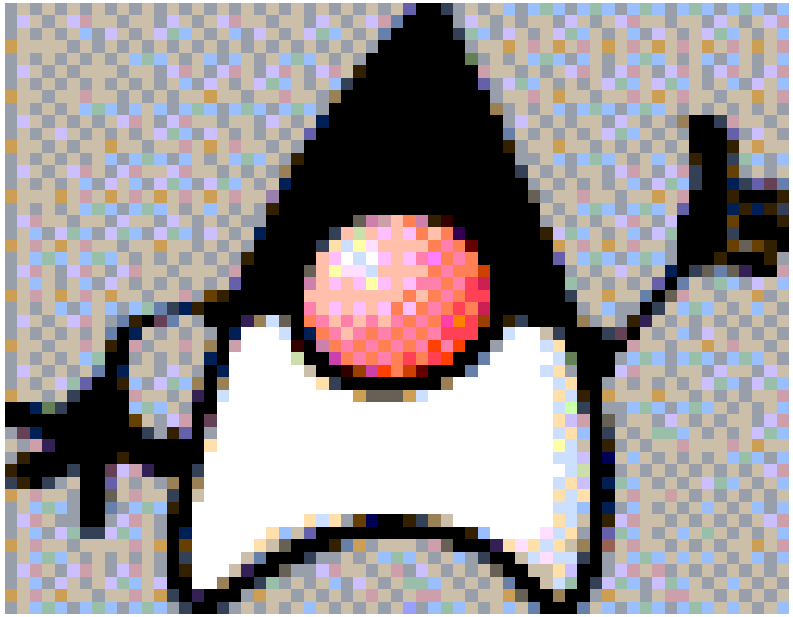
Introdução a Orientação a Objetos – Parte 2



Faculdade Dom Bosco de Porto Alegre
SisInfo & ADS – Estrutura de Dados
Profª Adriana Paula Zamin Scherer

Declarações de Métodos

accessLevel public, protected, private.

static Declara que o método é da classe, ao invés de ser do objeto.

returnType Tipo de retorno do método. Se não tem retorno, especificar: void.

Mensagem - chamada de método

Forma

< resultado = referência.método(parâmetros);>

resultado = terra.nomeOrbita();

- A captura do retorno é opcional.
- Parâmetros possuem tipo.
- O número de parâmetros é verificado em tempo de compilação.

Passagem de Parâmetros

- Toda passagem de parâmetro em Java é por valor.

  Obs.: Se o argumento sendo passado é uma referência a um objeto, então o valor passado é o endereço do objeto. Desta forma, o conteúdo do objeto referenciado pode ser alterado pelo método que recebe a referência.

Orientação a Objetos

Conceitos a serem abordados:

- **Características:**
  - ⇨ **Encapsulamento**
  - ⇨ **Polimorfismo**

Encapsulamento

- Mecanismo utilizado visando obter segurança, modularidade e autonomia para objetos;
- Conseguido através da definição de visibilidade privada dos atributos, ganhando-se assim autonomia para definir o que o mundo externo ao objeto poderá visualizar e acessar, normalmente através de métodos públicos.

  ***Dica**: sempre defina os atributos de uma classe como privados, a não ser que tenha uma boa justificativa para não serem.*

Encapsulamento - modificadores de visibilidade

public	Estes atributos e métodos são sempre acessíveis em todos os métodos de todas as classes. É o nível menos rígido de encapsulamento, equivale a não encapsular.
private	Estes atributos e métodos são acessíveis somente nos métodos (todos) da própria classe. Este é o nível mais rígido de encapsulamento.
protected	Estes atributos e métodos são acessíveis no pacote, nos métodos da própria classe e suas subclasses, o que será visto em Herança.
sem modifica-dor	Estes atributos e métodos são acessíveis dentro do mesmo pacote.

Encapsulamento

```
class Veiculo
{
   // variáveis de instância
   String nome;
   private float velocidade;

   // métodos
   public void acelera()
   {
      if (velocidade <= 10)
          velocidade ++;
   }

   void frea()
   {
      if (velocidade > 0)
          velocidade --;
   }

   void mostraVelocidade()
   {
      System.out.println(velocidade);
   }
}
```

Encapsulamento

```
class UsaVeiculo
{
  public static void main (String args[])
  {
     Veiculo v1 = new Veiculo();
     v1.nome = "Gol";
     System.out.println (v1.nome);
     v1.mostraVelocidade();

     for (int i=1; i<5; i++)
          v1.acelera();

     v1.mostraVelocidade();

//   System.out.println(v1.velocidade);
  }
}
```

this

- `this` é uma palavra-chave usada num método como referência para o objeto corrente.
- ela tem o significado de: “o objeto para o qual este trecho de código está sendo executado”.

this - exemplo

```
//Classe ponto
public class Ponto
{
   private float x,y;
   public Ponto(float x,float y)
   {
       this.x=x; this.y=y;
   }
   public void mostra()
   {
       System.out.println( "(" + this.x + "," +
                                    this.y + ")" );
   }
}
```

this - exemplo cont.

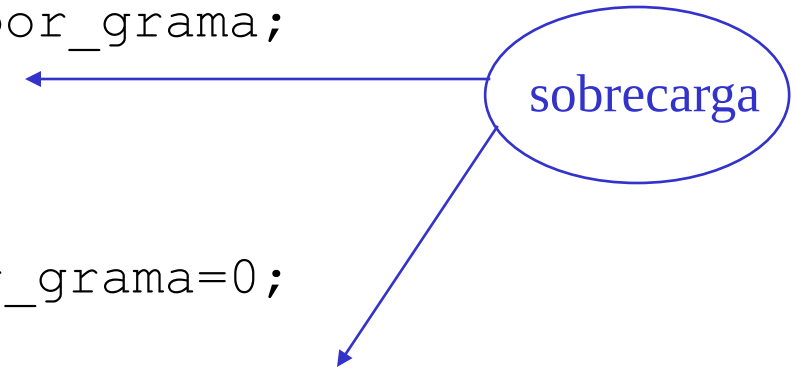
```
//Classe principal

public class Principal {

 public static void main(String args[]) {
  Ponto ap;
  ap=new Ponto((float)1.0,(float)3.0);
  ap.mostra();
 }
}
```

Polimorfismo (sobreposição e sobrecarga)

- Ocorre quando uma classe possui um **método com o mesmo nome e assinatura** (número, tipo e ordem de parâmetros) **de um método na sua superclasse**;
- Toda vez que isto ocorrer, a máquina virtual irá executar o método da classe mais especializada (a subclasse) e não o método da superclasse (sobreposição). Note que esta decisão ocorre em tempo de execução;
- Polimorfismo ocorre também quando existem **dois métodos com mesmo nome, na mesma classe e com assinaturas diferentes**. O método será escolhido de acordo com o número de parâmetros, tipo ou valor de retorno esperado. Note que esta decisão ocorre em tempo de compilação.

Polimorfismo - Sobrecarga

```
public class Fruta
{
        int gramas;
        int calorias_por_grama;
        public Fruta()
        {
          gramas=55;
          calorias_por_grama=0;
        }
        public Fruta(int g, int c)
        {
          gramas =g;
          calorias_por_grama =c;
        }
}
```



Polimorfismo - Sobrecarga

```
public class Feira
{
    public static void main(String args[])
    {
       Fruta melancia = new Fruta(4000, 5);
       Fruta manga = new Fruta ();
       manga.gramas=100;
       manga.calorias_por_grama=100;
       System.out.println("manga "+ manga.gramas +
       " gs " + manga.calorias_por_grama);
    }
}
```

Exercícios