

Aula 8

Professores:

*Carlos Bazílio
Isabel Rosseti*

Pacotes e Controle de Acesso

Conteúdo:

- revisão da aula anterior
- motivação
- pacotes
- controle de acesso
- exercício

Revisão da aula anterior

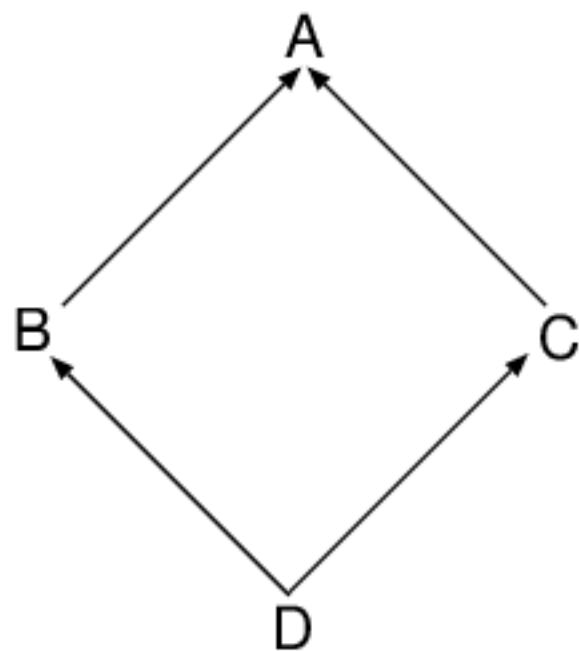
→ classe abstrata

- classe que necessariamente deve ser estendida
- não pode ser instanciada pois sua funcionalidade está incompleta
- podemos declarar uma classe abstrata usando o modificador **abstract**

Revisão da aula anterior

→ herança múltipla

- problema: herdar duas vezes de uma mesma classe



Revisão da aula anterior

→ Java:

- não permite herança múltipla com herança de código
- implementa o conceito de *interface*
- uma classe *estende* zero ou uma outra classe
- uma classe *implementa* zero ou mais interfaces
- para implementar uma interface em uma classe, usamos a palavra **implements**

Motivação

- como é possível criar módulos em Java?
 - solução: classes e pacotes
- como é possível proteger atributos e métodos de acessos indevidos em Java?
 - solução: controle de acesso

Modularidade em Java: pacotes

→ recursos que ajudam a modularidade:

- o uso de classes e
- o uso de pacotes

→ um pacote é um conjunto de classes e de outros pacotes

→ pacotes permitem:

- a criação de espaços de nomes e
- mecanismos de controle de acesso

Pacotes: espaços de nomes

- pacotes possuem nomes
- nome do pacote: qualifica os nomes de todas as classes e outros pacotes que o compõem
- exemplo: classe **Math**

```
int a = java.lang.Math.abs(-10); // a = 10;
```

OBS: Na verdade, a classe Math é constante.

Implementação de pacotes

- pacotes são tipicamente implementados como diretórios
- os arquivos das classes pertencentes ao pacote devem ficar em seu diretório
- hierarquias de pacotes são construídas através de hierarquias de diretórios

"Empacotando" uma classe

→ para declararmos uma classe como pertencente a um pacote devemos:

- declará-la em um arquivo dentro do diretório que representa o pacote;
- declarar, na primeira linha do arquivo, que a classe pertence ao pacote.

Importação de pacotes

- podemos usar só o nome de uma classe que pertença a um pacote se *importarmos* a classe
- a importação de uma classe pode ser feita no início do arquivo, após a declaração do pacote
- as classes do pacote padrão **java.lang** não precisam ser importadas
 - exemplo: **Math**

Exemplo de arquivo

```
package datatypes;      // Stack pertence a datatypes
import javax.swing.*;   // Importa todas as classes

/*
 a partir desse ponto, posso usar o nome de
 qualquer classe que pertença ao pacote
 javax.swing.
 */

public class Stack { // Stack é exportada.
    ...
}
```

Exemplo de arquivo

```
public class Stack {  
    int[] data;  
    int top_index;  
    Stack(int size) {  
        data = new int[size];  
        top_index = -1;  
    }  
    boolean isEmpty() {  
        return (top_index < 0);  
    }  
    void push(int n) { data[++top_index] = n; }  
    int pop() { return data[top_index--]; }  
    int top() { return data[top_index]; }  
}
```

Encapsulamento

- na classe Stack, nós encapsulamos a definição de pilha que desejávamos
- porém, por falta de *controle de acesso*, é possível forçar situações nas quais a pilha não se comporta como desejado:

```
Stack s = new Stack(10);
s.push(6);
s.top_index = -1;
System.out.println(s.isEmpty()); // true!
```

Controle de acesso

- linguagens OO disponibilizam formas de controlar o acesso aos membros de uma classe
- no mínimo, devemos diferenciar entre o que é *público* e o que é *privado*:
 - membros públicos podem ser acessados indiscriminadamente
 - membros privados só podem ser acessados pela própria classe

Redefinição de Stack

```
public class Stack {  
    private int[] data;  
    private int top_index;  
    public Stack(int size) {  
        data = new int[size];  
        top_index = -1;  
    }  
    public boolean isEmpty() {  
        return (top_index < 0);  
    }  
    public void push(int n) { data[++top_index] = n; }  
    public int pop() { return data[top_index--]; }  
    public int top() { return data[top_index]; }  
}
```

Exemplo de controle de acesso

→ agora o exemplo anterior não pode mais ser feito pois teremos um erro de compilação:

```
Stack s = new Stack(10);
s.push(6);
s.top_index = -1; // ERRO! compilação pára aqui!
System.out.println(s.isEmpty());
```

Pacotes: controle de acesso

- além de membros públicos e privados, temos também membros de pacote
- membro de pacote: só pode ser acessado por classes declaradas no mesmo pacote da classe que declara esse membro
- quando omitimos o modificador de controle de acesso, estamos informando que o membro é de pacote.

Mais sobre visibilidade

→ pelo que foi estudado até agora:

- membros públicos podem ser acessados indiscriminadamente
- membros privados só podem ser acessados pela própria classe e
- membros de pacote são acessados por classes declaradas no mesmo pacote da classe

Mais sobre visibilidade

- às vezes precisamos de um controle de acesso intermediário:
 - um membro que seja acessado somente nas sub-classes e nas classes declaradas no mesmo pacote
- linguagens OO tipicamente dão suporte a esse tipo de acesso
- modificador de controle de acesso **protected**

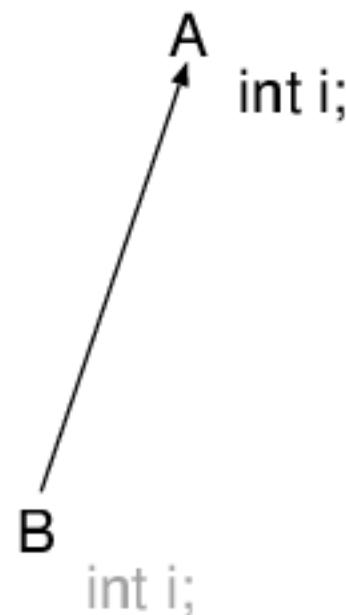
Resumo de visibilidade em Java

→ Resumindo todos os tipos de visibilidade:

- **private**: membros que são acessados somente pela própria classe
- **protected**: membros que são acessados pelas suas sub-classes e pelas classes do pacote
- **public**: membros são acessados por qualquer classe
- sem modificador ou *default* : membros que são acessados pelas classes do pacote

Visibilidade de membros

Pacote P1



OBS: Na verdade, o atributo i é um membro de pacote.

Visibilidade de membros

Pacote P1

A

```
int i;  
public int j;
```

B

```
int i;  
public int j;
```

Visibilidade de membros

Pacote P1

A

```
int i;  
public int j;  
protected int k;
```

B

```
int i;  
public int j;  
protected int k;
```

Visibilidade de membros

Pacote P1

A

```
int i;  
public int j;  
protected int k;  
private int l;
```

B

```
int i;  
public int j;  
protected int k;
```

Visibilidade de membros

Pacote P1

A
int i;
public int j;
protected int k;
private int l;

B
int i;
public int j;
protected int k;

Pacote P2

P1.A
public int j;
protected int k;

Exercício: enunciado

Altere a sua solução do exercício do Banco (uma possível solução para este exercício foi mostrada no final da aula 5) para que as classes **Banco**, **Conta** e **Poupança** façam parte de um pacote **SistemaBancario**.

Nesta solução alterada deve ser feita uma distinção entre membros privados e públicos. Geralmente, atributos são membros privados. Já os métodos são membros públicos. Se for necessário modificar algum atributo, deve-se inserir na classe um método público responsável por alterar o atributo correspondente.

Crie o seu programa de teste em uma classe **Teste** que não faça parte desse pacote.

Exercício: enunciado

Projete e implemente um sistema que modele um banco. Seu projeto deve permitir a criação de vários bancos e várias contas corrente e poupanças para cada banco. Para um dado banco deve ser possível: obter seu nome, obter seu código, criar uma nova conta, criar uma nova poupança e obter uma conta a partir de um código.

Para cada conta corrente criada deve ser possível: obter o nome do correntista, obter o banco a qual a conta pertence, obter seu saldo, fazer uma aplicação e efetuar um débito.

Uma conta poupança permite fazer tudo o que se pode fazer com uma conta corrente. No entanto, diferentemente de uma conta corrente, não se pode fazer uma retirada de poupança que torne seu saldo negativo. Quando essa situação ocorre, a poupança não faz a retirada e escreve uma mensagem na console avisando.

Exercício: enunciado (cont.)

Faça com que cada banco tenha um código próprio, o mesmo vale para as contas. Permita que contas e poupanças de bancos diferentes tenham o mesmo número. Escreva um programa de teste que crie um banco e, para este banco, crie uma conta corrente e uma poupança. Efetue as operações possíveis para todas as classes existentes.

Exercício: solução (classe Banco)

```

package SistemaBancario;
public class Banco{
    private static int prox_banco = 1;
    private final int MAX_CONTAS = 10;
    private String nome;
    private int codigo, prox_conta;
    private int ind_array;
    private Conta[] contas;

    public Banco(String n){
        nome = n;
        codigo = prox_banco++;
        prox_conta = 1;
        contas = new Conta[MAX_CONTAS];
        ind_array = 0;
    }

    public int pegaCodB(){
        return codigo;
    }

    public String pegaNomeB(){
        return nome;
    }

    public void altNomeB(String n){
        nome = n;
    }

    public Conta buscaConta(int c){
        int i;
        for(i=0; i<ind_array; i++)
            if(contas[i].pegaCodigo() == c)
                return contas[i];
        return null;
    }
}

```

OBS: Na verdade, na aula 3 foram criadas as classes banco e conta corrente.
 Já na aula 5, foi desenvolvida a classe poupança e foram modificadas as classes banco e conta corrente.

Exercício: solução (classe Banco)

```
public Conta criaConta  
(String nome){  
    Conta c;  
    if(ind_array==MAX_CONTAS)  
        c=null;  
    else{  
        c = new Conta(nome, prox_conta++,  
                       this);  
  
        contas[ind_array++] = c;  
    }  
    return c;  
}
```

```
public Poupanca criaPoupanca  
(String n){  
    Poupanca c;  
    if(ind_array==MAX_CONTAS)  
        c=null;  
    else{  
        c = new Poupanca(nome,  
                           prox_conta++, this);  
  
        contas[ind_array++] = c;  
    }  
    return c;  
}
```

Exercício: solução (classe Conta)

```

package SistemaBancario;
public class Conta{
    private String nome;
    private int codigo;
    private Banco banco;
    private float saldo;

    public Conta(String n, int c,
                Banco b){
        nome = n;
        codigo = c;
        banco = b;
        saldo = 0F;
    }

    public Banco pegaBanco(){
        return banco;
    }

    public void altBanco(Banco b){
        banco = b;
    }
}

```

```

    public String pegaNome(){
        return nome;
    }

    public void altNome(String n){
        nome = n;
    }

    public int pegaCodigo(){
        return codigo;
    }

    public void altCodigo(int c){
        codigo = c;
    }
}

```

Exercício: solução (classe Conta)

```
public float pegaSaldo(){
    return saldo;
}

public void aplica(float soma){
    saldo += soma;
}

public void retira(float soma){
    saldo -= soma;
}

public String toString(){
    return "Conta corrente";
}
```

OBS: Na verdade, os métodos foram descritos na aula 5.

Exercício: solução (classe Poupanca)

```
package SistemaBancario;
public class Poupanca extends Conta{
    public Poupanca(String n, int c, Banco b){
        super (n, c, b);
    }

    public void retira(float soma){
        if(pegSaldo()-soma<0)
            System.out.println("A poupança não pode ter saldo negativo");
        else
            super.retira(soma);
    }

    public String toString(){
        return "Poupanca";
    }
}
```

Exercício: solução (classe Teste)

```
import SistemaBancario.*;
class Teste{
    public static void main(String[] args){
        Banco itau = new Banco("Itau");
        System.out.println(itau.pegaCodB());
        System.out.println(itau.pegaNomeB());
        Conta maria = itau.criaConta("Maria");
        System.out.println(maria);
        Conta b = itau.buscaConta(1);
        b = itau.buscaConta(2);
        Poupanca jose = itau.criaPoupanca("Jose");
        System.out.println(jose + " " + jose.pegaNome());
        System.out.println(jose + " " + jose.pegaCodigo());
        System.out.println(jose + " " + jose.pegaSaldo());
        jose.aplica(100.0F);
        System.out.println(jose + " " + jose.pegaSaldo());
        jose.retira(150.0F);
        System.out.println(jose + " " + jose.pegaSaldo());
        jose.retira(30.5F);
        System.out.println(jose + " " + jose.pegaSaldo());
    }
}
```