EXTRA

Capítulo 15
Programação orientada por objectos

Ciclo de vida de software

Regras de boa conduta no desenvolvimento de programas com programação orientada por objectos

Identificação de novas classes e métodos, incluindo o uso de cartões CRC

Identificação de relações de herança, agregação e dependência entre classes e respectiva descrição através de diagramas UML

Ciclo de vida de software

- Consideram-se todas as actividades, desde a análise inicial até ao software se tornar obsoleto
- O processo formal de desenvolvimento de software descreve as fases do processo de desenvolvimento e fornece indicações sobre a forma de implementação dessas fases
- Fases do processo de desenvolvimento
  - análise
  - design
  - implementação
  - testes
  - instalação

Análise e design

Análise

definir o que é suposto o projecto executar

não pensar na forma como o programa executará as tarefas

☑ documento de requisitos:

- descrição do que fará o programa
- manual do utilizador, com o modo de utilização do programa
- critérios de performance

Design

planificar a implementação do sistema

identificar estruturas subjacentes ao problema a resolver

identificar classes e métodos necessários

☑ descrição de classes e métodos

☑ diagramas mostrando as relações entre classes

Implementação, testes e instalação

Implementação

escrever e compilar código, o qual implementa as classes e os métodos identificados na fase de design

☑ programa completo

Testes

executar testes para verificar se o programa funciona correctamente

☑ relatório sobre testes e respectivos resultados

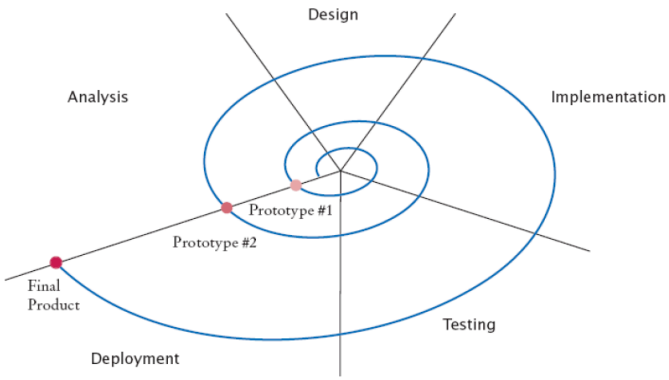
Instalação

instalar o programa

utilizar do programa

Modelo de desenvolvimento em espiral

- Processo com várias iterações. Numa primeira fase, a ênfase é colocada na construção de protótipos. Note-se que a experiência no desenvolvimento de um protótipo é reutilizada na iteração seguinte



- Este modelo apresenta um problema: existindo demasiadas iterações, o seu desenvolvimento pode implicar mais tempo do que desejável

Design orientado por objectos

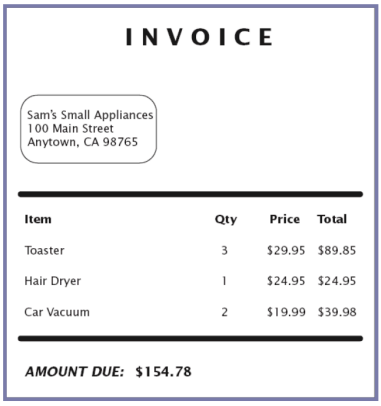
1. Identificação de classes

2. Responsabilidades de cada classe

3. Relação entre classes

Identificação de classes

- Uma classe representa de algum modo um conceito útil. Podem ser entidades concretas como por exemplo, contas bancárias, produtos ou rectângulos. Podem também ser conceitos abstractos como é o caso de janelas e streams
- A identificação de classes é feita normalmente segundo os nomes próprios existentes na descrição de tarefas
- De seguida, define-se o comportamento de cada classe
- Por fim, identificam-se os métodos olhando para os verbos referenciados na descrição de tarefas



INVOICE

Sam's Small Appliances
100 Main Street
Anytown, CA 98765

Item	Qty	Price	Total
Toaster	3	$29.95	$89.85
Hair Dryer	1	$24.95	$24.95
Car Vacuum	2	$19.99	$39.98

AMOUNT DUE: $154.78

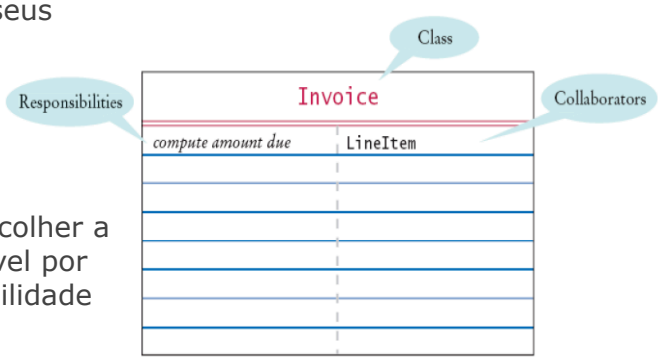
talvez ... Invoice, LineItem, Customer, ... !!

Identificação de classes

- Uma classe representa um conjunto de objectos com o mesmo comportamento. Deste modo, entidades com várias ocorrências na descrição do problema são bons candidatos para objectos; deve-se determinar o que têm em comum e assim desenhar as classes de modo as estas representarem essas características comuns
- Classes representam entidades como objectos, outras como tipos primitivos. Por exemplo, devemos ter uma classe endereço ou usar simplesmente uma string
- Nem todas as classes serão identificadas na fase de análise inicial
- Algumas classes até já podem existir

Cartões CRC
(Classe, Responsabilidades, Colaboradores)

- Um cartão CRC descreve uma classe, as suas responsabilidades e os seus colaboradores
- Metodologia:
  - um cartão para cada classe
  - para cada método (verbo), escolher a classe que deve ser responsável por ele e escrever essa responsabilidade no respectivo cartão
  - indicar quais as outras classes que são necessárias para implementar essa responsabilidade (os colaboradores)



Relação entre classes

Herança

Agregação

Dependência

Herança
relação *is-a*

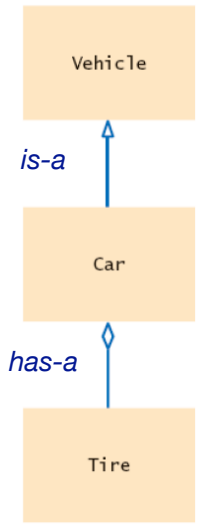
- Relação entre uma classe mais geral (super-classe) e uma classe mais especializada (sub-classe)
- Exemplos
  - uma conta de poupança é uma conta bancária
  - um círculo é uma elipse (só que tem largura igual a altura)
- Por vezes, abusa-se do conceito: uma classe Pneu deve ser uma sub-classe da classe Círculo? Talvez a relação de agregação has-a seja mais apropriada

Agregação
relação *has-a*

- Objectos de uma classe contêm referências a objectos de outra classe
- Uso de uma variável de instanciação
  - um pneu tem um círculo como fontreira
  - todo o carro tem um pneu (até mais do que 1)

```
class Car extends Vehicle
{
   ...
   private Tire[] tires;
}
```

```
class Tire
{
   ...
   private String rating;
   private Circle boundary;
}
```



notação UML para herança e agregação

Dependência relação *uses*

- Por exemplo, inúmeros programas dependem da classe Scanner para leitura de informação do utilizador
- Agregação é uma forma mais forte de dependência
- Agregação deve ser usada para recordar outro objecto entre chamadas de um método

Agregação e associação

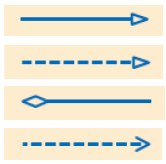
- A associação é um relacionamento mais geral entre classes, sendo usada na fase inicial de design
- Uma classe está associada a outra se for possível passar/*navegar* de objectos de uma classe para objectos de outras

Dado um objecto Bank, podemos "alcançar" objectos Customer

relação de associação

Diagramas UML

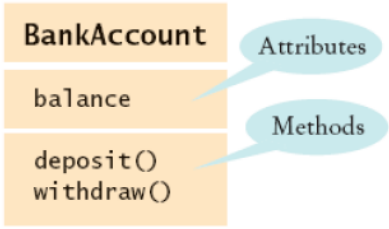
herança
implementação de interface
agregação
dependência



símbolos UML para relações

zero ou mais *
um ou mais 1..*
zero ou um 0..1
um 1

multiplicidade



atributos e métodos num diagrama de classe

relação de agregação com multiplicidade

Processo de desenvolvimento de um programa

1. Recolha de requisitos
2. Cartões CRC para identificar classes, responsabilidades e colaboradores
3. Diagramas UML para registar relações entre classes
4. Programa javadoc para documentar o comportamento dos métodos
5. Implementação do programa

Requisitos do Multibanco

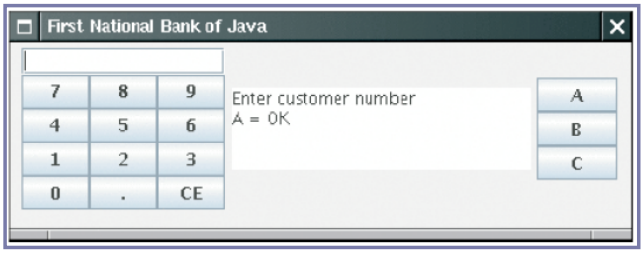
- O Multibanco é utilizado por clientes de um banco
- Um cliente tem
  - uma conta à ordem
  - uma conta-poupança
  - um número de cliente
  - o PIN
- Funcionamento
  - o cliente selecciona a conta
  - a máquina mostra o saldo da conta seleccionada
  - de seguida, o cliente pode levantar ou depositar dinheiro
  - o processo repete-se até o cliente decidir abandonar o programa

Requisitos do Multibanco duas interfaces independentes

- interface baseada em texto. Neste caso, a leitura da entrada é feita através de System.in. Um diálogo típico:

```
Enter account number: 1
Enter PIN: 1234
A=Checking, B=Savings, C=Quit: A
Balance=0.0
A=Deposit, B=Withdrawal, C=Cancel: A
Amount: 1000
A=Checking, B=Savings, C=Quit: C
```

- interface gráfica típica de um Multibanco real. Deve possuir, teclado, ecrã, e três botões: A, B e C. O funcionamento destes depende do estado da máquina



Requisitos do Multibanco interacção

- No início, o cliente deve indicar o número de cliente e de seguida pressionar o botão A. O ecrã mostrará:

```
Enter Customer Number
A = OK
```

- O cliente deve indicar o PIN e de seguida pressionar o botão A. O ecrã mostrará:

```
Enter PIN
A = OK
```

- Segue-se a pesquisa pelo número de cliente e PIN. Se a informação coincide com a de um cliente do banco, a interacção continua. Caso contrário, volta para o ecrã inicial

- Se o cliente for autorizado, o ecrã mostrará:

```
Select Account
A = Checking
B = Savings
C = Exit
```


Requisitos do Multibanco interacção

- se o cliente optar por C, o Multibanco volta para o estado inicial, isto é, solicita ao próximo cliente que indique o número de cliente
- senão, i.e. com a opção A ou B, o Multibanco memoriza a conta seleccionada. O ecrã mostrará:

```
Balance = balance in selected account
Enter amount and select transaction
A = Withdraw
B = Deposit
C = Cancel
```

- se o cliente agora optar por C, o Multibanco passa para o estado anterior
- senão, i.e. o se o cliente optar por A ou B
  - o valor indicado é levantado ou depositado
  - simulação: não é levantado nem depositado nenhum dnheiro
  - o Multibanco passa para o estado anterior

Multibanco
classes

- Os seguintes nomes próprios são passíveis de originarem classes:

```
ATM
User
Keypad
Display
Display message
Button
State
Bank account
Checking account
Savings account
Customer
Customer number
PIN
Bank
```

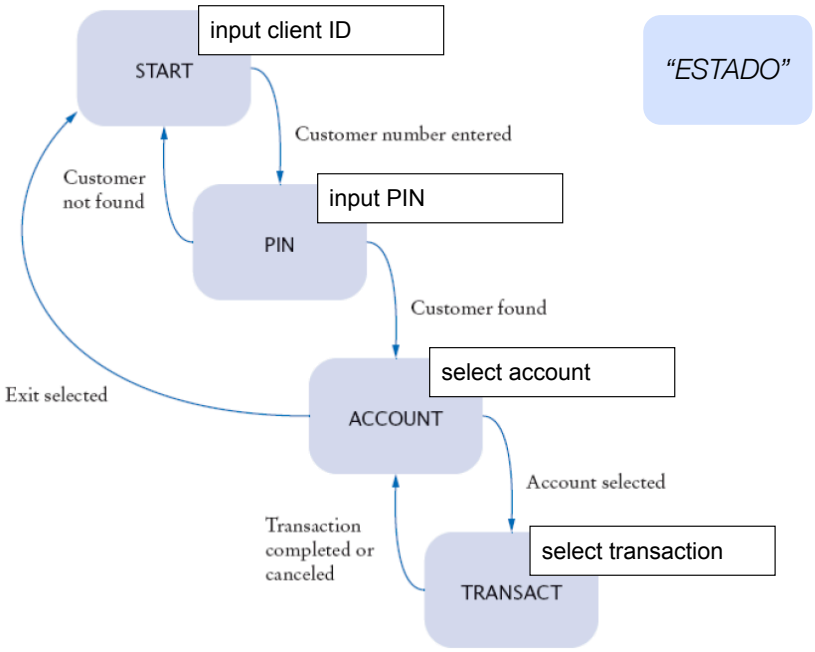

Multibanco
cartões CRC

ATM	
manage state	Customer
select customer	Bank
select account	BankAccount
execute transaction	

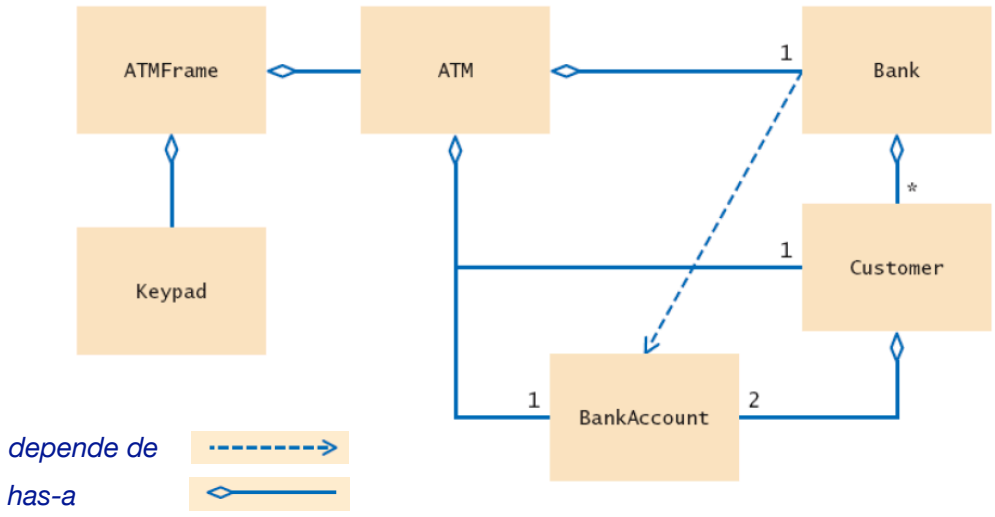
Customer
get accounts
match number and PIN

Bank	
find customer	Customer
read customers	

Multibanco
diagrama de estados



Multibanco
relação entre classes (UML)



relação entre classes

Multibanco implementação

- A implementação pode começar pelas classes que não dependem de outras. É o caso de Keypad e BankAccount. De seguida, implementa-se a classe Customer, a qual depende somente de BankAccount
- A estratégia bottom-up permite testar as classes individualmente
- As classes agregadas no diagrama UML originam variáveis de instanciação

```
private Bank theBank;
```

- Pela descrição dos estados do Multibanco, deduz-se que são necessárias variáveis de instanciação adicionais:

```
private int state;
private Customer currentCustomer;
private BankAccount currentAccount;
```


Multibanco implementação

- A maior parte dos métodos têm implementação implícita/directa. Por exemplo, para o método selectCustomer, a descrição pode ser literamente traduzida em instruções Java

```
/**
   Finds customer in bank. If found sets state to ACCOUNT, else to START.
   (Precondition: state is PIN)
   @param pin the PIN of the current customer
*/
public void selectCustomer(int pin)
{
   assert state == PIN;  // pre-condition (just keep going if state is PIN)
   currentCustomer = theBank.findCustomer(customerNumber, pin);
   state = currentCustomer == null ? START : ACCOUNT;
}
```

Ficheiros .java

ATMTester.java

ATMViewer.java
ATMFrame.java
KeyPad.java

ATM.java
Bank.java (BankATM.java)
Customer.java
BankAccount.java