Aula 5

Professores:

Carlos Bazílio
Isabel Rosseti

Herança e Sobrecarga em Java

Conteúdo:

- revisão da aula anterior
- motivação
- herança
- sobrecarga
- exercício

Fundação CECIERJ
Consórcio cederj

Revisão da aula anterior

- conceitos:
  - classificação/generalização
  - especialização
  - herança

Motivação

- é possível reusar as classes existentes para melhor adaptá-las a novas situações:
  - reutilizar ou alterar métodos;
  - adicionar métodos;
  - adicionar novos atributos.

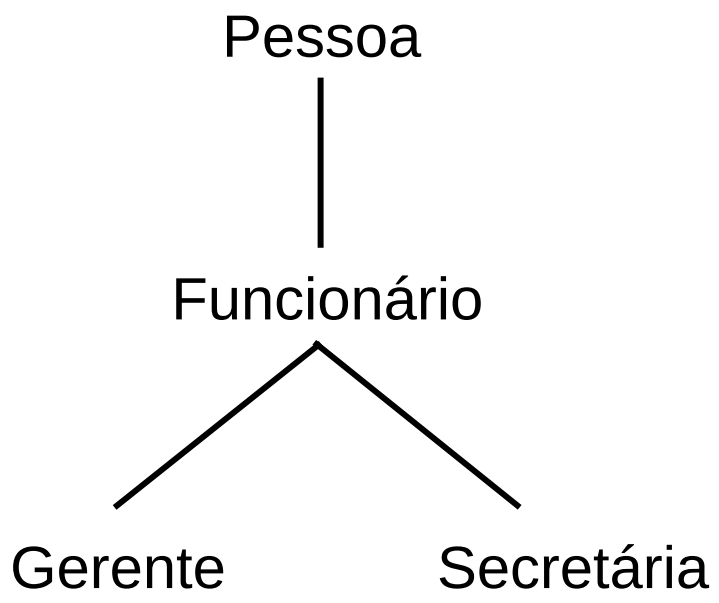
Motivação

- herança: técnica utilizada para criar novas classes a partir das existentes

- esta técnica é essencial na programação em Java

Herança

- classes podem ser compostas em hierarquias, através do uso de *herança*
- quando uma classe herda de outra, diz-se que ela a *estende* ou ela a *especializa*



Herança em Java

- quando uma classe *B* herda de *A*, diz-se que *B* é a *sub-classe* e *estende A*, a *superclasse*
- uma classe Java estende apenas uma outra classe
- esta restrição tem o nome de *herança simples*
- para criar uma sub-classe, usamos a palavra reservada **extends**

Exemplo de herança

- podemos criar uma classe que represente um pixel a partir da classe **Point**

- afinal, um pixel é um ponto colorido

```
class Point {
  int x, y;
  Point(int x0, int y0) {
    x = x0;
    y = y0;
  }
  void move(int dx, int dy){
    x += dx;
    y += dy;
  }
}
class Pixel extends Point {
  int color;
  Pixel(int x, int y, int c) {
      super(x, y);
      color = c;
    }
  }
```

super

- primeira coisa que o construtor de **Pixel** faz é chamar o construtor de **Point**, usando a palavra reservada **super**

```
class Pixel extends Point {
   int color;
   Pixel(int x, int y, int c) {
     super(x, y);
     color = c;
   }
 }
```

super

- isso é necessário pois **Pixel** é uma extensão de **Point**

- ou seja, ela deve inicializar sua parte **Point** antes de inicializar sua parte estendida

super

- se nós não chamássemos o construtor da superclasse explicitamente, Java faria uma chamada ao construtor padrão da superclasse automaticamente

- construtor padrão: `<nome_classe>( ){ }`
  - exemplo: `Point( ){ }`

Criação de objetos

- a classe **Pixel** herda a classe **Point**:
  - **Pixel** passa a ter tanto os atributos quanto os métodos de **Point**

```
Pixel px = new Pixel(1,2,0); // Pixel de cor 0
px.move(1,0);                // px está em (2,2)
```

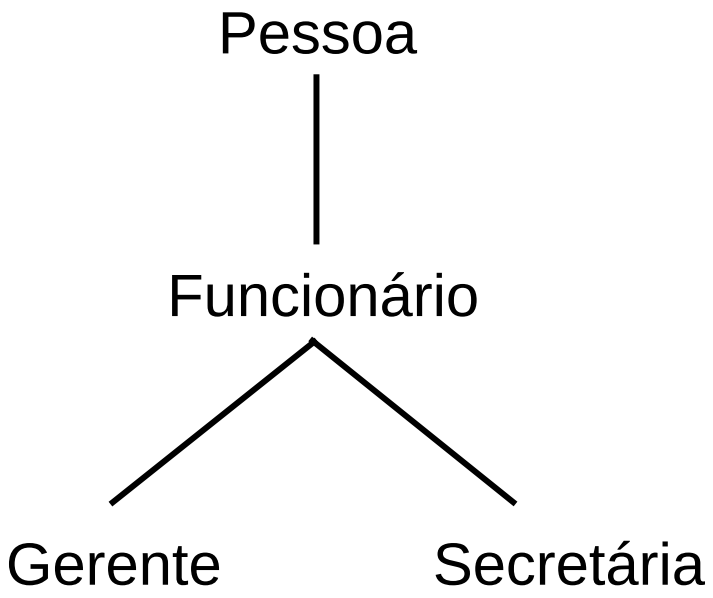
Árvore x Floresta

- as linguagens OO podem adotar um modelo de hierarquia em *árvore* ou em *floresta*

Árvore

- uma única hierarquia compreende todas as classes existentes

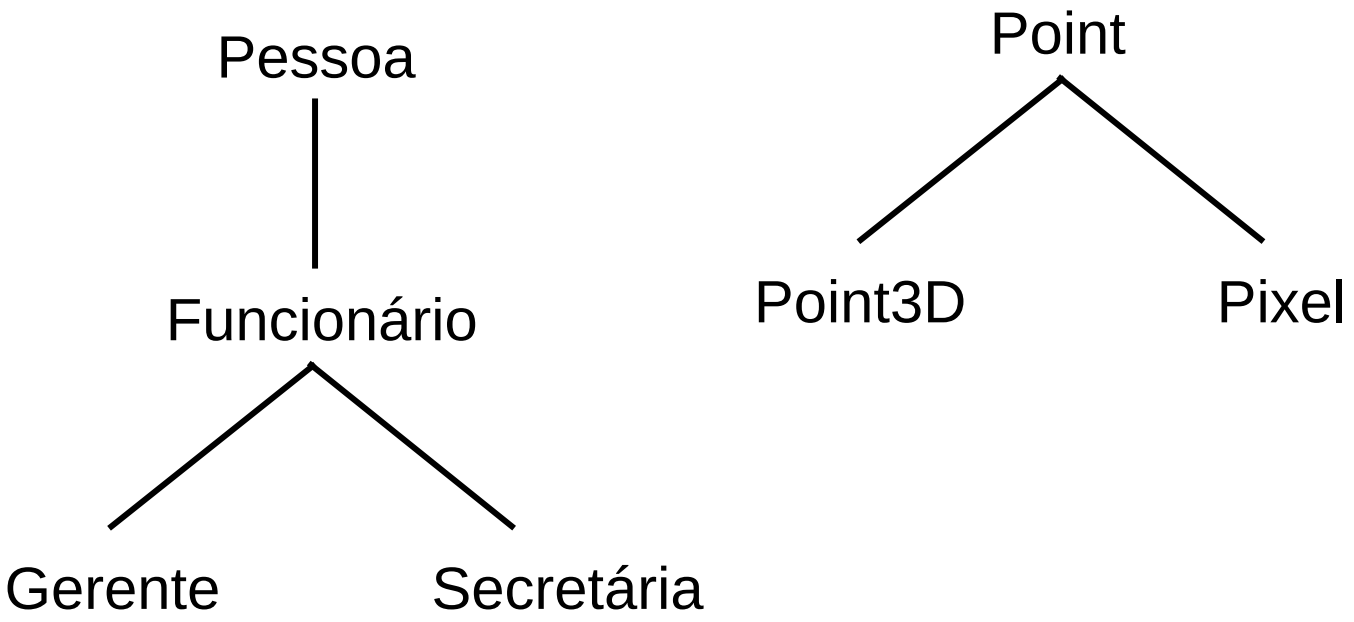
- isto é, existe uma superclasse comum a todas as classes



Floresta

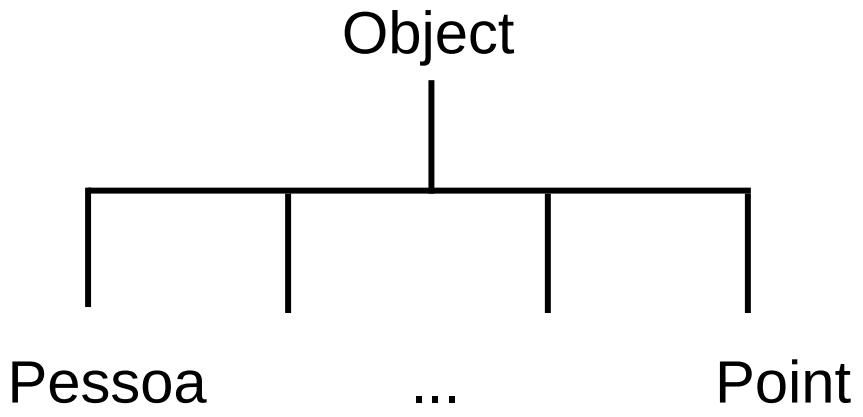
- pode haver diversas árvores de hierarquia que não se relacionam

- isto é, não existe uma superclasse comum a todas as classes



Modelo de Java

- Java adota o modelo de árvore
- **Object**: é a raiz da hierarquia de classes à qual todas as classes pertencem
- quando não declaramos que uma classe estende outra, ela, implicitamente, estende **Object**



Superclasse comum

- uma vantagem de ter uma superclasse comum é ter uma funcionalidade comum a todos os objetos:
  - **Object** define o método **toString** que retorna um texto descritivo do objeto
  - **Object** define o método **finalize** usado na destruição de um objeto

Sobrecarga

- um recurso usual em programação OO é o uso de *sobrecarga* de métodos
- sobrecarregar um método significa prover mais de uma versão de um mesmo método
- as versões devem possuir listas de parâmetros diferentes:
  - nos tipos dos parâmetros ou
  - no número desses parâmetros

Sobrecarga de construtores

- ao criarmos o construtor da classe **Point** para inicializar o ponto em uma dada posição, perdemos o construtor padrão
- construtor padrão: deixa o ponto na posição (0,0).
- nós podemos voltar a ter esse construtor usando sobrecarga

Sobrecarga de construtores: declaração

```
class Point {
  int x = 0;
  int y = 0;
  Point() {
  }
  Point(int x, int y) {
    this.x = x;
    this.y = y;
  }
  ...
}
```

Sobrecarga de construtores: exemplo de uso

- agora temos dois construtores e podemos escolher qual usar no momento da criação do objeto:

```
Point p1 = new Point();    // p1 está em (0,0)
Point p2 = new Point(1,2); // p2 está em (1,2)
```

Sobrecarga de métodos

- pode ser feita da mesma maneira que fizemos com os construtores

- quando sobrecarregamos um método, devemos manter a semântica: versões devem ter a mesma função

Sobrecarga de métodos: exemplo de uso

- a classe **Math** possui vários métodos sobrecarregados

- a semântica das várias versões são compatíveis

```
int a = Math.abs(-10);      // a = 10;
double b = Math.abs(-2.3); // b = 2.3;
```

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.

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)

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

Exercício: solução (classe Banco)

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



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

Exercício: solução (classe Banco)

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



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



 int pegaCodB(){return codigo;}
```

Exercício: solução (classe Banco)

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



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



 int pegaCodB(){return codigo;}


 String pegaNomeB(){return nome;}
```

Exercício: solução (classe Banco)

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



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



 int pegaCodB(){return codigo;}


 String pegaNomeB(){return nome;}
```

```
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;
}
```

Exercício: solução (classe Banco)

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



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



 int pegaCodB(){return codigo;}


 String pegaNomeB(){return nome;}
```

```
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;
}



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

Exercício: solução (classe Banco)

```
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)

```
class Conta{
 String nome;
 int codigo;
 Banco banco;
 float saldo;
```

Exercício: solução (classe Conta)

```
class Conta{
 String nome;
 int codigo;
 Banco banco;
 float saldo;


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

Exercício: solução (classe Conta)

```
class Conta{
 String nome;
 int codigo;
 Banco banco;
 float saldo;


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


 Banco pegaBanco(){return banco;}
```

Exercício: solução (classe Conta)

```
class Conta{
 String nome;
 int codigo;
 Banco banco;
 float saldo;


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


 Banco pegaBanco(){return banco;}


 String pegaNome(){return nome;}
```

Exercício: solução (classe Conta)

```
class Conta{
 String nome;
 int codigo;
 Banco banco;
 float saldo;


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


 Banco pegaBanco(){return banco;}


 String pegaNome(){return nome;}
```

```
 int pegaCodigo(){
  return codigo;
 }
```

Exercício: solução (classe Conta)

```
class Conta{
 String nome;
 int codigo;
 Banco banco;
 float saldo;


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


 Banco pegaBanco(){return banco;}


 String pegaNome(){return nome;}
```

```
 int pegaCodigo(){
  return codigo;
 }


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

Exercício: solução (classe Conta)

```
class Conta{
 String nome;
 int codigo;
 Banco banco;
 float saldo;


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


 Banco pegaBanco(){return banco;}


 String pegaNome(){return nome;}
```

```
int pegaCodigo(){
 return codigo;
}


float pegaSaldo(){
 return saldo;
}


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

Exercício: solução (classe Conta)

```
class Conta{
 String nome;
 int codigo;
 Banco banco;
 float saldo;


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


 Banco pegaBanco(){return banco;}


 String pegaNome(){return nome;}
```

```
int pegaCodigo(){
 return codigo;
}


float pegaSaldo(){
 return saldo;
}


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


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

Exercício: solução (classe Conta)

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

Exercício: solução (classe Poupanca)

```
class Poupanca extends Conta{
```

Exercício: solução (classe Poupanca)

```
class Poupanca extends Conta{

  Poupanca(String n, int c, Banco b){
    super (n, c, b);
  }
```

Exercício: solução (classe Poupanca)

```
class Poupanca extends Conta{

  Poupanca(String n, int c, Banco b){
    super (n, c, b);
  }


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

Exercício: solução (classe Poupanca)

```
class Poupanca extends Conta{

  Poupanca(String n, int c, Banco b){
    super (n, c, b);
  }


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


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

Exercício: solução (classe Teste)

```
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());
  }
}
```

OBS: Na verdade, chama-se o método retira de poupança quando tenta-se fazer qualquer retirada de uma poupança.