Programação orientada a objetos

Aspectos fundamentais

Sobrecarga de métodos

- É permitido incluir numa classe métodos que possuem o mesmo nome e o mesmo tipo de retorno, mas que diferem pelo número e/ou pelos tipos dos argumentos.

- Qual destes métodos é executado depende do número e/ou dos tipos dos argumentos passados pelo programa que chama o método. Esta técnica é chamada sobrecarga ("overloading") de métodos. Ela é frequentemente utilizada, por exemplo, para definir vários construtores para uma determinada classe.

```
public class Ponto{
  protected double x, y;
  public Ponto(){
      setPonto( 0, 0 );
  }
  public Ponto( double a, double b ){
      setPonto( a, b );
  }
  public void setPonto( double a, double b ){
      x = a;
      y = b;
  }
  public String emPalavras(){
      return "[" + x + ", " + y + "]";
  }
}
```

Sobrecarga de métodos

Herança

- Pode-se criar uma nova classe, acrescentando recursos a uma classe já construída, ou modificando alguns recursos desta classe. A nova classe herda (possivelmente com modificações) os recursos já disponíveis na classe anterior.
- A classe herdeira é chamada subclasse da classe anterior, que é a sua superclasse.

Herança

- Este processo de herança pode ser repetido em cascata, criando várias gerações de classes.
- Vale notar que a linguagem Java permite somente herança simples, uma classe pode herdar diretamente de uma única classe (possui uma única superclasse direta). Em contrapartida, uma classe pode possuir várias subclasses diretas.
- A relação de herança é indicada através da palavra-chave **extends**. Por "default", uma classe herda da classe Object.

```
public class Circulo extends Ponto{
  protected double r;
  public Circulo(){
      /* aqui ocorre uma chamada implicita ao
      construtor da superclasse */
      setRaio( 0 );
  }
  public Circulo(double a, double b, double r){
      super( a, b );
      setRaio( r );
  }
  public void setRaio( double r ){
      this.r = r;
  }
}
```

Herança

Superação de métodos

- Na construção de uma subclasse a partir de uma superclasse, pode-se também substituir um método já presente na superclasse por outro, de mesmo nome, mesmo tipo de retorno, igual número e mesmos tipos de argumentos. Este procedimento é chamado "overriding" em inglês, o que será traduzido aqui por superação.

```
public class Circulo extends Ponto{
  protected double r;
  public Circulo(){
      setRaio( 0 );
  }
  public Circulo( double a, double b, double r ){
      super( a, b );
      setRaio( r );
  }
  public void setRaio( double r ){
      this.r = r;
  }
  public String emPalavras(){
      return "[" + x + ", " + y + ", " + r + "]";
  }
}
```

Superação de métodos

```
public class Circulo extends Ponto{
  protected double r;
  public Circulo(){
      setRaio( 0 );
  }
  public Circulo( double a, double b, double r ){
      super( a, b );
      setRaio( r );
  }
  public void setRaio( double r ){
      this.r = r;
  }
  public String emPalavras(){
      return super.emPalavras() + "[" + r + "]";
  }
}
```

Superação de métodos

Polimorfismo

- Já que uma subclasse herda as propriedades da superclasse, uma instância de uma subclasse é também uma instância da sua superclasse (e da superclasse desta, etc... até a classe raiz Object).
- Assim, no exemplo acima, um Circulo também é um Ponto e uma instância de Circulo pode ser atribuída a uma referência declarada como Ponto.

```
import java.awt.Label;
import java.applet.Applet;
public class Test extends Applet{
  private Ponto meusPontos[];
  private Label rohtulos[];
  public void init(){
    meusPontos = new Ponto[ 2 ];
    meusPontos[ 0 ] = new Ponto( 3, 5 );
    meusPontos[ 1 ] = new Circulo( 15, 5, 1.5 );
    rohtulos = new Label[ 2 ];
    for( int i = 0; i < meusPontos.length; i++ )
    {
      rohtulos[i]=new Label(meusPontos[i].emPalavras() );
      add( rohtulos[ i ] );
    }
  }
}
```

Polimorfismo

- Se uma instância de uma subclasse for atribuída a uma referência do tipo da superclasse, somente os métodos que já estão definidos na superclasse podem ser chamados como descrito acima. Para chamar um método definido na subclasse, mas não na superclasse, é necessário efetuar uma coerção explicita. Por exemplo, se acrescentarmos à classe Circulo um método para calcular a área, ou seja,

Polimorfismo

```
public class Circulo extends Ponto{
   protected double r;
   public Circulo(){
         setRaio( 0 );
   }
   public Circulo( double a, double b, double r ){
         super( a, b );
         setRaio( r );
   }
   public void setRaio( double r ){
         this.r = r;
   }
   public double ahrea( ){
         return Math.PI * r * r;
   }
   public String emPalavras(){
         return "[" + x + ", " + y + ", " + r + "]";
   }
}
```

Polimorfismo

- devemos usar a coerção para chamar este método numa instância de Circulo que foi atribuídda a uma referência de tipo Ponto:

```
import java.awt.Label;
import java.applet.Applet;
   public class Test extends Applet{
     private Ponto meusPontos[];
     private Label rohtulos[];
     public void init(){
     meusPontos = new Ponto[ 2 ];
     meusPontos[ 0 ] = new Ponto( 3, 5 );
     meusPontos[ 1 ] = new Circulo( 15, 5, 1.5 );
     rohtulos = new Label[ 2 ];
     for( int i = 0; i < meusPontos.length; i++ ){
         rohtulos[ i ] = new Label( meusPontos[ i ].emPalavras() );
         add( rohtulos[ i ] );
     }
     double a = ( ( Circulo ) meusPontos[ 1 ] ).ahrea();
     rohtulos[ 1 ].setText( rohtulos[ 1 ].getText() + "[" + a + "]" );
   }
}
```

Polimorfismo