Polimorfismo com Interfaces

Pacotes em Java

Prof. Gustavo Wagner

(Alterações)

Prof. Tiago Massoni

(Slides Originais)

Desenvolvimento de Sistemas

FATEC-PB

Usando interfaces Java

```
public class BancoInvest {

  private Poupanca [] cb;

  public void inserir(Poupanca c){...}
  public Poupanca procurar(String n){...}
  ...
  public int numPoupancas(){...}
  public double saldoTotal(){...}
}
```

Usando interfaces Java

```
public class BancoSeguros {

  private Apolice [] cb;

  public Apolice procurar(int apolice){...}
  ...
  public int numApolices(){...}
  public double valorTotal(){...}
}
```

Auditor de banco de investimentos

```
class AuditorBI {
  final static double MINIMO = 500.00;

  boolean investigaBanco(BancoInvest b){
    double sm;
    sm = b.saldoTotal()/b.numPoupancas();
    return (sm < MINIMO);
  }
}
```

Auditor de banco de seguros

```
class AuditorBS {
  final static double MINIMO = 500.00;

  boolean investigaBanco(BancoSeguros b){
    double sm;
    sm = b.valorTotal()/b.numApolices();
    return (sm < MINIMO);
  }
}
```

Problema de duplicação

- Duplicação desnecessária de código
- O mesmo auditor deveria ser capaz de investigar qualquer tipo de banco que possua operações para calcular
  - o número de elementos
  - o valor total de todos os elementos
- Quem usa os bancos deveria depender do que é fixo
  - O que varia deve ser escondido do usuário!!

Auditor Genérico

```
class Auditor {
  final static double MINIMO = 500.00;
  private String nome;

  boolean investigaBanco(QualquerBanco b){
    double sm;
    sm = b.saldoTotal()/b.numElementos();
    return (sm < MINIMO);
  }
}
```

Com herança podemos resolver isso...mas seria a melhor opção?
E se tiver um banco que seja tanto banco de seguro quanto de investimento?!

Vamos definir uma interface

- Um "supertipo" definindo serviços em comum
- Herança de interface, não de código;

```
public interface QualquerBanco {
    public double saldoTotal();
    public int numElementos();
}
```

Interfaces

- Todos os métodos são abstratos
  - Não têm implementação!
  - provêem uma interface para serviços
  - são qualificados como **`public`** automaticamente
- não definem atributos
  - apenas podem definir constantes
  - por default, todas as "variáveis" definidas em uma interface são qualificados como **`public`**, **`static`** e **`final`**
- não definem construtores

Subtipos

obrigada "por contrato" a implementar todos os métodos da interface

```
public class BancoSeguros implements
                                QualquerBanco {
 public double saldoTotal(){
     //calcula soma das apolices
 }
 public int numElementos(){
     //calcula numero de apólices
 }
 /*...outros metodos...*/
}

public class BancoInvest implements
                                    QualquerBanco {
    /* ...mesma coisa, com poupancas... */
}
```

Polimorfismo com interfaces

```
BancoInvest bi = new BancoInvest();
BancoSeguros bs = new BancoSeguros();
Auditor a = new Auditor();
...
boolean r = a.auditarBanco(bi);
boolean r' = a.auditarBanco(bs);
...
```

Polimorfismo com interfaces

```
QualquerBanco bi = new BancoInvest();
QualquerBanco bs = new BancoSeguros();
Auditor a = new Auditor();
/* ... */
boolean r = a.auditarBanco(bi);
boolean r' = a.auditarBanco(bs);
/* ... */
```

No sistema em camadas

```
public class RepositorioContasArray {
  private Conta[] arrayContas;
  public void inserir(Conta c){..}
  public void remover(String num){..}
}
public class RepositorioContasListaEnc {
  private List listaContas;
  public void inserir(Conta c){..}
  public void remover(String num){..}
}
public class RepositorioContasBD {
  private Connection conexaoBD;
  public void inserir(Conta c){..}
  public void remover(String num){..}
}
```

Vários repositórios, quando mudar entre eles, afeta a Fachada?

Interface Negócio-dados

Interface negócio-dados

```
public interface IRepositorioContas {
  public void inserir(Conta c);
  public void remover(String num);...
}
```

```
public class RepositorioContasArray
                  implements IRepositorioContas {
  private Conta[] arrayContas;
  public void inserir(Conta c){..}
  public void remover(String num){..}
  ...
}
```

Obrigada "por contrato" a implementar todos os métodos da interface

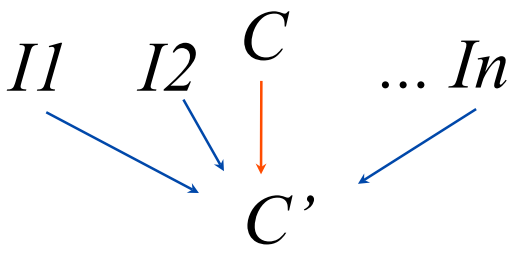
Polimorfismo com interfaces

```
public class FachadaBanco { ...
  private IRepositorioContas repContas;
  ...
  repContas.inserir(novaConta);
  ...
  repContas.remover(numero); ...
}
```

Definição de Classes: Forma Geral

Interfaces permitem simular herança múltipla

```
class C'
  extends C
  implements I1, I2, ..., In {
    /* ... */
  }
```



Módulos em Java

- Classes
  - agrupam definições de métodos, atributos, construtores, etc.
  - definem tipos (referência)
- Pacotes
  - agrupam classes relacionadas
  - estruturam sistemas de grande porte, facilitando a localização das classes
  - há mais classes do que pacotes

Pacotes e diretórios

- As classes de um pacote são definidas em arquivos com o mesmo cabeçalho:

```
package nomeDoPacote;
```

- Cada pacote é associado a um diretório do sistema operacional:
  - os arquivos `.class` das classes do pacote são colocados neste diretório
  - é recomendável que o código fonte das classes do pacote também esteja neste diretório

Nomeando pacotes

- O nome de um pacote é parte do nome do seu diretório associado: o pacote

  exemplos.banco

  deve estar no diretório

  /home/gustavo/exemplos/banco

  assumindo que o compilador Java foi informado para procurar classes em

  /home/gustavo/

  Este é o CLASSPATH!!

- No Eclipse, temos que criar pacotes e colocar as classes no mesmo

Pacotes e visibilidade de declarações

- **`public`**
- **`private`**
- **`protected`**
  - atributos, métodos e construtores
  - declaração utilizada no pacote onde ela é introduzida e nas subclasses da classe
- Ausência de modificador
  - declaração utilizada no pacote onde ela é introduzida

Importação de pacotes

- Importando definição específica

```
package modelo;
import bib.ClasseImportada;
public class Exemplo{ /*...*/ }
```

- Importando todas as definições públicas

```
package modelo;
import bib.*;
public class Exemplo{ /*...*/ }
```

Importação de pacotes

- Tanto **`ClasseImportada`** quanto **`bib.ClasseImportada`** podem ser usados no corpo das classes
  - Usando **`bib.ClasseImportada`** não precisa de import
- **`ClasseImportada`** deve ser declarada como public

Pacotes da biblioteca de Java

- Default: threads e manipulação de strings (**java.lang**)
- Acesso a Internet e WWW (**java.net**)
- Definição de interfaces gráficas (**javax.swing**)
- Suporte a objetos distribuídos (**java.rmi**)
- Interface com Banco de Dados (**java.sql**)
- Arquivos (**java.io**), utilitários de propósito geral (**java.util**)

Aula Prática
Pacotes
Interfaces

Objetivos da prática

- Modificar sistema bancário em camadas para dividí-lo em pacotes
- Modificá-lo também para incorporar interfaces na separação entre Fachada e Repositórios
  - Interface negócio-dados

Pacotes e camadas

- Estruture o projeto de camadas usando pacotes
  - Um pacote para interface "gráfica"
    - gui
  - Um pacote para Fachada
    - fachada
  - Um pacote para cada subsistema (classe básica e repositórios)
    - contas, clientes
  - Um pacote para classes utilitárias (classe Console)

Interfaces negócio-dados

- Adaptar banco em camadas para utilizar interfaces negócio-dados
  - Primeiro, fechar com RepositorioClientesArray
  - Depois, RepositorioContasArray