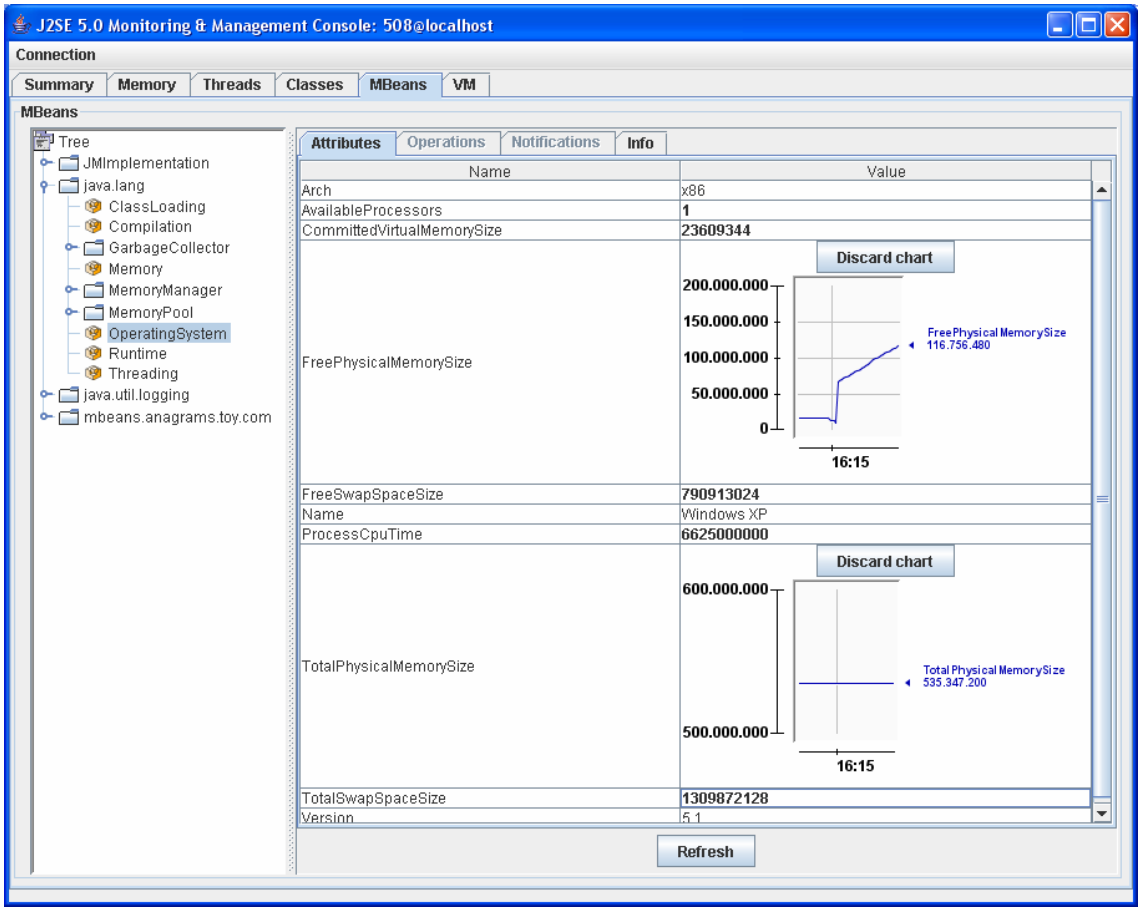

Figura 59: Tela do JConsole na aba "*Attributes*" com alguns atributos expostos no formato de gráfico

Inúmeros recursos gastos na execução de aplicações podem ser monitorados pelo JMX. Nesse trabalho foi apresentada a monitoração da memória para exemplificar que é possível a monitoração de recursos através da JVM.

Como dito na seção 2.3.10, o JConsole apresenta funcionalidades que exibem informações sobre *Threads* e Memória de forma completa e confiável. Ainda pelo JConsole pode-se saber o consumo de recursos em tempo de execução das aplicações Java que estão sendo executadas. Um exemplo disso é a informação disposta através da aba "*Memory*" na Tela da Aplicação de Gerenciamento (JConsole). A figura 60 apresenta as informações de consumo de memória por parte do Jogo de Anagramas.

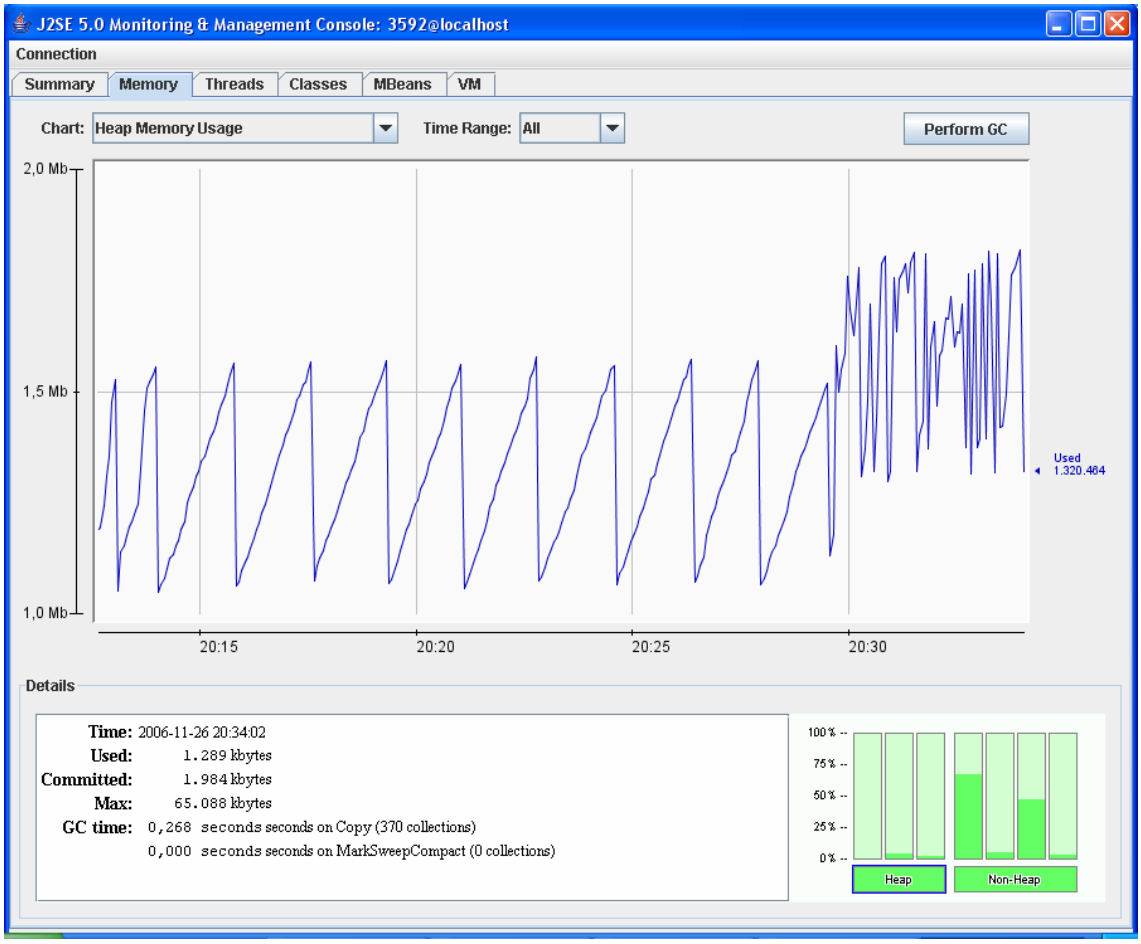

Figura 60: Tela do JConsole na aba “Memory”

4 Teste Realizado

As subseções seguintes irão representar o teste realizado com a execução de cada etapa necessária para o desenvolvimento do gerenciamento de uma aplicação Java. Para o desenvolvimento da aplicação gerenciável e dos recursos de gerenciamento foi utilizada a IDE NetBeans. As etapas abordadas a seguir foram baseadas no exemplo apresentado na seção 3.

O objetivo específico do teste é o de realizar o gerenciamento e análise de uma simples aplicação que consuma muitos recursos físicos computacionais, possibilitando a fácil percepção do consumo dos recursos utilizados.

4.1 Definição e Implementação da Aplicação Java

Nesta etapa realizou-se a análise e definição da aplicação a ser gerenciada. Dentre várias aplicações analisadas, foi escolhida uma aplicação que apresenta o máximo de combinações possíveis do acontecimento de um evento determinado por um usuário. A aplicação analisa o maior número de combinações possíveis. Ela simplesmente receberá a quantidade de números que serão comparados, entre 2 a 5, e também, os números que serão comparados.

Esta aplicação foi implementada e testada antes de iniciar a implementação das classes de gerenciamento. A figura 61 apresenta o principal trecho do código da aplicação que será gerenciado pelo JMX.

Esta etapa poderia ser desnecessária caso não fosse um teste objetivo. Para isso, bastava fornecer alguma aplicação Java para ser gerenciada.

```
1     for(int i=0; i<N; i++){
2           for(int j=0;j<N;j++){
3                 if(N==2){
4                       resultado+= num[i]+" "+num[j];
5                       resultado+="\n";
6                       quant++;
7                 }else{
8                       for(int k=0;k<N;k++){
9                             if(N==3){
10                                  resultado+=num[i]+" "+num[j]+" "+num[k];
11                                  resultado+="\n";
12                                  quant++;
13                            }else{
14                                  for(int l=0;l<N;l++){
15                                        if(N==4){
16                                              resultado+=num[i]+"
17    "+num[j]+" "+num[k]+" "+num[k];
18                                              resultado+="   ";
19                                              quant++;
20                                        }else{
21                                              for(int m=0;m<N;m++){
22
23                                                    resultado+=num[i]+     "
24    "+num[j]+" "+num[k]+" "+num[l]+" "+num[m];
25                                                    resultado+="    ";
26                                                          quant++;
27                                              }
28                                        }
29                                  }
30                            }
31                      }
32                }
33                resultado+="\n";
34          }
35    }
```

Figura 61: Techo de código da aplicação gerenciada “Combinações.java”

Como pode-se notar na figura 61, este código fará comparações dimensionais, uma vez que ele apresenta laços de repetição (*for*) dentro de um outro laço de repetição (*for*), provocando assim, alto consumo de recursos físicos quando for executado. Logo, esta aplicação supre a expectativa para o gerenciamento proposto na escolha da aplicação para ser testada.

4.2 Implementação dos Recursos para Gerenciamento da Aplicação Java

Antes de iniciar a implementação dos objetos necessários para o gerenciamento da aplicação escolhida na seção 4.1, foram analisadas as informações a serem manipuladas na aplicação. Como o objetivo do teste e o de obter informações referentes capacidade tolerarada pelos recursos físicos dum determinado ambiente em que será executada determinada aplicação, foram escolhidos alguns parâmetros, dentre eles: tempo consumido para o processamento das combinações possíveis e a quantidade de memória consumida pela aplicação.

Em seguida, foram desenvolvidas duas classes, sendo uma a *Interface* do MBean e a outra o MBean completo. Ambas foram desenvolvidas de modo semelhante ao apresentado na seção 3.

A *Interface* do MBean representa a *interface* das variáveis e operações (métodos) do MBean que serão apresentadas visualmente ao gerente durante a monitoração da aplicação.

O MBean foi implementado para disponibilizar o tempo consumido para o processamento das combinações possíveis em milésimos de segundo e a função de calcular o tempo consumido para o processamento das combinações possíveis em segundos.

Uma vez que será utilizado o JConsole para a realização da gerência, não é necessário instrumentar recursos relativos ao consumo da memória, pois estes recursos já serão oferecidos de forma nativa pelo JConsole, tornando inconveniente a recriação da instrumentação do recurso.

Todos os métodos e atributos listados na interface do MBean serão apresentados na aba de “MBeans” do JConsole, permitindo assim, o controle e monitoração do objeto gerenciado. Para o teste foram listados: os atributos tempoDeProcessamentoMilesimos e tempoDeProcessamentoSegundos e a operação TempoEmSegundos. A figura 62 apresenta interface do MBean que apresentam os recursos que serão visualizados no momento da gerência.

```
1   public interface MBeanCombinacoesMBean{
2           public int getTempoDeProcessamentoMilesimos();
3           public int getTempoDeProcessamentoSegundos();
4           public void tempoEmSegundos();
5   }
```

Figura 62: Techo da Interface do MBean

Para testar o MBean foi implementado um JUnit de teste. Esta etapa não é obrigatória para o desenvolvimento do gerenciamento de aplicações utilizando a API JMX, porém essa prática torna-se importante em aplicações de médio e grande porte, que em caso de erro dificilmente será detectado causando, assim, falhas e imprecisões na aplicação de gerenciamento, foi abordada neste teste com o intuito de ser uma boa prática.

Em seguida, foi implementado o Agente JMX responsável pelo controle dos recursos e serviços da máquina local, isso através do Servidor de MBeans. O nível Agente que será responsável pela execução das tarefas solicitadas pelo gerente sobre os objetos gerenciados. Logo, na criação do Agente JMX o MBean foi registrado. Sem este registro é impossível o uso do MBean pela aplicação de gerenciamento, o registro é uma tarefa simples e de rápido desenvolvimento. A figura 63 apresenta o código do registro do MBean.

```
1   public void init() throws Exception {
2           mbean = new MBeanCombinacoes();
3           getMBeanServer().registerMBean(mbean, new
4   ObjectName("mbeans.comb.toy.com:type=MBeanCombinacoes"));
5   }
```

Figura 63: Techo de código com o registro do MBean no Agente JMX

Posteriormente, foram criadas algumas chamadas aos métodos do MBean para realizar a contagem de tempo gasto para o processamento das possíveis combinações de um evento, estas chamadas foram introduzidas em pontos estratégicos para que o tempo a ser calculado pelo MBean seja real. Logo, foi introduzida uma chamada no início do código apresentado na figura 61 para iniciar a contagem e outro ao final do código apresentado na mesma figura para parar a contagem do tempo.

4.3 Gerenciamento do Teste Implementado Utilizando o JConsole

Inicialmente, todas as classes foram compiladas e a classe principal do projeto foi executada juntamente com o JConsole, através do NetBeans. Foi gerenciado o tempo consumido para processar as combinações possíveis na comparação de números fornecidos no inicio da execução da aplicação, a variável constante do tempo é apresentada em

milésimos de segundo para exibir com exatidão os valores do tempo. A figura 64 apresenta a tela de gerenciamento com alguns resultados obtidos.

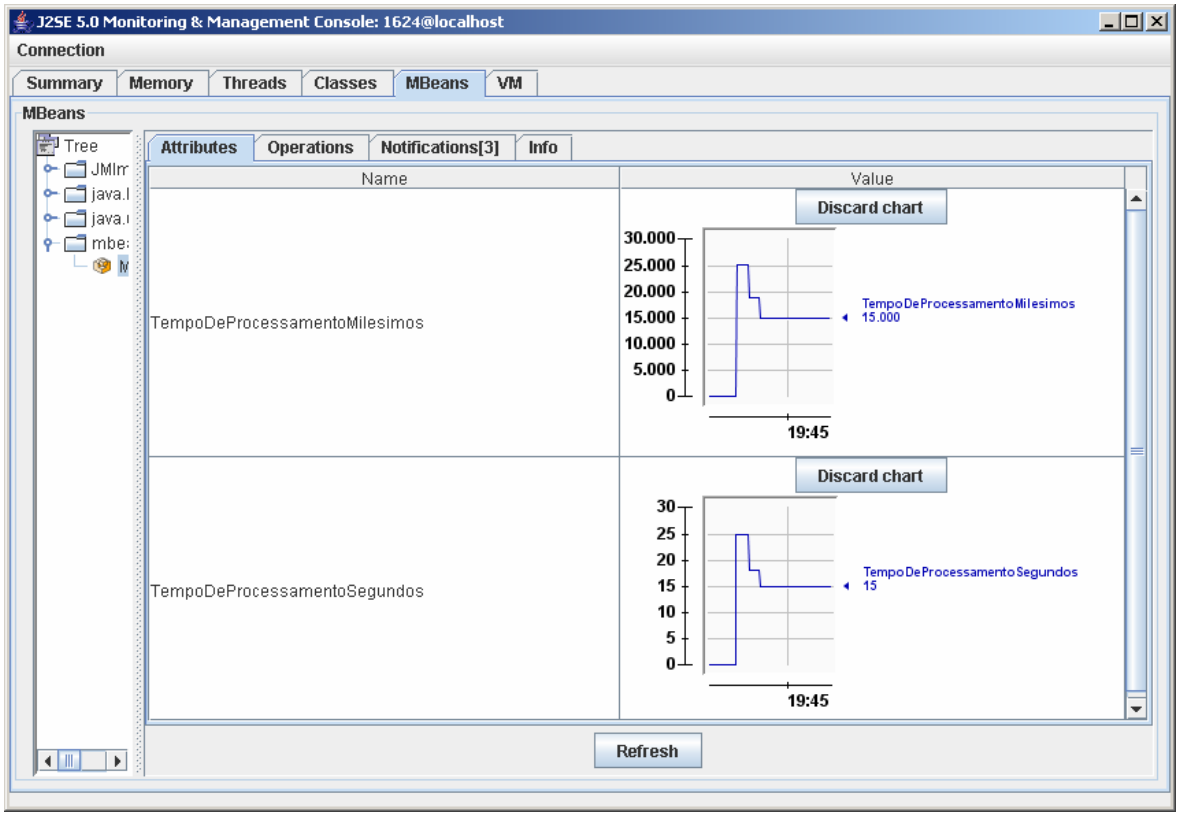

Figura 64: Tela do JConsole durante o gerenciamento da aplicação

O tempoDeProcessamentoSegundo foi obtido através da invocação da Operação tempoEmSegundo(), que é apresentado na figura 65. A operação transforma o resultado da análise obtida em milésimos de segundos para segundos.

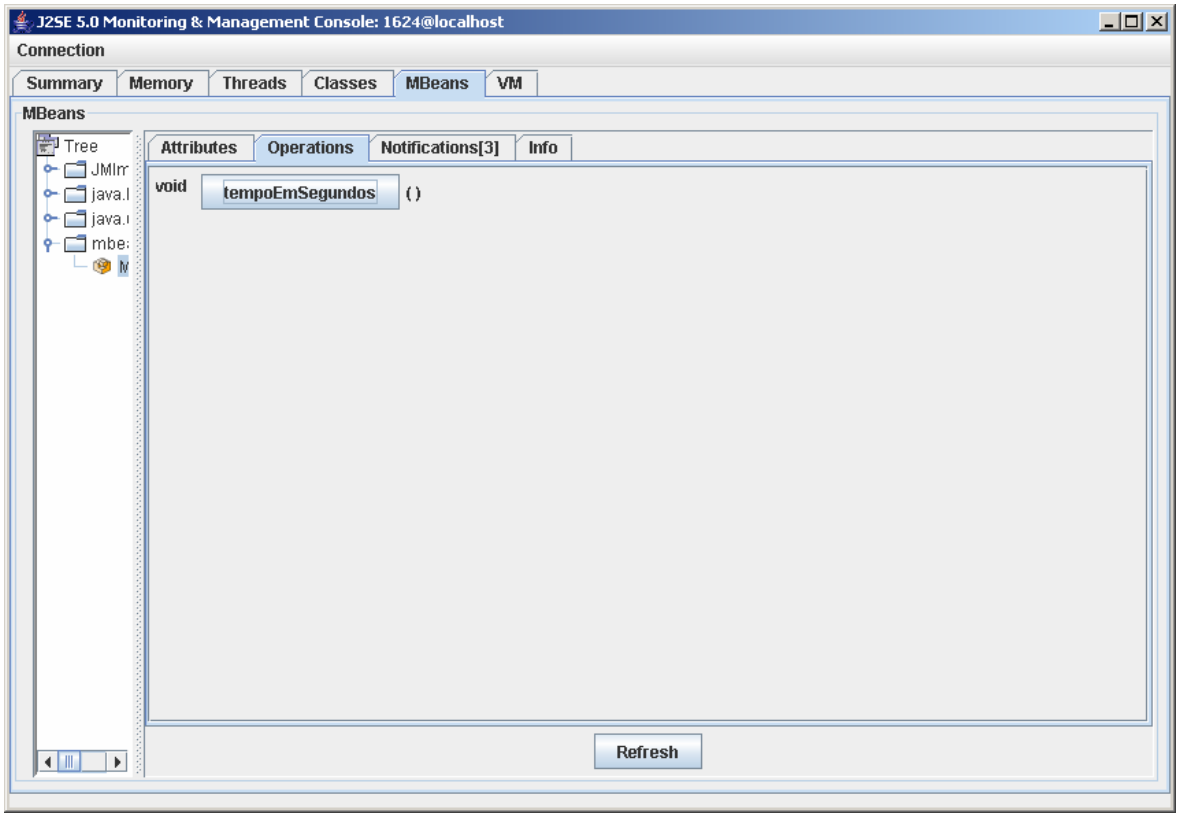

Figura 65: Tela do JConsole na aba de operações com a operação instrumentada na aplicação

Em seguida, foram analisados e testados os parâmetros de memória. E em relação ao consumo de memória durante a execução da aplicação pôde-se notar o consumo variando entre 2.5Mb a 3.0Mb.

Como o JConsole apresenta uma ferramenta nativa do Java o *garbage collector* ou simplesmente coletores, que faz o controle da memória do objeto gerenciado que automaticamente libera blocos de memória que não serão mais usados em uma aplicação. Esta ferramenta foi utilizada e testada para verificar o consumo de memória da aplicação após seu o uso. Para utilizá-la foi necessário apenas selecionar a opção "*Perform GC*" implementada na aba "*Memory*" do JConsole. O *garbage collector* é uma ferramenta muito eficiente e por isso foi abordada neste teste.

A figura 66 apresenta o uso da memória durante a execução da aplicação. Até o período das 20 (vinte) horas e 1 (um) minuto o consumo de memória pela aplicação oscilava entre 2.5Mb a 3.0Mb, logo após a execução da operação do *perform garbage collector* passou a oscilar entre 1.9 a 2.4. Através dos gráfico apresentado na figura 66

pôde-se então comprovar a eficiência do *garbage collector* na liberação de memória utilizada pela aplicação.

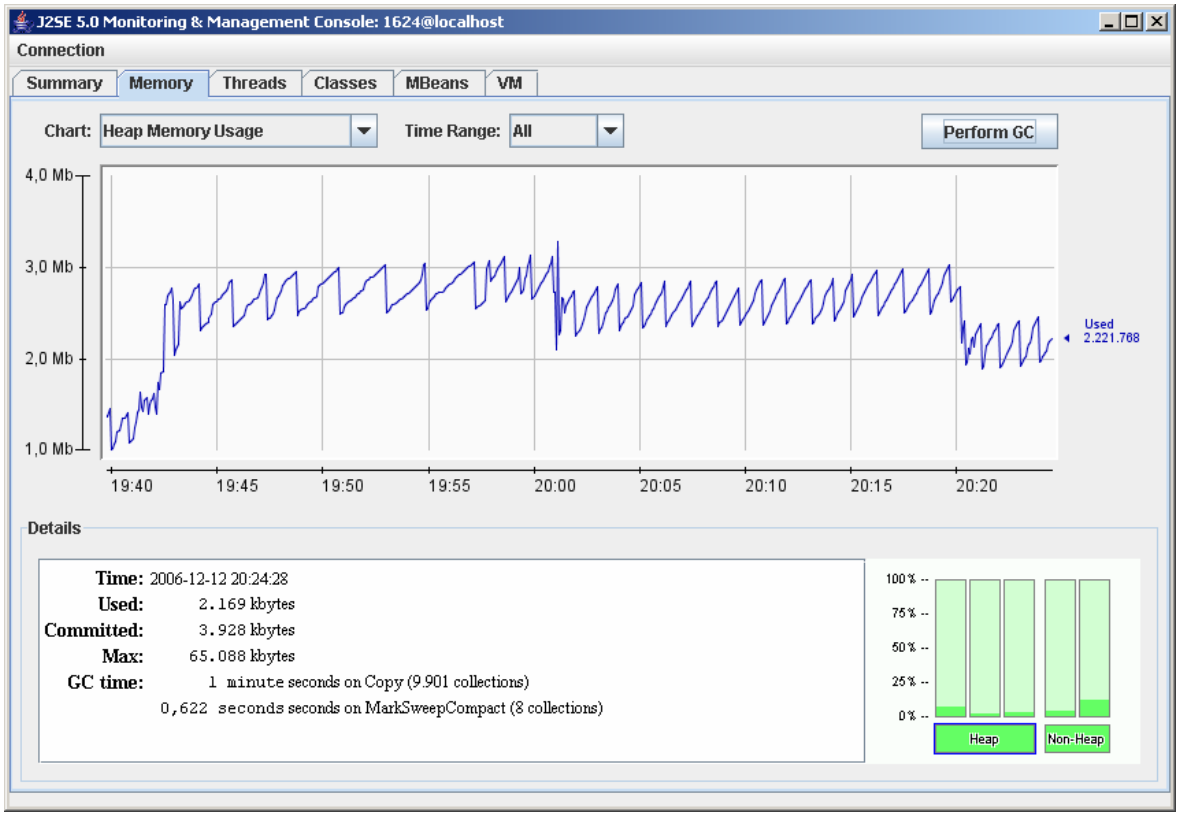

Figura 65: Tela do JConsole na aba de operações com a operação instrumentada na aplicação

5 Considerações Finais

Através do exemplo e do teste realizado neste documento é possível perceber a amplitude das informações que podem ser trabalhadas nas aplicações Java com a utilização da API JMX, que vão das informações de componente físicos locais ao alto nível das aplicações Java que podem ser gerenciadas localmente ou remotamente.

Um exemplo da amplitude da capacidade de gerenciamento pode ser percebida na gerência do servidor de aplicação JBoss, servidor este desenvolvido pela *Sun Microsysten* para ser totalmente gerenciado utilizando a API JMX e que possui como característica a estabilidade, mesmo em ambientes com alto número de acesso e serviços. Atualmente inúmeras corporações fazem uso deste servidor de aplicação, inclusive a Secretaria da Fazenda do Estado do Tocantins.

No exemplo apresentado na seção 3 é notável a facilidade de implementação e uso do gerenciamento de aplicações utilizando a API JMX. E no teste evidenciou-se, ainda mais, esta simplicidade do gerenciamento e ainda mostrou o nível das informações que podem ser trabalhadas, utilizando até mesmo, aplicações de gerenciamento pré-definidas gratuitas como é o caso do JConsole.

A partir do teste e do estudo realizado sobre a API JMX é possível perceber que as maiores dificuldades encontradas no desenvolvimento de gerenciamento em aplicações Java utilizando esta API é a análise e definição do gerenciamento da aplicação.

Para a definição da aplicação alguns itens deverão ser destacados, entre eles: como se deseja gerenciar os recursos e serviços; por qual método se deseja gerenciar os recursos e serviços; se será de forma local ou remota, por quem serão gerenciados os recursos e serviços, se utilizará alguma aplicação pronta, como o JConsole, MC4J, Spring WLS JMX Console ou o WLST, ou se desenvolverá uma nova aplicação de gerenciamento, caso se

desenvolva uma nova aplicação de gerenciamento quais os protocolos de comunicação adicionais serão utilizados.

Enfim, as maiores dificuldades surgem com a falta de conhecimento em relação ao desenvolvimento de sistemas de gerenciamento de redes e aplicações, pois estes deverão apresentar estabilidade e segurança ao administrador que fará uso das informações coletadas e configurações realizadas pelo sistema.

As ferramentas para implementação de gerenciamento de aplicações Java são muito simples e de fácil utilização. E a API JMX é bem estável e bem documentada, e conta com várias IDE's (ambientes de desenvolvimento de código) que facilitam mais o processo de implementação de aplicações de gerenciamento.

As limitações definidas através do estudo e teste da API JMX, devem-se aos fatos dela trabalhar apenas aplicações Java e possui a necessidade da JVM (*Java Virtual Machine* – Máquina Virtual Java) em todos os dispositivos gerenciados, aumentando a necessidade de recursos computacionais no ambiente de gerenciamento.

Os pontos positivos levantados sobre a API JMX é que esta API trabalha com outros padrões de gerenciamento de redes, é multiplataforma, possui várias ferramentas de implementação, é estável e bem documentada.

6 Referências Bibliográficas

(FRANCISCO, 2001) *Adell Péricas*. **Gerência Dinâmica de Rede Baseada na Tecnologia Java JMX.** Disponível em: http://www.inf.furb.br/~pericas/publicacoes/GerenciaDinamicaJMX.pdf. Página: 03. Acessado: 05/10/2006.

(INDRUSIAK, 1996) *Leandro Soares*. **Linguagem Java.** Disponível em: http://www.inf.ufrgs.br/tools/java/introjava.pdf. Capítulo 02. Página: 03-05. Acessado: 05/10/2006.

(NETBEANS, 2006)[1] *NetBeans.org*. **NetBeans.** Disponível em: http://www.netbeans.org/products/ide/. Acessado: 18/09/2006.

(NETBEANS, 2006)[2] *NetBeans.org*. **Adding Java Management Extensions (JMX) Instrumentation to a Java Application .** Disponível em: http://www.netbeans.org/kb/articles/jmx-tutorial.html . Acessado: 18/09/2006.

(SUN, 2004) *Sun Microsystems*. **Java Management Extensions (JMX) Technology Overview.** Disponível em: http://java.sun.com/j2se/1.5.0/docs/guide/jmx/overview/architecture.html#wp996882 . Acessado: 18/09/2006.

(SUN, 2006)[1] *Sun Microsystems*. **JavaTM Management Extensions (JMX) Specification, version 1.4.** Disponível em: http://download.java.net/jdk6/docs/ technotes/guides/jmx/JMX_1_4_specification.pdf Acessado: 05/12/2006.

(SUN, 2006)[2] *Sun Microsystems*. **Java Management Extensions (JMX).** Disponível em: http://java.sun.com/j2se/1.5.0/docs/guide/jmx/index.html . Acessado: 18/09/2006.

(SUN, 2006)[3] *Sun Microsystems*. **Java SE Technologies at a Glance.** Disponível em: http://java.sun.com/javase/technologies/index.jsp. Acessado: 05/10/2006.

(VAZ, 2003) *Rodrigo Cardoso*. **JUnit – Framework para Testes em Java.** Disponível em: http://www.ulbra-to.br/ensino/43020/artigos/relatorios2003-2/TCC/JUnit.pdf. Páginas: 16 e 17. Acessado: 20/10/2006.

7 Anexos

O anexo 1 apresenta o código da classe “Anagrams.java” criada na seção 3.1.

```
1    package com.toy.anagrams.ui;
2
3    import com.toy.anagrams.lib.WordLibrary;
4    import java.awt.Dimension;
5    import java.awt.Point;
6    import java.awt.Toolkit;
7    import java.awt.event.ActionListener;
8    import java.awt.event.WindowListener;
9    import javax.swing.JFrame;
10
11   import com.toy.anagrams.agent.JMXAgent;
12   import com.toy.anagrams.mbeans.*;
13
14   public class Anagrams extends JFrame {
15      public static void main(String[] args) throws Exception {
16         new Anagrams().setVisible(true);
17      }
18      private int wordIdx = 0;
19      public Anagrams() throws Exception {
20         initComponents();
21         getRootPane().setDefaultButton(guessButton);
22         scrambledWord.setText(WordLibrary.getScrambledWord(wordIdx));
23         pack();
24         guessedWord.requestFocusInWindow();
25         Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
26         Dimension frameSize = getSize();
27         setLocation(new Point((screenSize.width - frameSize.width) / 2,
28                           (screenSize.height - frameSize.width) / 2));
29         initManagement();
30      }
31      private void initManagement() throws Exception {
32        JMXAgent agent = JMXAgent.getDefault();
33        agent.getAnagramsStats().startThinking();
34
35   agent.getAnagramsStats().setCurrentAnagram(WordLibrary.getScrambledWor
```

```
36   d(wordIdx));
37      }
38      private void initComponents() {
39         java.awt.GridBagConstraints gridBagConstraints;
40         mainPanel = new javax.swing.JPanel();
41         scrambledLabel = new javax.swing.JLabel();
42         scrambledWord = new javax.swing.JTextField();
43         guessLabel = new javax.swing.JLabel();
44         guessedWord = new javax.swing.JTextField();
45         feedbackLabel = new javax.swing.JLabel();
46         buttonsPanel = new javax.swing.JPanel();
47         guessButton = new javax.swing.JButton();
48         nextTrial = new javax.swing.JButton();
49         mainMenu = new javax.swing.JMenuBar();
50         fileMenu = new javax.swing.JMenu();
51         aboutMenuItem = new javax.swing.JMenuItem();
52         exitMenuItem = new javax.swing.JMenuItem();
53         setTitle("Anagrams");
54         addWindowListener(new java.awt.event.WindowAdapter() {
55            public void windowClosing(java.awt.event.WindowEvent evt) {
56               exitForm(evt);
57            }
58         });
59         mainPanel.setLayout(new java.awt.GridBagLayout());
60         mainPanel.setBorder(new javax.swing.border.EmptyBorder(new
61   java.awt.Insets(12, 12, 12, 12)));
62         mainPanel.setMinimumSize(new java.awt.Dimension(297, 200));
63         scrambledLabel.setText("Scrambled Word:");
64         gridBagConstraints = new java.awt.GridBagConstraints();
65         gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
66         gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
67         gridBagConstraints.insets = new java.awt.Insets(0, 0, 12, 6);
68         mainPanel.add(scrambledLabel, gridBagConstraints);
69         scrambledWord.setColumns(20);
70         scrambledWord.setEditable(false);
71         gridBagConstraints = new java.awt.GridBagConstraints();
72         gridBagConstraints.gridwidth =
73   java.awt.GridBagConstraints.REMAINDER;
74         gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
75         gridBagConstraints.weightx = 1.0;
76         gridBagConstraints.insets = new java.awt.Insets(0, 0, 12, 0);
77         mainPanel.add(scrambledWord, gridBagConstraints);
78         guessLabel.setDisplayedMnemonic('Y');
79         guessLabel.setLabelFor(guessedWord);
80         guessLabel.setText("Your Guess:");
81         gridBagConstraints = new java.awt.GridBagConstraints();
82         gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
83         gridBagConstraints.anchor = java.awt.GridBagConstraints.WEST;
84         gridBagConstraints.insets = new java.awt.Insets(0, 0, 20, 6);
85         mainPanel.add(guessLabel, gridBagConstraints);
86         guessedWord.setColumns(20);
87         gridBagConstraints = new java.awt.GridBagConstraints();
88         gridBagConstraints.gridwidth =
89   java.awt.GridBagConstraints.REMAINDER;
90         gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
91         gridBagConstraints.weightx = 1.0;
```

```
92          gridBagConstraints.insets = new java.awt.Insets(0, 0, 20, 0);
93          mainPanel.add(guessedWord, gridBagConstraints);
94          feedbackLabel.setText(" ");
95          gridBagConstraints = new java.awt.GridBagConstraints();
96          gridBagConstraints.gridx = 1;
97          gridBagConstraints.gridwidth =
98    java.awt.GridBagConstraints.REMAINDER;
99          gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
100         gridBagConstraints.weightx = 1.0;
101         gridBagConstraints.insets = new java.awt.Insets(0, 0, 20, 0);
102         mainPanel.add(feedbackLabel, gridBagConstraints);
103         buttonsPanel.setLayout(new java.awt.GridBagLayout());
104         guessButton.setMnemonic('G');
105         guessButton.setText("Guess");
106         guessButton.setToolTipText("Guess the scrambled word.");
107         guessButton.addActionListener(new java.awt.event.ActionListener() {
108            public void actionPerformed(java.awt.event.ActionEvent evt) {
109               guessedWordActionPerformed(evt);
110            }
111         });
112         gridBagConstraints = new java.awt.GridBagConstraints();
113         gridBagConstraints.gridheight =
114   java.awt.GridBagConstraints.REMAINDER;
115         gridBagConstraints.anchor = java.awt.GridBagConstraints.SOUTHEAST;
116         gridBagConstraints.weightx = 1.0;
117         gridBagConstraints.weighty = 1.0;
118         gridBagConstraints.insets = new java.awt.Insets(0, 0, 0, 6);
119         buttonsPanel.add(guessButton, gridBagConstraints);
120         nextTrial.setMnemonic('N');
121         nextTrial.setText("New Word");
122         nextTrial.setToolTipText("Fetch a new word.");
123         nextTrial.addActionListener(new java.awt.event.ActionListener() {
124            public void actionPerformed(java.awt.event.ActionEvent evt) {
125               nextTrialActionPerformed(evt);
126            }
127         });
128         gridBagConstraints = new java.awt.GridBagConstraints();
129         gridBagConstraints.gridwidth =
130   java.awt.GridBagConstraints.REMAINDER;
131         gridBagConstraints.gridheight =
132   java.awt.GridBagConstraints.REMAINDER;
133         gridBagConstraints.anchor = java.awt.GridBagConstraints.SOUTHEAST;
134         gridBagConstraints.weighty = 1.0;
135         buttonsPanel.add(nextTrial, gridBagConstraints);
136         gridBagConstraints = new java.awt.GridBagConstraints();
137         gridBagConstraints.gridwidth =
138   java.awt.GridBagConstraints.REMAINDER;
139         gridBagConstraints.gridheight =
140   java.awt.GridBagConstraints.REMAINDER;
141         gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
142         gridBagConstraints.weighty = 1.0;
143         mainPanel.add(buttonsPanel, gridBagConstraints);
144         getContentPane().add(mainPanel, java.awt.BorderLayout.CENTER);
145         fileMenu.setMnemonic('F');
146         fileMenu.setText("File");
147         aboutMenuItem.setMnemonic('A');
```

```
148       aboutMenuItem.setText("About");
149       aboutMenuItem.setToolTipText("About");
150       aboutMenuItem.addActionListener(new java.awt.event.ActionListener() {
151          public void actionPerformed(java.awt.event.ActionEvent evt) {
152             aboutMenuItemActionPerformed(evt);
153          }
154       });
155       fileMenu.add(aboutMenuItem);
156       exitMenuItem.setMnemonic('E');
157       exitMenuItem.setText("Exit");
158       exitMenuItem.setToolTipText("Quit Team, Quit!");
159       exitMenuItem.addActionListener(new java.awt.event.ActionListener() {
160          public void actionPerformed(java.awt.event.ActionEvent evt) {
161             exitMenuItemActionPerformed(evt);
162          }
163       });
164       fileMenu.add(exitMenuItem);
165       mainMenu.add(fileMenu);
166       setJMenuBar(mainMenu);
167    }
168    private void aboutMenuItemActionPerformed(java.awt.event.ActionEvent
169 evt) {
170       new About(this).setVisible(true);
171    }
172    private void nextTrialActionPerformed(java.awt.event.ActionEvent evt) {
173       wordIdx = (wordIdx + 1) % WordLibrary.getSize();
174       try {
175          JMXAgent.getDefault().getAnagramsStats().startThinking();
176 JMXAgent.getDefault().getAnagramsStats().setCurrentAnagram(WordLibrary.g
177 etScrambledWord(wordIdx));
178       }catch(Exception e) {e.printStackTrace();}
179       feedbackLabel.setText(" ");
180       scrambledWord.setText(WordLibrary.getScrambledWord(wordIdx));
181       guessedWord.setText("");
182       getRootPane().setDefaultButton(guessButton);
183       guessedWord.requestFocusInWindow();
184    }
185    private void exitMenuItemActionPerformed(java.awt.event.ActionEvent evt)
186 {
187       System.exit(0);
188    }
189    private void guessedWordActionPerformed(java.awt.event.ActionEvent evt)
190 {
191       if (WordLibrary.isCorrect(wordIdx, guessedWord.getText())){
192          try{
193             JMXAgent.getDefault().getAnagramsStats().stopThinking();
194          }catch(Exception e) {e.printStackTrace();}
195          feedbackLabel.setText("Correct! Try a new word!");
196          getRootPane().setDefaultButton(nextTrial);
197       } else {
198          feedbackLabel.setText("Incorrect! Try again!");
199          guessedWord.setText("");
200       }
201       guessedWord.requestFocusInWindow();
202    }
203    private void exitForm(java.awt.event.WindowEvent evt) {
```

```
204         System.exit(0);
205     }
206     private javax.swing.JMenuItem aboutMenuItem;
207     private javax.swing.JPanel buttonsPanel;
208     private javax.swing.JMenuItem exitMenuItem;
209     private javax.swing.JLabel feedbackLabel;
210     private javax.swing.JMenu fileMenu;
211     private javax.swing.JButton guessButton;
212     private javax.swing.JLabel guessLabel;
213     private javax.swing.JTextField guessedWord;
214     private javax.swing.JMenuBar mainMenu;
215     private javax.swing.JPanel mainPanel;
216     private javax.swing.JButton nextTrial;
217     private javax.swing.JLabel scrambledLabel;
218     private javax.swing.JTextField scrambledWord;
219 }
```

Anexo 1: Código inicial da classe "Anagrams.java"

O anexo 2 apresenta o código da classe "WordLibrary.java" criada na seção 3.1.

```
1     public final class WordLibrary {
2         private static final String[] WORD_LIST = {
3             "abstraction",
4             "ambiguous",
5             "arithmetic",
6             "backslash",
7             "bitmap",
8             "circumstance",
9             "combination",
10            "consequently",
11            "consortium",
12            "decrementing",
13            "dependency",
14            "disambiguate",
15            "dynamic",
16            "encapsulation",
17            "equivalent",
18            "expression",
19            "facilitate",
20            "fragment",
21            "hexadecimal",
22            "implementation",
23            "indistinguishable",
24            "inheritance",
25            "internet",
26            "java",
27            "localization",
28            "microprocessor",
29            "navigation",
30            "optimization",
31            "parameter",
32            "patrick",
33            "pickle",
```

```
34          "polymorphic",
35          "rigorously",
36          "simultaneously",
37          "specification",
38          "structure",
39          "lexical",
40          "likewise",
41          "management",
42          "manipulate",
43          "mathematics",
44          "hotjava",
45          "vertex",
46          "unsigned",
47          "traditional"};
48      private static final String[] SCRAMBLED_WORD_LIST = {
49          "batsartcoin",
50          "maibuguos",
51          "ratimhteci",
52          "abkclssha",
53          "ibmtpa",
54          "iccrmutsnaec",
55          "ocbmnitaoni",
56          "ocsnqeeutnyl",
57          "ocsnroitmu",
58          "edrcmeneitgn",
59          "edepdnneyc",
60          "idasbmgiauet",
61          "ydanicm",
62          "neacsplutaoni",
63          "qeiuaveltn",
64          "xerpseisno",
65          "aficilatet",
66          "rfgaemtn",
67          "ehaxedicalm",
68          "milpmeneatitno",
69          "niidtsniugsiahleb",
70          "niehiratcen",
71          "nietnret",
72          "ajav",
73          "olacilazitno",
74          "imrcpoorecssro",
75          "anivagitno",
76          "poitimazitno",
77          "aparemert",
78          "aprtcki",
79          "ipkcel",
80          "opylomprich",
81          "irogorsuyl",
82          "isumtlnaoesuyl",
83          "psceficitaoni",
84          "tsurtcreu",
85          "elixalc",
86          "ilekiwse",
87          "amanegemtn",
88          "aminupalet",
89          "amhtmetacsi",
```

```
90          "ohjtvaa",
91          "evtrxe",
92          "nuisngde",
93          "rtdatioialn"
94      };
95      private WordLibrary() {
96      }
97      public static String getWord(int idx) {
98          return WORD_LIST[idx];
99      }
100     public static String getScrambledWord(int idx) {
101         return SCRAMBLED_WORD_LIST[idx];
102     }
103     public static int getSize() {
104         return WORD_LIST.length;
105     }
106     public static boolean isCorrect(int idx, String userGuess) {
107         return userGuess.equals(getWord(idx));
108     }
109 }
```

Anexo 2: Código da classe “WordLibrary.java”