Orientação a Objetos e Java

Sérgio Soares
sergio@dsc.upe.br

Interfaces

Objetivo

Depois desta aula você será capaz de desenvolver sistemas mais reusáveis e extensíveis, através da utilização de interfaces que permitem relacionar classes com implementações diferentes mas oferecendo os mesmos tipos de serviços.

Interfaces

Leitura prévia essencial

• Páginas 389 a 395 do livro *Java: how to program* (de Harvey e Paul Deitel)

Auditor de Banco

```
public class AuditorB {
  private final static double MINIMO = 500.00;
  private String nome;
  /* ... */
  public boolean auditarBanco(Banco banco) {
    double saldoTotal, saldoMedio;
    int numeroContas;
    saldoTotal = banco.saldoTotal()
    numeroContas = banco.numeroContas();
    saldoMedio = saldoTotal/numeroContas;
    return (saldoMedio < MINIMO);
  }
}
```

Auditor de Banco Modular

```
public class AuditorBM {
  private final static double MINIMO = 500.00;
  private String nome;
  /* ... */
  public boolean auditarBanco(BancoModular banco){
    double saldoTotal, saldoMedio;
    int numeroContas;
    saldoTotal = banco.saldoTotal()
    numeroContas = banco.numeroContas();
    saldoMedio = saldoTotal/numeroContas;
    return (saldoMedio < MINIMO);
  }
}
```

Problema

- Duplicação desnecessária de código
- O mesmo auditor deveria ser capaz de investigar qualquer tipo de banco *que possua operações para calcular*
  - o número de contas, e
  - o saldo total de todas as contas.

Auditor Genérico

```
public class AuditorGenerico {
  private final static double MINIMO = 500.00;
  private String nome;
  /* ... */
  public boolean auditarBanco(QualquerBanco banco){
    double saldoTotal, saldoMedio;
    int numeroContas;
    saldoTotal = banco.saldoTotal()
    numeroContas = banco.numeroContas();
    saldoMedio = saldoTotal/numeroContas;
    return (saldoMedio < MINIMO);
  }
}
```

Definindo Interfaces

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

Interfaces

- Caso especial de classes abstratas...
  - todos os métodos são abstratos
    - provêem uma interface para serviços e comportamentos
    - são qualificados como **`public`** por default
  - não definem atributos
    - definem constantes
    - por default todos os "atributos" definidos em uma interface são qualificados como **`public`**, **`static`** e **`final`**
  - não definem construtores

Interfaces

- Não pode-se criar objetos
- Definem tipo de forma abstrata, apenas indicando a assinatura dos métodos
- Os métodos são implementados pelos subtipos (subclasses)
- Mecanismo de projeto
  - podemos projetar sistemas utilizando interfaces
  - projetar serviços sem se preocupar com a sua implementação (abstração)

Subtipos sem Herança de Código

```
public class Banco
            implements QualquerBanco {
    /* ... */
}

public class BancoModular
            implements QualquerBanco {
    /* ... */
}
```

`implements`

- ***`classe`*** **`implements`**
  `interface1, interface2, ...`
- ***`subtipo`*** **`implements`**
  `supertipo1, supertipo2, ...`
- Múltiplos supertipos:
  - uma classe pode implementar mais de uma interface (contraste com classes abstratas...)

implements

- Classe que implementa uma interface deve *definir* os métodos da interface:
  - classes concretas têm que implementar os métodos
  - classes abstratas podem simplesmente conter métodos abstratos correspondentes aos métodos da interface

Usando Auditores

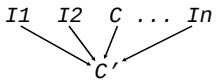
```
Banco b = new Banco();
BancoModular bm = new BancoModular();
Auditor a = new Auditor();
/* ... */
boolean r = a.auditarBanco(b);
boolean r' = a.auditarBanco(bm);
/* ... */
```

Interfaces e Reusabilidade

- Evita duplicação de código através da definição de um tipo genérico, tendo como subtipos várias classes não relacionadas
- Tipo genérico pode agrupar objetos de várias classes definidas independentemente, sem compartilhar código via herança, tendo implementações totalmente diferentes
- Classes podem até ter mesma semântica...

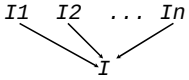
Definição de Classes: Forma Geral

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



Subtipos com Herança Múltipla de Assinatura

```
interface I
  extends I1, I2, ..., In {
  /*... assinaturas de novos
        métodos ...
  */
}
```



O que usar? Quando?

Classes (abstratas)

- Agrupa objetos com implementações compartilhadas
- Define novas classes através de herança (simples) de código
- Só uma pode ser supertipo de outra classe

Interfaces

- Agrupa objetos com implementações diferentes
- Define novas interfaces através de herança (múltipla) de assinaturas
- Várias podem ser supertipo do mesmo tipo

Cadastro de Contas: Parametrização

```
public class CadastroContas {
    private RepositorioContas contas;
    public CadastroContas(RepositorioContas r) {
        if (r != null) contas = r;
    }
    /* ... */
}
```

A estrutura para armazenamento das contas é fornecida na inicialização do cadastro, e pode depois ser trocada!

Repositório: Definição

```
public interface RepositorioContas {
    void inserir(Conta conta);
    void atualizar(Conta conta);
    Conta procurar(String numero);
    void remover(String numero);
    boolean existe(String numero);
}
```

Repositório: Implementações

```
public class RepositorioContasArray
        implements RepositorioContas {...}

public class RepositorioContasLista
        implements RepositorioContas {...}

public class RepositorioContasVector
        implements RepositorioContas {...}

public class RepositorioContasBDR
        implements RepositorioContas {...}
```

Cadastro de Contas: Parametrização

```
public void cadastrar(Conta conta) {
    if (conta != null) {
        String numero = conta.getNumero();
        if (!contas.existe(numero)) {
            contas.inserir(conta);
        }
    }
}
```

Cadastro de Contas: Parametrização

```
public void debitar(String numero,
                    double valor){
    Conta conta;
    conta = contas.procurar(numero);
    if (conta != null) {
        conta.debitar(val);
    }
}
```

Exercícios

- Que outros mecanismos de Java poderiam ter sido usados para definir o tipo `RepositorioContas`?
- Explique como o mecanismo de interfaces favorece reusabilidade e extensibilidade. Justifique.

Interfaces

Resumo

- Cláusula `interface`
- Cláusula `implements`
- Herança de código versus herança de assinaturas
- Interfaces e parametrização de sistemas

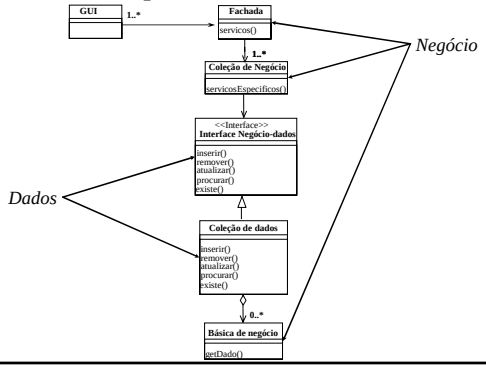
Interfaces

Leitura adicional

- Capítulo 7 do livro *Thinking in Java* (de Bruce Eckel)
- Seção 6.4 do livro *A Programmer's Guide to Java Certification* (de Khalid Mughal e Rolf Rasmussen)

Vendo o código como um bolo... com várias camadas!

Arquitetura de software



Classes Básicas de Negócio

```
public class Conta {
  private double saldo;
  private String numero;
  private Cliente correntista;
  ...
  public void creditar(double valor) {
    saldo = saldo + valor;
  }
}
```

`Cliente, Livro, Animal, Veiculo`

Interfaces Negócio-Dados

```
public interface RepositorioContas {
  public void inserir(Conta conta);
  public void atualizar(Conta conta);
  public void remover(String numero);
  public Conta procurar(String numero);
  public RepositorioContas procurar(Conta conta);
  public boolean existe(String numero);
  public IteratorContas getIterator();
}
```

`RepositorioClientes, RepositorioLivros, RepositorioAnimais, RepositorioVeiculos`

Interface Iterator

```
public interface IteratorContas {
  public boolean hasNext();
  public Conta next();
}
```

IteratorClientes, IteratorLivros,
IteratorAnimais, IteratorVeiculos

Coleção de dados iterável

```
public class IteratorContasArray
               implements IteratorContas {
  private Conta[] contas;
  private int proximo; ...
  public IteratorContasArray(Conta[] conta) {...}
  public void add(Conta conta) {...}
  public boolean hasNext() {...}
  public Conta next() {...}
}
```

Classes Coleção de Dados

```
public class RepositorioContasArray
    implements RepositorioContas {
  private Conta[] contas;
  private int indice;
  public void inserir(Conta conta) {
    contas[indice] = conta;
    indice = indice + 1;
  } ...
}
```

RepositorioContasArquivo, RepositorioContasLista
RepositorioContasBDR, RepositorioContasBDOO

Classes Coleção de Negócio

```
public class CadastroContas {
  private RepositorioContas contas;
  public CadastroContas(RepositorioContas rep) {
    contas = rep;
  }
  public void cadastrar(Conta conta) {
    if (!contas.existe(conta.getNumero())) {
      contas.inserir(conta);
    } else ...
  } ...
}
```

CadastroClientes, CadastroLivros,
CadastroAnimais, CadastroVeiculos

Classe Fachada

```
public class Banco {
  private CadastroContas contas;
  private CadastroClientes clientes;
  ...
  public void cadastrar(Conta conta) {
    Cliente c = conta.getCorrentista();
    if (clientes.existe(c.getCodigo()) {
      contas.cadastrar(conta);
    } else ...
  }
}
```

Livraria, Zoo, Locadora