Paradigmas de Programação

Aula 08

1. Palavras Reservadas
2. Classes Abstractas
3. Exemplos
4. Usar o final
5. Palavras Reservadas Usadas
6. Links Úteis

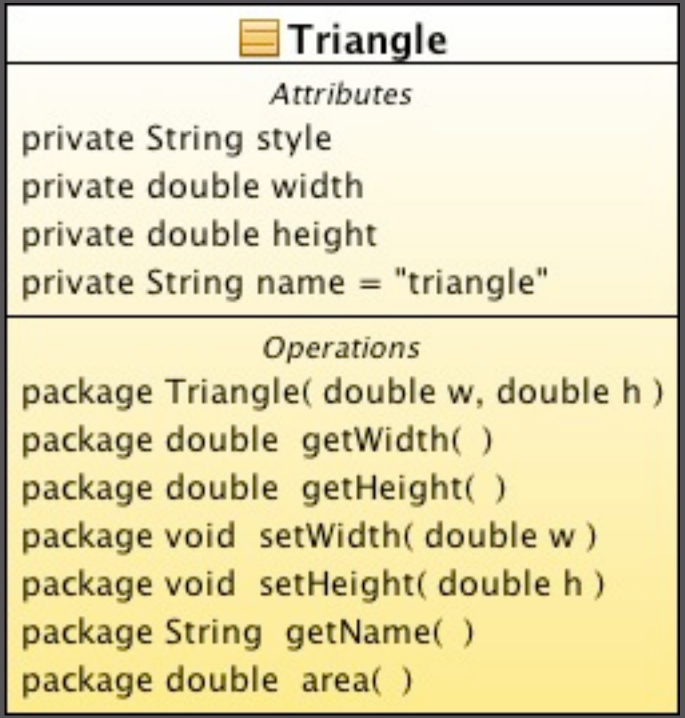
Palavras reservadas usadas

- abstract
- assert***
- boolean
- break
- byte
- case
- catch
- char
- class
- const*
- continue
- default
- do
- double
- else
- enum****
- extends
- final
- finally
- float
- for
- goto*
- if
- implements
- import
- instanceof
- int
- interface
- long
- native
- new
- package
- private
- protected
- public
- return
- short
- static
- strictfp**
- super
- switch
- synchronized
- this
- throw
- throws
- transient
- try
- void
- volatile
- while

```
   * not used
  ** added in 1.2
 *** added in 1.4
**** added in 5.0
```

```java
public class Rectangle {

    private double width;
    private double height;
    private String name="rectangle";

    Rectangle(double w, double h) {
        this.width=w;
        this.height=h;
    }

    double getWidth() { return width; }
    double getHeight() { return height; }
    void setWidth(double w) { width = w; }
    void setHeight(double h) { height = h; }
    String getName() { return name; }

    boolean isSquare() {
        if(getWidth() == getHeight()) return true;
        return false;
    }

    double area() {
        return getWidth() * getHeight();
    }
}
```

```java
public class Triangle {

    private String style;
    private double width;
    private double height;
    private String name="triangle";

    // Constructor for Triangle.
    Triangle(double w, double h) {
        this.width=w;
        this.height=h;
    }

    // Accessor methods for width and height.
    double getWidth() { return width; }
    double getHeight() { return height; }
    void setWidth(double w) { width = w; }
    void setHeight(double h) { height = h; }
    String getName() { return name; }


    double area() {
        return getWidth() * getHeight() / 2;
    }
}
```



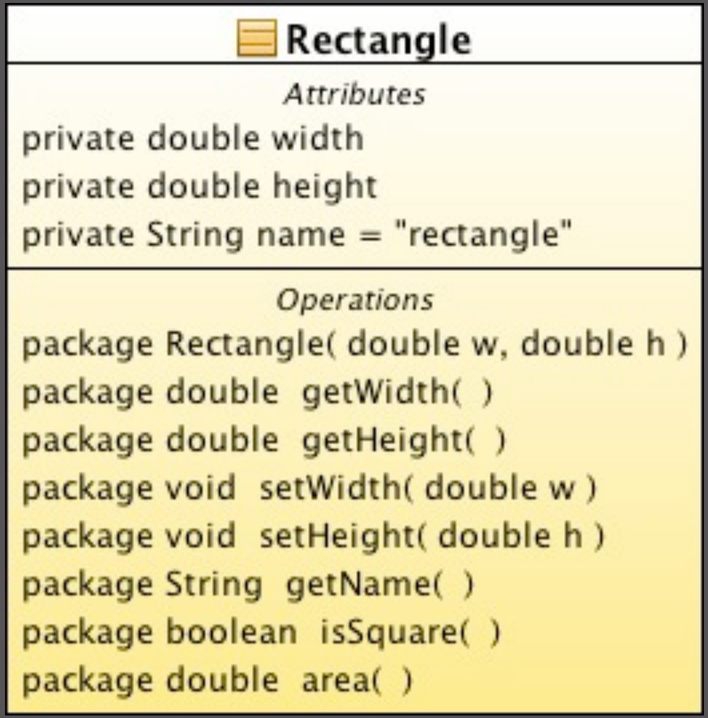
Triangle

Attributes

private String style
private double width
private double height
private String name = "triangle"

Operations

package Triangle(double w, double h)
package double getWidth( )
package double getHeight( )
package void setWidth(double w)
package void setHeight(double h)
package String getName( )
package double area( )



Rectangle

Attributes

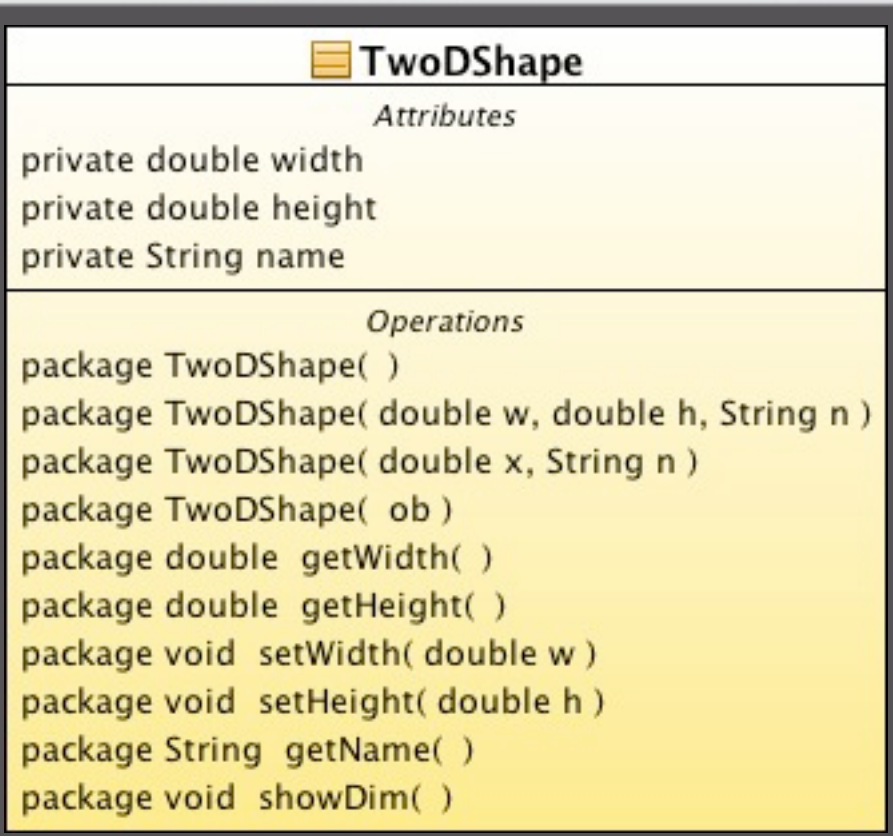
private double width
private double height
private String name = "rectangle"

Operations

package Rectangle(double w, double h)
package double getWidth( )
package double getHeight( )
package void setWidth(double w)
package void setHeight(double h)
package String getName( )
package boolean isSquare( )
package double area( )

```
public class TwoDShape {

    private double width;
    private double height;
    private String name;

    // A default constructor.
    TwoDShape() {
        width = height = 0.0;
        name = "null";
    }

    // Parameterized constructor.
    TwoDShape(double w, double h, String n) {
        width = w;
        height = h;
        name = n;
    }

    // Construct object with equal width and height.
    TwoDShape(double x, String n) {
        width = height = x;
        name = n;
    }

  // continua no próximo slide
```

```
    // continuação do slide anterior



    // Construct an object from an object.
    TwoDShape(TwoDShape ob) {
        width = ob.width;
        height = ob.height;
        name = ob.name;
    }

    // Accessor methods for width and height.
    double getWidth() { return width; }
    double getHeight() { return height; }
    void setWidth(double w) { width = w; }
    void setHeight(double h) { height = h; }
    String getName() { return name; }

    void showDim() {
        System.out.println("Width and height are " +
                width + " and " + height);
    }

}
```



TwoDShape

Attributes

private double width
private double height
private String name

Operations

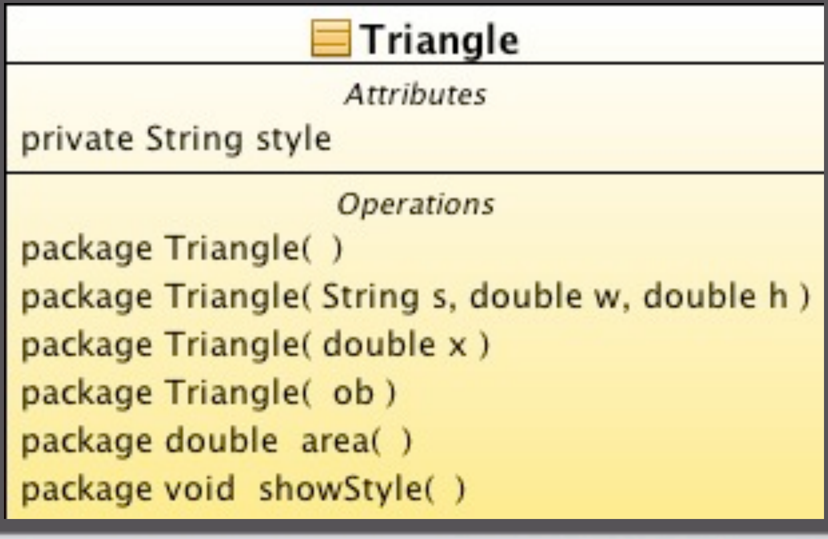
package TwoDShape( )
package TwoDShape(double w, double h, String n)
package TwoDShape(double x, String n)
package TwoDShape(ob)
package double getWidth( )
package double getHeight( )
package void setWidth(double w)
package void setHeight(double h)
package String getName( )
package void showDim( )

Reparem que não é possível criar o método area() geral para todas as formas.

Como obrigamos todas as formas (subclasses de TwoDShape) a criarem o método area()?

```
class Triangle extends TwoDShape{

    private String style;

    // A default constructor.
    Triangle() {
        super();
        style = "null";
    }

    // Constructor for Triangle.
    Triangle(String s, double w, double h) {
        super(w, h, "triangle");
        style = s;
    }

    // Construct an isosceles triangle.
    Triangle(double x) {
        // call superclass constructor
        super(x, "triangle");
        style = "isosceles";
    }

   // continua no próximo slide
```

```
    // continuação do slide anterior



    double area() {
        return getWidth() * getHeight() / 2;
    }

    void showStyle() {
        System.out.println("Triangle is " + style);
    }

}
```



Triangle

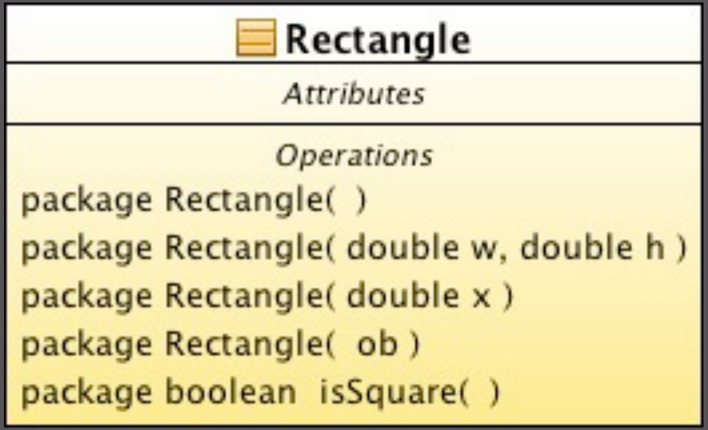
Attributes

private String style

Operations

package Triangle( )
package Triangle(String s, double w, double h)
package Triangle(double x)
package Triangle(ob)
package double area( )
package void showStyle( )

```
class Rectangle extends TwoDShape{
    // A default constructor.
    Rectangle() {
        super();
    }

    // Constructor for Rectangle.
    Rectangle(double w, double h) {
        // call superclass constructor
        super(w, h, "rectangle");
    }

    // Construct a square.
    Rectangle(double x) {
        // call superclass constructor
        super(x, "rectangle");
    }

    boolean isSquare() {
        if(getWidth() == getHeight()) return true;
        return false;
    }

 }
```



Rectangle

Attributes

Operations

package Rectangle( )

package Rectangle(double w, double h)

package Rectangle(double x)

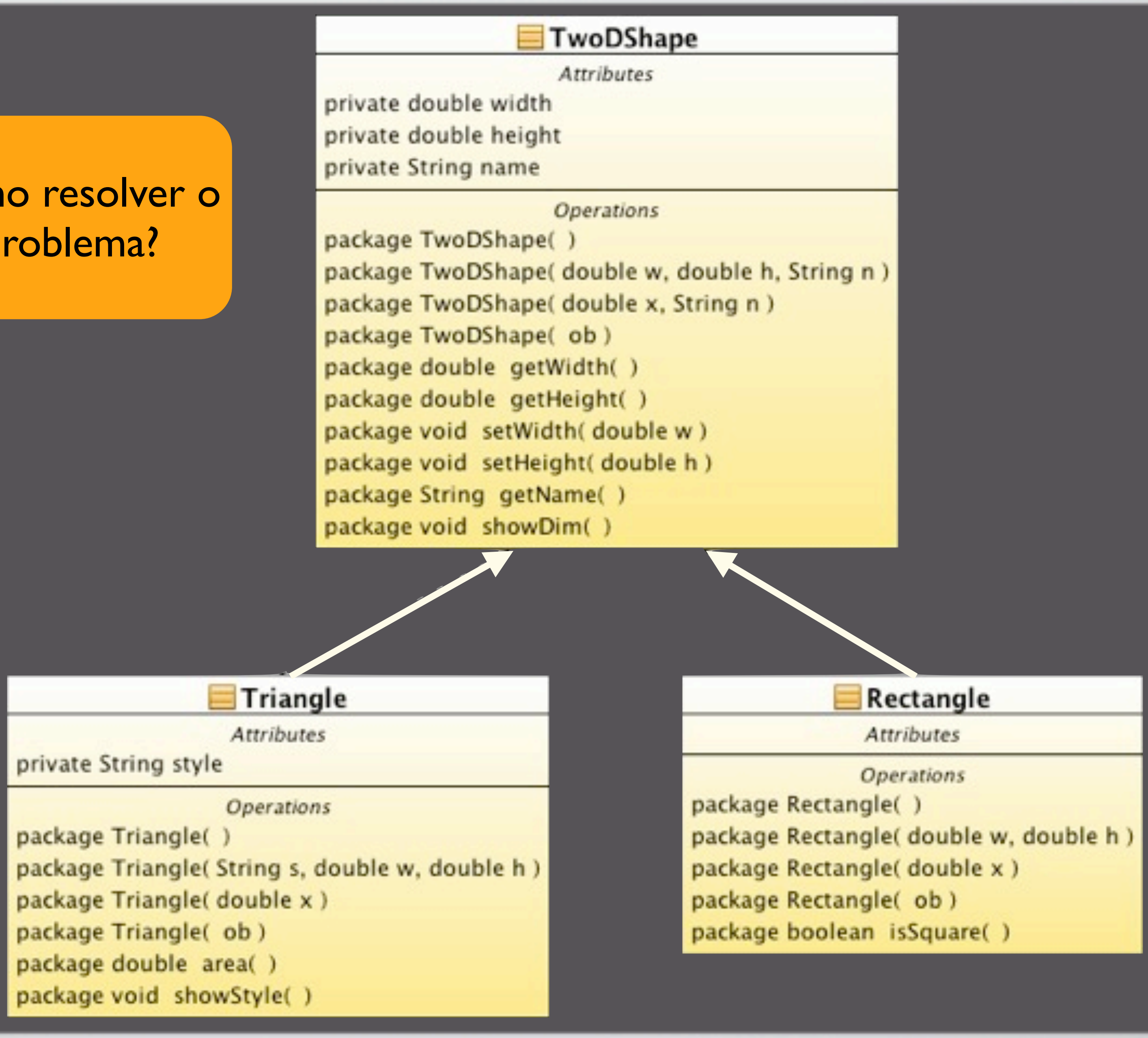
package Rectangle(ob)

package boolean isSquare( )

A classe Rectangle não tem definido o método area()

Podiamos também ter o caso de ter um método que calculasse area com um nome diferente

Como resolver o problema?



TwoDShape

Attributes

private double width
private double height
private String name

Operations

package TwoDShape( )
package TwoDShape(double w, double h, String n)
package TwoDShape(double x, String n)
package TwoDShape(ob)
package double getWidth( )
package double getHeight( )
package void setWidth(double w)
package void setHeight(double h)
package String getName( )
package void showDim( )

Triangle

Attributes

private String style

Operations

package Triangle( )
package Triangle(String s, double w, double h)
package Triangle(double x)
package Triangle(ob)
package double area( )
package void showStyle( )

Rectangle

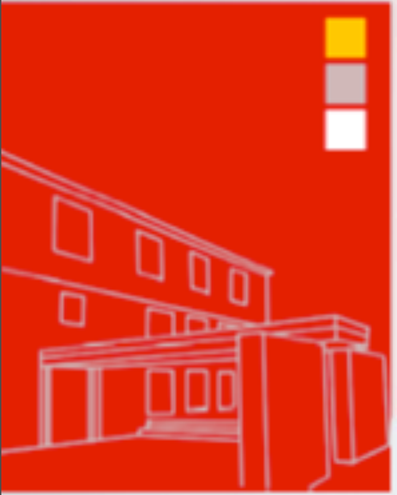
Attributes

Operations

package Rectangle( )
package Rectangle(double w, double h)
package Rectangle(double x)
package Rectangle(ob)
package boolean isSquare( )

- Por vezes queremos criar uma superclasse que defina a forma geral que será partilhada por todas as suas subclasses

- Para o exemplo anterior seria necessário definir um método **`area()`** na superclasse (**`TwoDShape`**) sem qualquer implementação que fosse obrigatório ser implementado nas suas subclasses



Classes Abstractas

- Classes abstractas são exactamente todas as classes nas quais pelo menos um ou mesmo todos os métodos de instância não se encontram implementados, mas declarados sintacticamente.
- Torna-se igualmente evidente que, por tal motivo, uma classe abstracta não pode criar instâncias.

- Uma classe abstracta, ao não implementar certos métodos, delega nas suas subclasses a implementação particular de tais métodos, facilitando o aparecimento de diferentes implementações dos mesmos métodos nas suas diferentes subclasses.

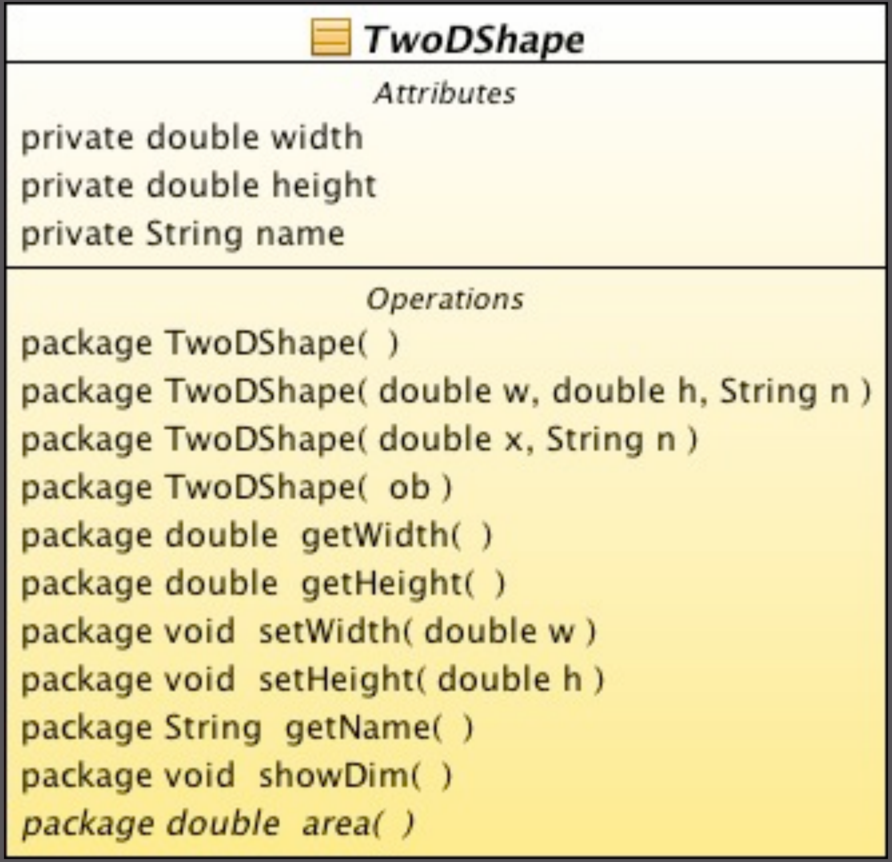
- Na relação normal entre classes e subclasses a redefinição de métodos é opcional

- A sintaxe de criação de uma classe abstracta é simples.
- Na declaração da classe usar a palavra reservada **`abstract`** seguida da palavra reservada **`class`** e do nome da classe. Para o caso da **`TwoDShape`**:

```
abstract class TwoDShape {

     /* fields and methods*/

}
```

```
abstract class TwoDShape {

    private double width;
    private double height;
    private String name;

    // A default constructor.
    TwoDShape() {
        width = height = 0.0;
        name = "null";
    }

    // Parameterized constructor.
    TwoDShape(double w, double h, String n) {
        width = w;
        height = h;
        name = n;
    }

    // Construct object with equal width and height.
    TwoDShape(double x, String n) {
        width = height = x;
        name = n;
    }

   // continua no próximo slide
```

```
    // continuação do slide anterior



    // Accessor methods for width and height.
    double getWidth() { return width; }
    double getHeight() { return height; }
    void setWidth(double w) { width = w; }
    void setHeight(double h) { height = h; }
    String getName() { return name; }

    void showDim() {
        System.out.println("Width and height are " +
                width + " and " + height);
    }

    // Now, area() is abstract.
    abstract double area();
}
```



TwoDShape

Attributes

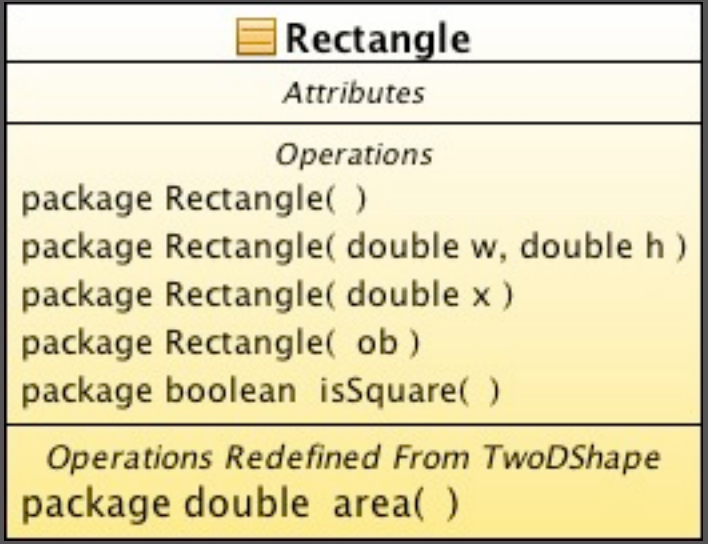
private double width
private double height
private String name

Operations

package TwoDShape( )
package TwoDShape(double w, double h, String n)
package TwoDShape(double x, String n)
package TwoDShape(ob)
package double getWidth( )
package double getHeight( )
package void setWidth(double w)
package void setHeight(double h)
package String getName( )
package void showDim( )
package double area( )

```
class Rectangle extends TwoDShape{
    // A default constructor.
    Rectangle() {
        super();
    }

    // Constructor for Rectangle.
    Rectangle(double w, double h) {
        // call superclass constructor
        super(w, h, "rectangle");
    }

    // Construct a square.
    Rectangle(double x) {
        // call superclass constructor
        super(x, "rectangle");
    }

    boolean isSquare() {
        if(getWidth() == getHeight()) return true;
        return false;
    }

   // continua no próximo slide
```

```
    // continuação do slide anterior


    double area() {
        return getWidth() * getHeight();
    }

}
```



Rectangle

Attributes

Operations

package Rectangle( )

package Rectangle(double w, double h)

package Rectangle(double x)

package Rectangle(ob)

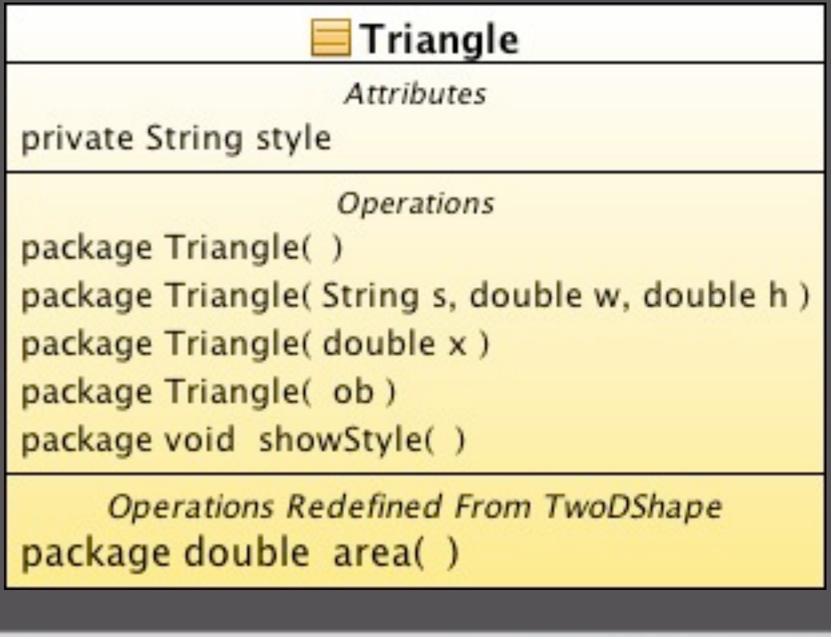
package boolean isSquare( )

Operations Redefined From TwoDShape

package double area( )

```
class Triangle extends TwoDShape{

    private String style;

    // A default constructor.
    Triangle() {
        super();
        style = "null";
    }

    // Constructor for Triangle.
    Triangle(String s, double w, double h) {
        super(w, h, "triangle");
        style = s;
    }

    // Construct an isosceles triangle.
    Triangle(double x) {
        // call superclass constructor
        super(x, "triangle");
        style = "isosceles";
    }


   // continua no próximo slide
```

```
    // continuação do slide anterior


    double area() {
        return getWidth() * getHeight() / 2;
    }

    void showStyle() {
        System.out.println("Triangle is " + style);
    }

}
```



Triangle

Attributes

private String style

Operations

package Triangle( )
package Triangle(String s, double w, double h)
package Triangle(double x)
package Triangle(ob)
package void showStyle( )

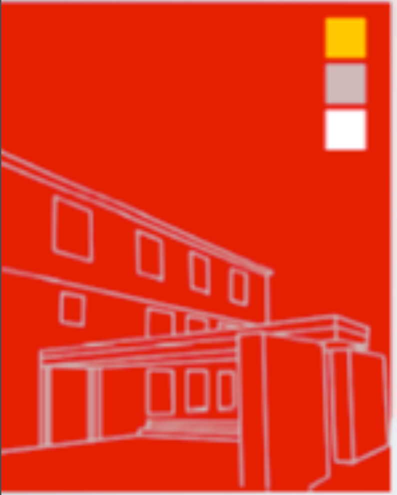
Operations Redefined From TwoDShape

package double area( )

```
public class AbsShape {

    public static void main(String[] args) {
        TwoDShape shapes[] = new TwoDShape[4];
        shapes[0] = new Triangle("right", 8.0, 12.0);
        shapes[1] = new Rectangle(10);
        shapes[2] = new Rectangle(10, 4);
        shapes[3] = new Triangle(7.0);

        for(int i=0; i < shapes.length; i++) {
            System.out.println("object is " +
            shapes[i].getName());
            System.out.println("Area is " + shapes[i].area());
            System.out.println();
        }
    }

}
```

Output:

```
   object is triangle
   Area is 48.0

   object is rectangle
   Area is 100.0

   object is rectangle
   Area is 40.0

   object is triangle
   Area is 24.5
```



Usar o `final`

- Por mais útil que seja o *overriding* e a herança por vezes poderemos querer evitar o seu uso em alguns membros
- Em java é muito fácil evitar o *overriding* de um método ou a herança de uma classe com recurso à palavra reservada `final`

O `final` evita o *Overriding*

```
class A {
   final void meth() {
      System.out.println("This is a final method.");
   }
}

class B extends A {
   void meth() { // ERROR! Can't override.
      System.out.println("Illegal!");
   }
}
```

O final evita a Herança

```
final class A {
   // ...
}


// The following class is illegal.
class B extends A { // ERROR! Can't subclass A
   // ...
}
```

O `final` para declarar constantes

```
class ErrorMsg {
   // Error codes.
   final int OUTERR   = 0;
   final int INERR    = 1;
   final int DISKERR  = 2;
   final int INDEXERR = 3;

   String msgs[] = {
      "Output Error", "Input Error",
      "Disk Full", "Index Out-Of-Bounds"
   };

   // Return the error message.
   String getErrorMsg(int i) {
   if(i >=0 & i < msgs.length)
      return msgs[i];
   else
      return "Invalid Error Code";
   }
}
```

```
class FinalD {
   public static void main(String args[]) {
      ErrorMsg err = new ErrorMsg();

      System.out.println(err.getErrorMsg(err.OUTERR));
      System.out.println(err.getErrorMsg(err.DISKERR));
   }
}
```

Palavras reservadas usadas

abstract
assert***
boolean
break
byte
case
catch
char
class
const*

continue
default
do
double
else
enum****
extends
final
finally
float

for
goto*
if
implements
import
instanceof
int
interface
long
native

new
package
private
protected
public
return
short
static
strictfp**
super

switch
synchronized
this
throw
throws
transient
try
void
volatile
while

```
   * not used
  ** added in 1.2
 *** added in 1.4
**** added in 5.0
```

Links Úteis

- http://docs.oracle.com/javase/tutorial/java/concepts/inheritance.html
- http://docs.oracle.com/javase/tutorial/java/landl/abstract.html
- http://docs.oracle.com/javase/tutorial/java/landl/final.html