Classes abstratas e interfaces em JAVA

Paradigmas de Linguagens de Programação

Programação Orientada a Objetos

Profa. Heloisa

Classes Abstratas e Métodos Abstratos

- ***Métodos Abstratos*** – métodos que não tem corpo.

- ***Classes abstratas*** – possuem métodos abstratos.
  - Subclasses devem definir os métodos abstratos, ou serem também abstratas.
  - Não podem ser instanciadas.
  - Campos não podem ser abstratos.
  - Podem conter métodos não abstratos, que serão herdados pelas subclasses.
  - Pode-se declarar variáveis de referência do tipo da classe abstrata, que quando fizerem referência a objetos das subclasses, dão acesso a métodos das subclasses.

Exemplos de classes abstratas

```
abstract class Classe {
   public  abstract void  M1();
   public abstract void  M2() ;
   public  void M3 () { System.out.println("Metodo M3");  }
 }

class SubClasse1 extends Classe {
   public void M1 () {  System.out.println("Metodo M1- Subclasse 1");  }
   public void M2 () {  System.out.println("Metodo M2 - Subclasse 1");  }
}

class SubClasse2 extends Classe {
   public void M1 () {  System.out.println("Metodo M1 - Subclasse 2");  }
   public void M2 () { System.out.println("Metodo M2 - Subclasse 2");  }
 }
```

```
public class UsaClasse {

  public static void main (String S[ ]) {
    Classe Objeto[ ] = new Classe[2]; //instancia um array de objetos
    int i;

    Objeto[0] = new SubClasse1();
    Objeto[1] = new SubClasse2();
    Objeto[0].M1();
    Objeto[0].M2();
    Objeto[1].M1();
    Objeto[1].M2();
   }
}
```

O método toString

- Retorna os campos da classe em que está definido na forma de String;
- Deve ser declarado como public e retornar uma String;
- Se o método toString existir em uma classe, as instâncias dessa classe poderão ser impressas ou processadas como uma String diretamente a partir da referência à instância, sem que o método precise ser chamado diretamente.

Exemplo usando classes abstratas

```
abstract class ObjetoGeometrico
 {
  public abstract Ponto2D centro();
  public abstract double calculaArea();
  public abstract double calculaPerimetro();
  }
```

```
class Circulo extends ObjetoGeometrico
  {
  private Ponto2D centro;
   private double raio;
   Circulo (Ponto2D centro, double raio)
     {
      this.centro = centro;
      this.raio = raio;
     }
  public Ponto2D centro ()
     {    return centro;  }
  public double calculaArea()
     {  return Math.PI*raio*raio;  }
  public double calculaPerimetro ()
      {  return 2.0*Math.PI*raio;  }
   public String toString ()
      {   return "Circulo com centro em "+centro+" e raio "+raio; }
 }
```

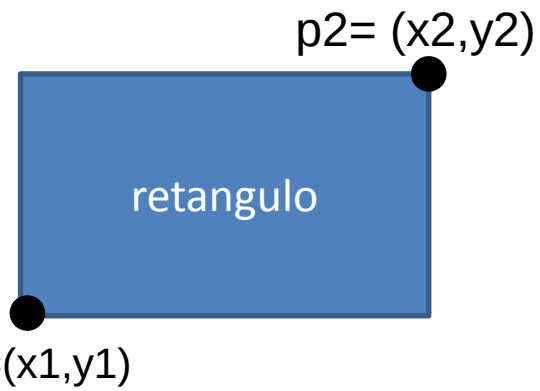
- A classe Circulo deve implementar todos os métodos que foram declarados na classe abstrata ObjetoGeometrico ou ser também abstrata

- Os métodos da classe abstrata não poderiam ser declarado como ***private*** ou ***protected*** pois não é possível fazer a sobrecarga de um método se o acesso é tornado mais restritivo.

- Os métodos também não poderiam ser declarado sem modificador nem como ***static***.

```
class Retangulo extends ObjetoGeometrico
  {
  private Ponto2D primeiroCanto, segundoCanto;

   Retangulo (Ponto2D pc, Ponto2D sc)
    {
     primeiroCanto = pc;
     segundoCanto = sc;
    }
  public Ponto2D centro ()
    {
        double coordX = (primeiroCanto.getX() + segundoCanto.getX())/2.;
        double coordY = (primeiroCanto.getY() + segundoCanto.getY())/2.;
        return new Ponto2D (coordX, coordY);
    }
  public double calculaArea()
    {
       double ladoX = Math.abs(primeiroCanto.getX()-segundoCanto.getX());
       double ladoY = Math.abs(primeiroCanto.getY()-segundoCanto.getY());
       return ladoX*ladoY;
     }
```



```
public double calculaPerimetro ()
    {
      double ladoX = Math.abs(primeiroCanto.getX()-segundoCanto.getX());
      double ladoY = Math.abs(primeiroCanto.getY()-segundoCanto.getY());
      return 2*ladoX+2*ladoY;
    }
  public String toString ()
    {   return "Retangulo com cantos "+primeiroCanto+" e "+segundoCanto;
     }
}
```

```
class DemoOG
  {
  public static void main (String[ ] argumentos)
    {
     Circulo c1 = new Circulo (new Ponto2D(0,0),100);
     Circulo c2 = new Circulo (new Ponto2D(-1,-1),1);
     Circulo c3 = new Circulo (new Ponto2D(10,8),0);
     Retangulo r1 = new Retangulo (new Ponto2D(-2,-2), new Ponto2D(2,2));
     Retangulo r2 = new Retangulo (new Ponto2D(-100,-1), new Ponto2D(100,1));
     Retangulo r3 = new Retangulo (new Ponto2D(0,0), new Ponto2D(0,0));

// Imprimir todos os dados de cada um desses objetos geometircos
     imprimeTodosOsDados(c1);
     imprimeTodosOsDados(c2);
     imprimeTodosOsDados(c3);
     imprimeTodosOsDados(r1);
     imprimeTodosOsDados(r2);
     imprimeTodosOsDados(r3);
    }  // fim de main
```

```
public static void imprimeTodosOsDados (ObjetoGeometrico og)
 {
  System.out.println(og);
  System.out.println("Perimetro:"+og.calculaPerimetro());
  System.out.println("Area:"+og.calculaArea());
  System.out.println();
 }
```

- Se uma classe é subclasse de outra classe abstrata, uma instância dessa classe pode ser passada como argumento para qualquer método que espere como argumento uma **referência a classe abstrata** (mecanismo de polimorfismo)

- Embora não seja possível criar instâncias de classes abstratas, a referência a og pode comportar-se como uma instância de Circulo ou Retangulo e ser usada para chamar os métodos que foram declarados na classe ObjetoGeometrico.

```
class DemoOG2
  {
  public static void main (String[ ] argumentos)
    {
     ObjetoGeometrico o1, o2, o3, o4;
     o1 = new Circulo (new Ponto2D(0,0),100);
     o2 = new Retangulo (new Ponto2D(-2,-2), new Ponto2D(2,2));
     o3 = new Circulo (new Ponto2D(10,8),0);
     o4 = new Retangulo (new Ponto2D(-100,-1), new Ponto2D(100,1));

// Vamos ver que referência é instância de que classe
      System.out.println("o1 é um circulo?  "+ o1 instanceof  Circulo);
      System.out.println("o1 é um retangulo?  "+ o1 instanceof  Retangulo);
      System.out.println("o1 é um ObjetoGeometrico?  "+ o1 instanceof  ObjetoGeometrico);
      System.out.println("o2 é um circulo?  "+ o2 instanceof  Circulo);
      System.out.println("o2 é um retangulo?  "+ o2 instanceof  Retangulo);
      System.out.println("o2 é um ObjetoGeometrico?  "+ o2 instanceof  ObjetoGeometrico);
    }  // fim de main
```

- No método main foram declaradas referências da classe abstrata ObjetoGeometrico.

- Não é possível criar instâncias da classe abstrata mas as referências a ela poderão apontar para instâncias de classes que derivadas dela.

- O operador instanceof retorna true se o objeto é instância da classe e false se não for.

- Com o resultado do programa podemos ver que qualquer uma das instâncias pode ser considerada como instância de sua própria classe ou da classe ObjetoGeometrico.

Passagens de parâmetros a métodos – tipos primitivos

- Tipos primitivos de dados são passados por valor

```
public class PassaPrimitivo {
  public static void main(String[] args) {
     int x = 3;
     // chama passMethod() com argumento x
     passMethod(x);
     // imprime x para ver se o valor mudou
      System.out.println(“Depois de chamar passMethod, x = " + x);
   }
    // muda parâmetro em passMethod()
    public static void passMethod(int p) { p = 10; }
 }
```

Saída: Depois de chamar passMethod, x = 3

Passagens de parâmetros a métodos – tipos de referência

- Tipos de dados de referência (objetos, arrays, strings) também são passados por valor
  - Quando o método retorna, a referência passada como argumento ainda faz referência ao mesmo objeto
  - Entretanto, os valores dos campos dos objetos podem ser alterados dentro do método, se tiverem o nível de acesso apropriado

Passagens de parâmetros a métodos – tipos de referência

- Suponha que a classe Círculo tem um método que move a figura geométrica, acrescentando valores nas coordenados do centro da figura

```
public void moveCirculo(Circulo circulo, int deltaX, int deltaY) {
    // código para mover a origem do circulo para x+deltaX, y+deltaY
     circulo.setX(circulo.getX() + deltaX);
     circulo.setY(circulo.getY() + deltaY);
    // codigo para atribuir uma nova referência a circle
      circulo = new Circulo(0, 0);
}
```

Passagens de parâmetros a métodos – tipos de referência

- Supondo que o método foi chamado com os parâmetros:

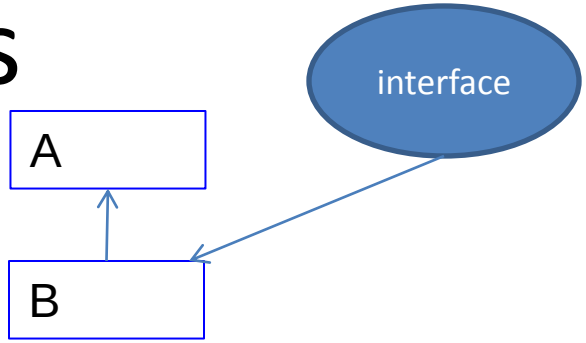
  moveCirculo(meuCirculo, 15, 22)

- As coordenadas do centro do círculo são alteradas pelo método

- A referência é alterada dentro do método para um novo objeto com centro x = 0 e y = 0

- Quando o método retorna, a referência tem o mesmo valor de antes da chamada

Interfaces

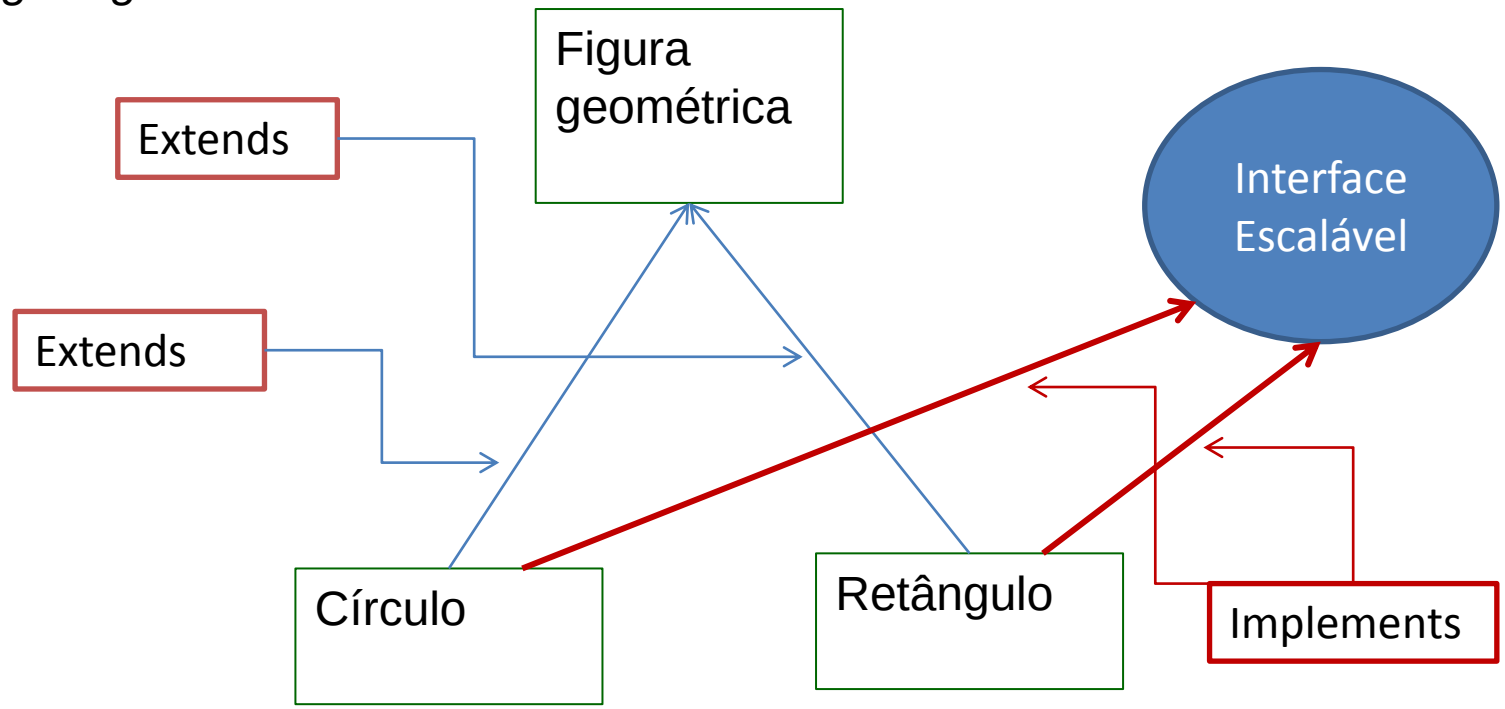


- Tem funcionalidade semelhante a das classes abstratas mas segue um modelo de declaração diferente.
- Podem ser vistas como um "contrato" que define tudo o que uma classe deve fazer se quiser ter um determinado status.
- Chama-se interface pois é a maneira pela qual poderemos conversar com um objeto que assinou o contrato estabelecido pela interface (ou implementa os métodos da interface)
- Ela é um contrato onde quem assina se responsabiliza por implementar esses métodos (cumprir o contrato).

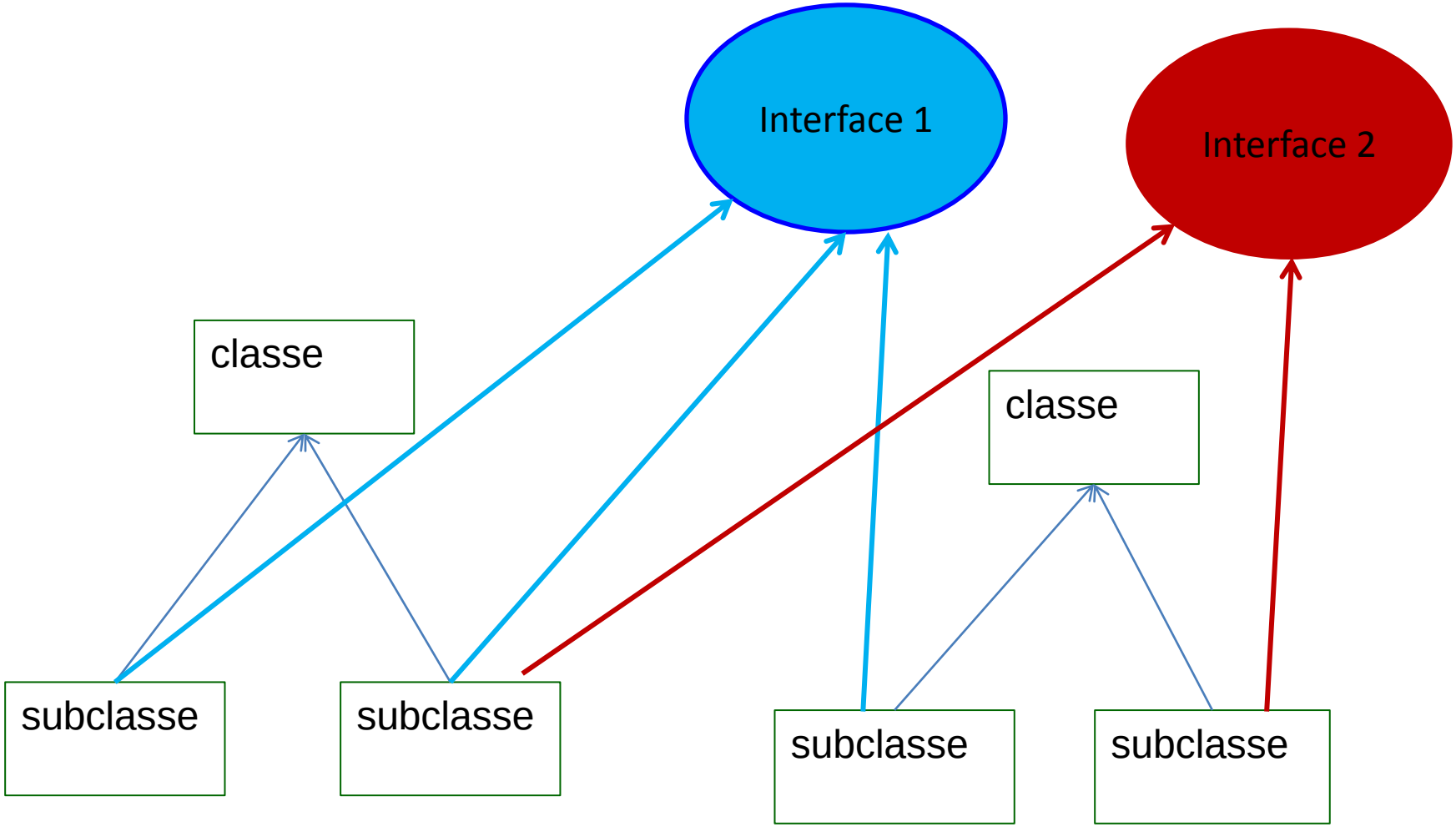
A interface é um contrato que quem assina se responsabiliza por implementar seus métodos (cumprir o contrato).

"Contrato:
"quem desejar ser escalável precisa saber ampliar e espelhar sua respectiva figura geométrica".

Métodos de escalável:
Amplia
Espelha



Classes não relacionadas por hierarquia podem implementar uma mesma interface, se tiverem um fator comum



Interfaces

- Qualquer classe pode implementar várias interfaces simultaneamente.
- É um tipo específico de herança múltipla.
- A interface define **o que o objeto deve fazer**, e não **como ele faz**, nem **o que ele tem**.
- **Como o objeto faz** alguma coisa vai ser definido em uma **implementação** dessa interface.
- Interfaces não podem ser instanciadas.
- Podem ser **implementadas** por classes ou **estendidas** por outras interfaces

Definição de interfaces

- Declaração de interfaces:

<Modificadores> **interface** <Nome da interface > [**extends** interface-pai1, interface-pai2, ..., interface-pain]

{.... .....}

```
public interface Ex_Interface extends Interface1, Interface2, Interface3
   { // declaração de constantes
     // base natural logaritmo natural
        double E = 2.718282;
     // assinatura de metodos
     void doSomething (int i, double x);
     int doSomethingElse(String s);
    }
```

Implementação de interfaces

- É implementada com a palavra ***implements*** seguida pelo nome de uma ou mais interfaces, separados por vírgula

- Declaração de classe que implementa interfaces:

<Modificadores> **class** <Nome da classe > [**extends** <Nome da superclasse] [**implements** interface1, interface2, ..., interfacen]
{.... .....}

- Modificadores:
  - Modificador ***public***: interface acessível por todas as classes de todos os pacotes
  - Sem modificador: interface acessível pelas classes do mesmo pacote.

Exemplo usando interface - 1

```
interface Interface {
   public   void  M1( );
   public  void  M2( ) ;
 }
class Classe {
   public  void M3 ( ) { System.out.println("Metodo M3");  }
 }
 class SubClasse1 extends Classe implements Interface {
    public void M1 ( ) { System.out.println("Metodo M1- Subclasse 1");  }
    public void M2 ( ) { System.out.println("Metodo M2 - Subclasse 1");  }
}
class SubClasse2 extends Classe implements Interface {
    public void M1 () { System.out.println("Metodo M1 - Subclasse 2");  }
    public  void M2 () { System.out.println("Metodo M2 - Subclasse 2");  }
}
```

Usando interface como tipo

- Quando uma interface é definida, define-se um novo tipo de dados de referência
- Um nome de interface pode ser usado em qualquer lugar que seria usado um outro nome de tipo de dado
- Se uma variável é definida como sendo do tipo de uma interface, qualquer objeto atribuído a ela **deve** ser uma instância de uma classe que implementa a interface

```
public class UsaInterface {
  public static void main (String S[]) {
     Interface Objeto[ ] = new Interface[2];
     int i;
     Objeto[0] = new SubClasse1( );
     Objeto[1] = new SubClasse2( );
     Objeto[0].M1( );
     Objeto[0].M2( );
     Objeto[1].M1( );
     Objeto[1].M2( );
   }
 }
```

Exemplo usando interface - 2

```
interface Interface2 {
   public   void  M1( );
   public   void  M2( ) ;
 }
class Classe {
   public  void M3 ( ) { System.out.println("Metodo M3 - Classe");  }
 }
class OutraClasse {
 public  void M4 ( ) { System.out.println("Metodo M4 - Outra Classe");  }
}
class SubClasse1 extends Classe implements Interface2 {
    public void M1 ( ) { System.out.println("Metodo M1- Subclasse 1");  }
    public void M2 ( ) { System.out.println("Metodo M2 - Subclasse 1");  }
}
```

```
class SubClasse2 extends Classe {
   public void M4 () { System.out.println("Metodo M4 - Subclasse 2");  }
}

class SubClasse3 extends OutraClasse implements Interface2 {
   public void M1 () { System.out.println("Metodo M1- Subclasse 3");  }
   public void M2 () { System.out.println("Metodo M2 - Subclasse 3");  }
}
```

```
public class UsaInterface2 {
  public static void main (String S[]) {
    Interface2 Objeto[] = new Interface2[2];
    int i;
    SubClasse2 OutroObjeto = new SubClasse2();
    Classe ObjetoClasse = new Classe();
    Objeto[0] = new SubClasse1();
    Objeto[1] = new SubClasse3();
    Objeto[0].M1();            // M1 de SubClasse1
    Objeto[0].M2();            // M2 de SubClasse1
    Objeto[1].M1();            // M1 de SubClasse3
    Objeto[1].M2();            // M2 de SubClasse3
    OutroObjeto.M4();          // M4 de SubClasse2
    ObjetoClasse.M3();         // M3 de Classe

  }
}
```

Interfaces

- Interfaces podem conter constantes, assinaturas de métodos, métodos default, métodos estáticos (Java 8) e tipos aninhados

- Uma interface pode definir uma série de métodos, mas só contém a implementação de métodos estáticos ou default.

- Métodos default são definidos com a palavra chave ***default*** e métodos estáticos são definidos com a palavra chave ***static***.

- Campos, se houver, são implicitamente ***static*** e ***final***, devendo ser inicializados na sua declaração.

- Podem ser úteis, entre outras coisas, para implementar bibliotecas de constantes, já que os campos devem ser ***static*** e ***final***.

Método default

- Métodos default são métodos concretos (tem código definido) na interface.
- Permitem evoluir interfaces sem ter que modificar as classes que implementam as interfaces.
- É declarado com o modificador default

Exemplo – interface com método abstrato

```
interface InterfaceA {
   public void algumaCoisa();
}

public class MinhaClasse implements InterfaceA {
   public static void main(String[] args) {
   // código de main
   }
   public void algumaCoisa() {
      System.out.println(“Ola mundo!");
   }
}
```

Exemplo – interface com método abstrato e default

```
interface InterfaceA {
   public void algumaCoisa();
   default public void digaOla() {
        System.out.println(“Ola");
    }
}
public class MinhaClasse implements InterfaceA {
   public static void main(String[] args) {
   // código de main
   }
   public void algumaCoisa() {
      System.out.println(“Ola mundo!");
   }
}
```

Conflito entre interfaces múltiplas

- Como as classes podem implementar mais de uma interface, podem ocorrer conflitos entre interfaces que possuem métodos default com mesma assinatura
- Esse conflito gera um **erro de compilação**
- Para eliminar esse conflito é necessário redefinir o método na classe que implementa as interfaces com um código específico para a classe, sobrepondo os métodos das interfaces

Exemplo – conflito entre métodos default de interfaces implementadas pela mesma classe

```
interface InterfaceA {
   public void algumaCoisa();
   default public void digaOla() { System.out.println(“Eu sou a interface A") ; }
}
interface InterfaceB {
   default public void digaOla() { System.out.println(“Eu sou a interface B") ; }
}
public class MinhaClasse implements InterfaceA, InterfaceB {
   .........
   public void algumaCoisa() {
      System.out.println(“Ola mundo!");
   }
   public void digaOla() {
       System.out.println(“Eu sou a classe MinhaClasse”);
   }
}
```

Métodos static em interfaces

- Interfaces podem conter também métodos estáticos
- São declarados com a palavra chave ***static***
- Semelhantes aos métodos default, exceto que não podem ser sobrepostos nas classes que implementam a interface

```
interface X {
     static void digaOi() { System.out.println(“Oi"); }
 }
 class Y implements X {... }
public class Z {
   public static void main (String[] args) {
        X.digaOi();
        // Y.digaOi(); // não compila
}
}
```

Exemplo usando interface - 3

```
Interface ObjetoGeometrico
 {
  Ponto2D centro();
  double calculaArea();
  double calculaPerimetro();
  }
```

```
class Circulo implements ObjetoGeometrico
  {
  private Ponto2D centro;
   private double raio;
   Circulo (Ponto2D centro, double raio)
     {
      this.centro = centro;
      this.raio = raio;
     }
  public Ponto2D centro ()
     {    return centro;  }
  public double calculaArea()
     {  return Math.PI*raio*raio;  }
  public double calculaPerimetro ()
      {  return 2.0*Math.PI*raio;  }
   public String toString ()
      {   return "Circulo com centro em "+centro+" e raio "+raio; }
 }
```

- A classe Circulo deve implementar todos os métodos que foram declarados na interface ObjetoGeometrico.

- O método centro era implicitamente ***public*** na interface e foi declarado como ***public*** na classe Circulo.

- O método centro não poderia ser declarado como ***private*** ou ***protected*** pois não é possível fazer a sobrecarga de um método se o acesso é tornado mais restritivo.

- O método centro não poderia ser declarado sem modificador nem como ***static***.

```
class Retangulo implements ObjetoGeometrico
  {
  private Ponto2D primeiroCanto, segundoCanto;

   Retangulo (Ponto2D pc, Ponto2D sc)
    {
     primeiroCanto = pc;
     segundoCanto = sc;
    }
  public Ponto2D centro ()
    {
        double coordX = (primeiroCanto.getX() + segundoCanto.getX())/2.;
        double coordY = (primeiroCanto.getY() + segundoCanto.getY())/2.;
        return new Ponto2D (coordX, coordY);
    }
  public double calculaArea()
    {
       double ladoX = Math.abs(primeiroCanto.getX()-segundoCanto.getX());
       double ladoY = Math.abs(primeiroCanto.getY()-segundoCanto.getY());
       return ladoX*ladoY;
     }
```

```
public double calculaPerimetro ()
    {
      double ladoX = Math.abs(primeiroCanto.getX()-segundoCanto.getX());
      double ladoY = Math.abs(primeiroCanto.getY()-segundoCanto.getY());
      return 2*ladoX+2*ladoY;
    }
  public String toString ()
    {   return "Retangulo com cantos "+primeiroCanto+" e "+segundoCanto;
     }
}
```

```
class DemoOG
  {
  public static void main (String[ ] argumentos)
    {
     Circulo c1 = new Circulo (new Ponto2D(0,0),100);
     Circulo c2 = new Circulo (new Ponto2D(-1,-1),1);
     Circulo c3 = new Circulo (new Ponto2D(10,8),0);
     Retangulo r1 = new Retangulo (new Ponto2D(-2,-2), new Ponto2D(2,2));
     Retangulo r2 = new Retangulo (new Ponto2D(-100,-1), new Ponto2D(100,1));
     Retangulo r3 = new Retangulo (new Ponto2D(0,0), new Ponto2D(0,0));

// Imprimir todos os dados de cada um desses objetos geometricos
     imprimeTodosOsDados(c1);
     imprimeTodosOsDados(c2);
     imprimeTodosOsDados(c3);
     imprimeTodosOsDados(r1);
     imprimeTodosOsDados(r2);
     imprimeTodosOsDados(r3);
    }  // fim de main
```

```
public static void imprimeTodosOsDados (ObjetoGeometrico og)
 {
  System.out.println(og);
  System.out.println("Perimetro:"+og.calculaPerimetro());
  System.out.println("Area:"+og.calculaArea());
  System.out.println();
 }
```

- Se uma classe implementa uma interface, uma instância dessa classe pode ser passada como argumento para qualquer método que espere como argumento uma referência a interface implementada (mecanismo de polimorfismo)

- Embora não seja possível criar instâncias de interfaces, a referência a og pode comportar-se como uma instância de Circulo ou Retangulo e ser usada para chamar os métodos que foram declarados na interface.

```
class DemoOG2
  {
  public static void main (String[ ] argumentos)
    {
     ObjetoGeometrico o1, o2, o3, o4;
     o1 = new Circulo (new Ponto2D(0,0),100);
     o2 = new Retangulo (new Ponto2D(-2,-2), new Ponto2D(2,2));
     o3 = new Circulo (new Ponto2D(10,8),0);
     o4 = new Retangulo (new Ponto2D(-100,-1), new Ponto2D(100,1));

// Vamos ver que referência é instância de que classe
      System.out.println("o1 é um circulo?  "+ o1 instanceof  Circulo);
      System.out.println("o1 é um retangulo?  "+ o1 instanceof  Retangulo);
      System.out.println("o1 é um ObjetoGeometrico?  "+ o1 instanceof  ObjetoGeometrico);
      System.out.println("o2 é um circulo?  "+ o2 instanceof  Circulo);
       System.out.println("o2 é um retangulo?  "+ o2 instanceof  Retangulo);
      System.out.println("o2 é um ObjetoGeometrico?  "+ o2 instanceof ObjetoGeometrico);
   }  // fim de main
```

- No método main foram declaradas referências da interface ObjetoGeometrico.

- Não é possível criar instâncias da interface mas as referências a ela poderão apontar para instâncias de classes que implementam a interface.

- O operador instanceof retorna true se o objeto é instância da classe e false se não for.

- Com o resultado do programa podemos ver que qualquer uma das instâncias pode ser considerada como instância de sua própria classe ou da interface ObjetoGeometrico.

Herança Múltipla usando Interfaces

- Exemplo: alguns objetos geométricos podem ser escaláveis (seu tamanho original pode ser modificado).

- A interface Escalavel declara métodos que um objeto geométrico escalável deve implementar:

  - método que permite a modificação do tamanho
  - método que permite o espelhamento do objeto – modificação de sua posição horizontal.

```
Interface Escalavel
 {
   void amplia(double escala);
   void espelha()
 }
```

```
class CirculoEscalavel implements ObjetoGeometrico, Escalavel
  {
  private Ponto2D centro;
   private double raio;
   CirculoEscalavel (Ponto2D centro, double raio)
     {
      this.centro = centro;
      this.raio = raio;
     }
  public Ponto2D centro ()
     {    return centro;  }
  public double calculaArea()
     {  return Math.PI*raio*raio;  }
  public double calculaPerimetro ()
      {  return 2.0*Math.PI*raio;  }
```

```
public void amplia (double escala)
   {
     raio *= escala;
    }

 public void espelha ()
    {
      centro = new Ponto2D(-centro.getX(),centro.getY());
    }

 public String toString ()
    {
     return “Circulo com centro em “+centro+” e raio “+raio; }
    }
```

```
class DemoCirculoEscalavel
  {
  public static void main (String[ ] argumentos)
    {
     CirculoEscalavel  ce = new CirculoEscalavel (new Ponto2D(10,10),30);

      System.out.println(ce);
      ce.amplia(3);
       System.out.println(ce);
       ce.espelha();
       System.out.println(ce);
       System.out.println("ce é um circulo escalavel?  "+
                                              ce instanceof  CirculoEscalavel);
       System.out.println("ce é um ObjetoGeometrico?  "+
                                              ce instanceof  ObjetoGeometrico);
       System.out.println("ce é escalavel?  "+ ce instanceof  Escalavel);
    }  // fim de main
```