Orientação a Objetos com Java: Herança e Interface

prof. Angelo C. Loula

Interfaces

- Interface: ‘contrato’ sobre o funcionamento de uma parte de uma classe
- Interfaces são similares a classes, mas possuem somente constantes e declaração de métodos, mas sem a definição (corpo) do método
- Interfaces não podem ser instanciadas, somente implementadas ou estentidas
- Ao implementar uma interface, a classe deve definir uma corpo para os métodos

Interfaces

```
public interface GroupedInterface {

    // constant declarations
    double E = 2.718282;  // base of natural logarithms

    // method signatures
    void doSomething (int i, double x);
    int doSomethingElse(String s);

}
```

Atributos são implicitamente `public, static, final`

- Herança (múltipla) em Interfaces
  - ampliando funcionalidades

```
public interface GroupedInterface extends Interface1,
                                          Interface2, Interface3 {
  ...
}
```



```
public interface Relatable {
   // returns 1, 0, -1 if this is greater
   // than, equal to, or less than other
   public int isLargerThan(Relatable other);

}
public class RectanglePlus implements Relatable {
    public int width = 0;
    public int height = 0;

    public RectanglePlus(int w, int h) {
       width = w;
       height = h;
    }
    public int getArea() {
       return width * height;
    }
    public int isLargerThan(Relatable other) {
       RectanglePlus otherRect = (RectanglePlus)other;
       if (this.getArea() < otherRect.getArea())
              return -1;
       else if (this.getArea() > otherRect.getArea())
              return 1;
       else
              return 0;
    }
}
```

Interfaces

- Interface como Tipo
  - A definição de uma interface define um tipo, assim como uma classe faria
  - Qualquer variável, atributo, parâmetro pode ser declarado como do tipo de interface e pode conter dentro dela qualquer objeto que *implemente* a interface.

```
public Object findLargest(Object object1, Object object2) {
   Relatable obj1 = (Relatable)object1;
   Relatable obj2 = (Relatable)object2;
   if ( (obj1).isLargerThan(obj2) > 0)
      return object1;
   else
      return object2;
}
```


Herança

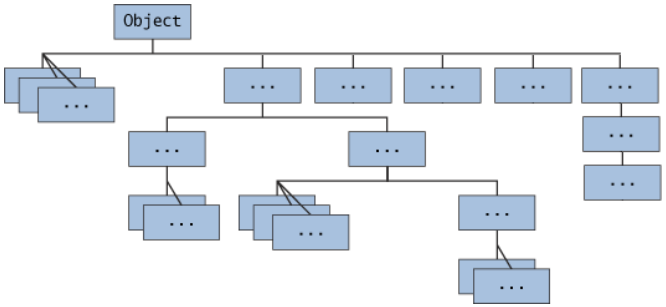
- Todas as classes (com exceção de Object) tem uma e somente uma classe pai
- Declarando uma subclasse

```
modif class nome extends superclasse{
  //atributos, métodos, etc
}
```

- Herda-se atributos, métodos e classes aninhadas, mas não construtores
  - Somente public, protected e pacote-protegido podem ser acessadas na subclasse.

Herança

- Todas as classes herdam (direta ou indiretamente) de `Object`




```
public class Bicycle {
   private int cadence = 0;
   private int speed = 0;
   private int gear = 1;

   public Bicycle(int cad, int spe, int ge) {
     gear = ge;
     cadence = cad;
     speed = spe;
   }
   public void changeCadence(int newValue) {
    cadence = newValue;
   }
   public void changeGear(int newValue) {
     gear = newValue;
   }
   public void speedUp(int increment) {
     speed = speed + increment;
   }
   public void applyBrakes(int decrement) {
     speed = speed - decrement;
   }
   public void printStates() {
     System.out.println("cadence:"+cadence+"
     speed:"+speed+" gear:"+gear);
   }
   public int getCadence(){
     return cadence;
   }
}
```

Definição da classe

Atributos

Construtor

Métodos

Uma subclasse

```
public class MountainBike extends Bicycle {

    private int seatHeight;


    public MountainBike(int startHeight, int
   startCadence, int startSpeed, int startGear) {
         super(startCadence, startSpeed, startGear);
         seatHeight = startHeight;
     }

     public void setHeight(int newValue) {
         seatHeight = newValue;
     }
    public void printStates() {
      System.out.println("cadence:"+this.getCadence()+"
      speed:"+speed+" gear:"+gear+" height:"+seatHeight);
    }

}
```


Herança

- Cast de Objetos
  - Um objeto pode ser convertido para sua superclasse ou subclasse
- Cast implícito:

```
Object obj = new MountainBike();
```

- Gera erro de compilação:

```
MountainBike myBike = obj;
```

- Cast explícito:

```
MountainBike myBike = (MountainBike)obj;
```

- Evitando erro de execução:

```
if (obj instanceof MountainBike) {
   MountainBike myBike = (MountainBike)obj;
}
```


- Sobrescrevendo e Escondendo

```
public class Animal {
    public static void testClassMethod() {
        System.out.println("Class method in Animal.");
    }
    public void testInstanceMethod() {
        System.out.println("Instance method in Animal.");
    }
}
public class Cat extends Animal {
    public static void testClassMethod() {
        System.out.println("Class method in Cat.");
    }
    public void testInstanceMethod() {
        System.out.println("Instance method in Cat.");
    }

    public static void main(String[] args) {
       Cat myCat = new Cat();
       Animal myAnimal = myCat;
       Animal.testClassMethod();
       myAnimal.testInstanceMethod();
    }
}
```

Esconder

Sobrescrever

Saída:
```
Class method in Animal.
Instance method in Cat.
```


- Expressão `super`

```
public class Superclass {
    public void printMethod() {
        System.out.println("Printed in Superclass.");
    }
}

public class Subclass extends Superclass {

    public void printMethod() {
        super.printMethod();
        System.out.println("Printed in Subclass");
    }
    public static void main(String[] args) {
      Subclass s = new Subclass();
      s.printMethod();
    }

}
```

Saída:
```
Printed in Superclass.
Printed in Subclass
```

- Expressão `super`

```
public MountainBike(int startHeight, int startCadence,
  int startSpeed, int startGear) {
        super(startCadence, startSpeed, startGear);
        seatHeight = startHeight;
    }
```

- Se não houver chamada explicita para construtor da superclasse, será inserido de forma automática e implícita a chamada `super();`
  - `Object` possui construtor sem parâmetros

Herança

- `final` para métodos e classes
- Métodos declarados como `final` não podem ser sobrescritos
  - Construtores podem fazem uso de outros métodos, e estes métodos devem ser final para não modificar a funcionalidade do construtor
- Classes declaradas como `final` não podem ser estendidas (possuir subclasse)

Herança

- Classes abstratas
  - Classes que não podem ser instanciadas
  - Podem somente ser estendidas
  - Inclua `abstract` na declaração
  - Podem possuir método abstratos
  - Subclasses podem também ser abstratas
- Métodos abstratos
  - Somente declarados mas sem implementação
  - Toda classe com métodos abstratos deve ser abstrata

Herança

- Classes abstratas vs Interfaces
  - Herança Múltipla
  - Classes abstratas podem ter atributos não estáticos, não finais
  - Classes abstratas podem ter métodos implementados
  - Classes abstratas só com métodos abstratos devem ser interfaces