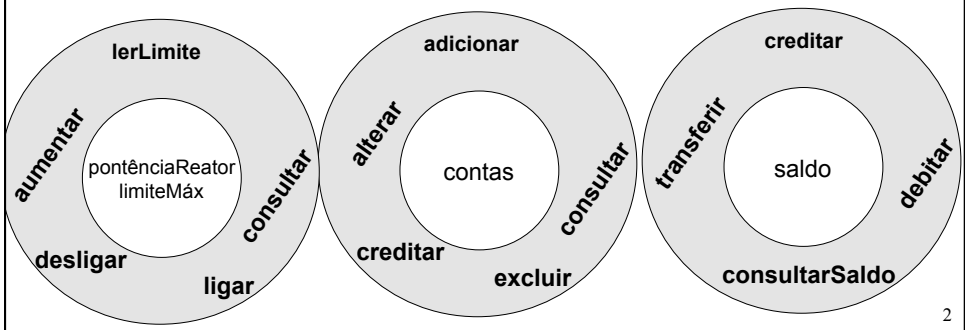
Programação Orientada a Objetos

Encapsulamento

Franciéric Alves – francieric [at] gmail.com

Encapsulamento

- Um objeto deve “encapsular” todo o estado e expõe apenas parte do seu comportamento;
- Em outras palavras, nem todo elemento de uma classe deve ser visível (e alterável)
  - Preferencialmente, temos de visibilidade pública a atributos com visibilidade privada ou protegida



2

Encapsulamento

- Nos três itens anteriores, é importante ocultar o estado pois:
  - Nenhum dos atributos mencionados deve ser manipulado diretamente sem validações
  - Apenas métodos que tenham codificações que mantenham integridade devem ter acesso aos dados
  - A leitura dos dados pode ser liberada, mas a escrita deve ser criteriosa

3

Exemplo

```
class Conta {
   String numero;
   double saldo;
   double limite;

   void saca(double valor) {
      if (valor <= this.saldo)
         this.saldo = this.saldo - valor;
   }
}
```

Usar o método saca, previne que o saldo fique em um estado inconsistente

No entanto os atributos ainda são acessíveis!

4

Exemplo de violação

```
class Testa {
   public static void main(String args[]) {
      Conta c = new Conta();
      c.limite = 100;
      c.saldo = -2000;
   }
}
```

O acesso direto ao atributo não permite validação

5

Solução

- Obrigar ao utilizador a chamar o método saca e não permitir o acesso direto ao atributo.
- Para isso, basta declarar os atributos usando a palavra chave **private** e os métodos de acesso como **public**:

```
class Conta {
   private String numero;
   private double saldo;
   private double limite;

   public void saca(double valor) {
      if (valor <= this.saldo)
         this.saldo = this.saldo - valor;
   }
}
```

6

Modificadores de acesso e visibilidade

- Utilizados para garantir o encapsulamento em Java
- Podem ser aplicados em: classes e membros (atributos, métodos e construtores) individualmente
- Modificadores existentes:
  - **public:** É visível em qualquer lugar.
  - **protected:** Só é visível na mesma classe e em suas subclasses.
  - ***package:*** é o padrão. Só é visível em classes do mesmo pacote. A ausência de modificador o torna package.
  - **private** Só é visível dentro da mesma classe.

7

Ainda sobre Encapsulamento e modificadores

- Os modificadores também podem ser usados para modificar o acesso a métodos.
  - Quando existe um método apenas auxiliar à própria classe, e não queremos que outras pessoas o usem.
- Há também o modificador **public**, que permite a todos acessarem um determinado atributo ou método.
- É muito comum que atributos sejam **private**, e quase todos os métodos sejam **public**.
  - Assim, toda conversa de um objeto com outro é feita por troca de mensagens, isso é, acessando seus métodos.

8

Padrão JavaBean

- Um JavaBean é um componente reutilizável que tem como finalidade representar um modelo de dados.
  - Define convenções para que atributos de dados sejam tratados como "propriedades".
- Características:
  - Construtor público sem argumentos
  - Atributos de dados **private**.
  - Métodos de acesso (acessors / getter) e/ou de alteração (mutators / setter)
    - Para cada atributo usa-se a convenção **get*Propriedade*()** ou **is*Propriedade*()** (boolean) e **set*Propriedade*()**.

9

Exemplo de JavaBean

```
public class Loja {
   private String nome;

   public Loja() {
      // Construtor vazio
   }

   public Loja(String nome) {
      this.nome = nome;
   }

   public void setNome(String nome) {
      this.nome = nome;
   }

   public String getNome() {
      return this.nome;
   }
}
```

10

Exemplo de leitura (get) e escrita(set)

```
public class Conta {
    private String titular;
    private double saldo;

    public Conta(String titular, double saldoInicial) {
        this.titular= titular;
        //Nota: deve-se validar o saldoInicial antes
        this.saldo = saldoInicial;
    }
    public String getTitular() {
        return titular;
    }
    public void setTitular(String titular) {
        this.titular = titular;
    }
    public double getSaldo() {
        return saldo;
    }
    //demais métodos: creditar, debitar...
}
```

o setSaldo não existe por segurança

11

Vantagens e consequências do encapsulamento

- Vantagens:
  - Diminuindo a visibilidade de uma classe, ocultamos detalhes de implementação
  - Facilitam-se alterações na aplicação, pois uma regra só precisa ser modificada em um único lugar.
  - Facilita o aprendizado, pois o mínimo de funcionalidades é exposta
- Consequência direta:
  - diminui-se o acoplamento e tornamos nosso código mais utilizável e fácil de manter

12

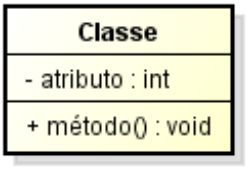
Representação UML

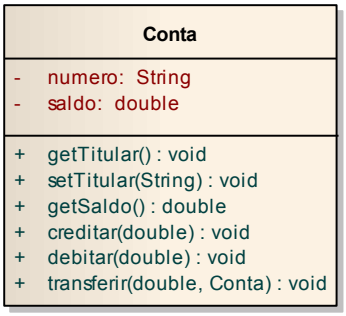
- Em UML, a visibilidade é definida pelos caracteres abaixo:

  + Publica

  # protegida

  - privada





13