DigitalReskilling

Programação Orientada a Objetos

Aula 08 - Classes Abstratas

Vitor Santos

Conteúdos

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

Palavras Reservadas

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

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

```
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();
    }
}
```

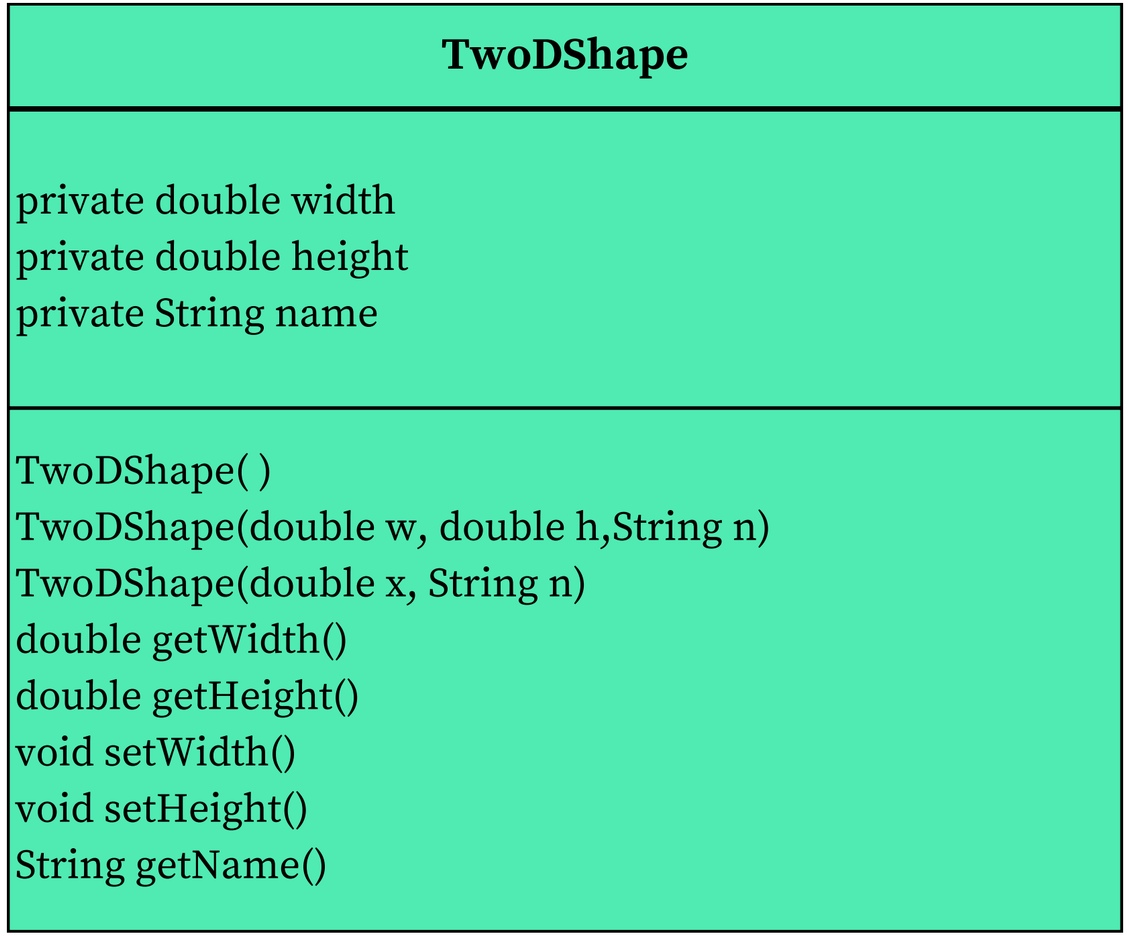
```
public class Triangle{
    private String style;
    private double width;
    private double height;
    private String name="triangle";

    Triangle(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; }

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

Rectangle
private double width private double height private String name = “rectangle”
Rectangle(double w, double h) double getWidth() double getHeight() void setWidth() void setHeight() String getName() boolean isSquare( ) double area()

Triangle
private String style private double width private double height private String name = “triangle”
Triangle (double w, double h) double getWidth() double getHeight() void setWidth() void setHeight() String getName() double area()

```
public class TwoDShape{
    private double width;
    private double height;
    private String name="triangle";

    TwoDShape( ) {
        this.width=0;
        this.height=0;
        this.name="";
    }

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

     TwoDShape(double x, String n) {
         this.width=x;
         this.height=x;
         this.name=n;
     }
```

```
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: " + this.width + " " + this.height);
    }
}
```



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

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

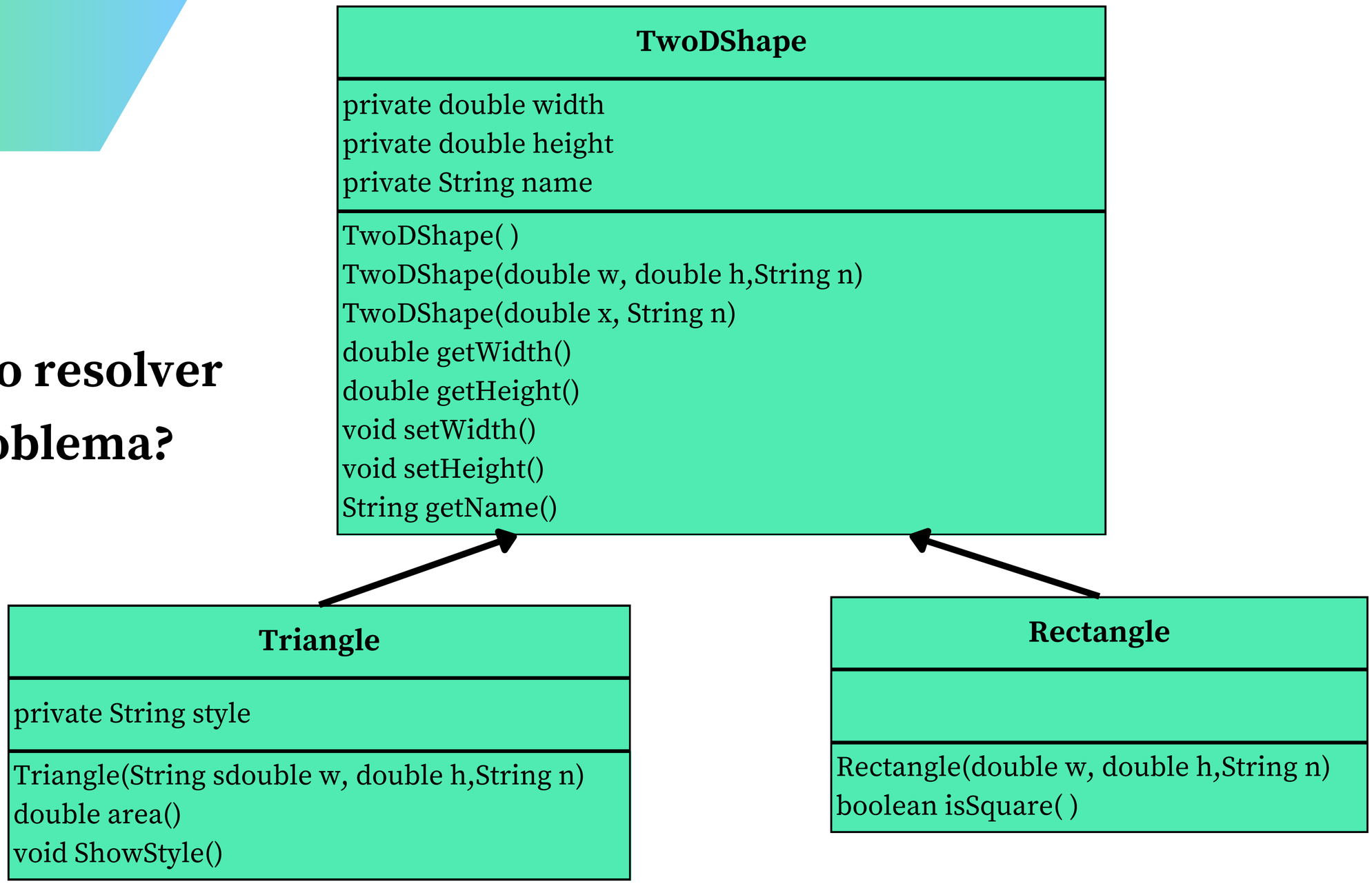
```
public class Rectangle extends TwoDShape {

    Rectangle(double w, double h) {
        super (w, h , “rectangle”)
    }

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

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.

```
public class Triangle extends TwoDShape{
   private String style;

   Triangle(String s, double w, double h) {
      super(w,h,”triangle”);
      this.style = s;
   }
   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;
   }

   void showStyle(){
      System.out.println(“”The triangle is:  “ + this.style);
   }
}
```

Como resolver o problema?



- 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 Abstratas

- Classes abstratas são exatamente todas as classes nas quais pelo menos um ou mesmo todos os métodos de instância não se encontram implementados, mas declarados sintaticamente.

- Torna-se igualmente evidente que, por tal motivo, uma classe abstrata não pode criar instâncias.

- Uma classe abstrata, 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.

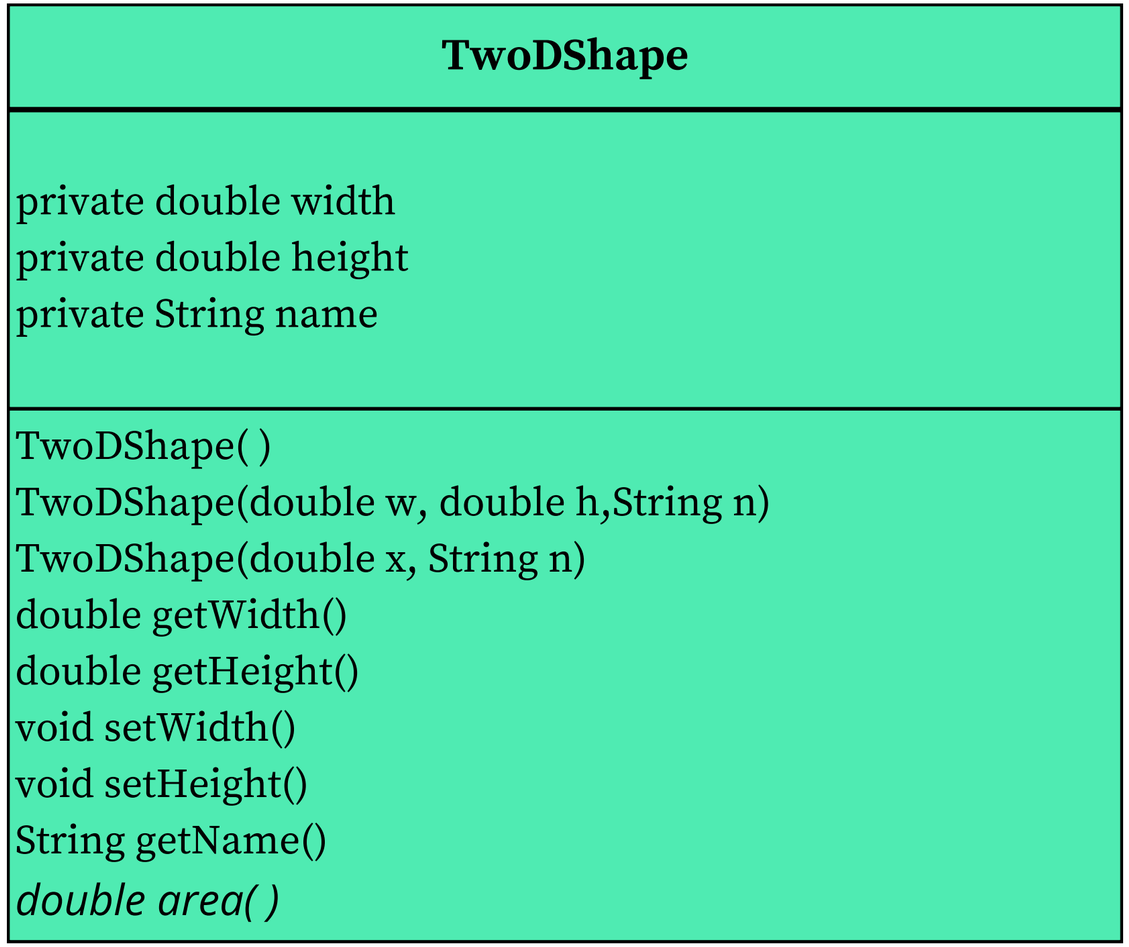
- A sintaxe de criação de uma classe abstracta é simples.
- Na declaração da classe usar a palavra reservada **abstract** seguida da palavra reservada **class** e do nome da classe. Para o caso da TwoDShape:

```
abstract class TwoDShape{
    ...
}
```

```
public abstract class TwoDShape{
    private double width;
    private double height;
    private String name="triangle";

    TwoDShape( ) {
        this.width=0;
        this.height=0;
        this.name="";
    }

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

     TwoDShape(double x, String n) {
         this.width=x;
         this.height=x;
         this.name=n;
     }
```

```
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: " + this.width + " " + this.height);
}

abstract double area( );
}
```

Agora, area( ) é um método abstrato


TwoDShape
private double width
private double height
private String name
TwoDShape( )
TwoDShape(double w, double h,String n)
TwoDShape(double x, String n)
double getWidth()
double getHeight()
void setWidth()
void setHeight()
String getName()
double area( )

```
public class Rectangle extends TwoDShape {

    Rectangle(double w, double h) {
        super (w, h , “rectangle”)
    }

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

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

Se o acesso estivesse protected em TwoDShape, poderiamos usar this.width

```
public class Triangle extends TwoDShape{
   private String style;

   Triangle(String s, double w, double h) {
      super(w,h,”triangle”);
      this.style = s;
   }
   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;
   }

   void showStyle(){
      System.out.println(“”The triangle is:  “ + this.style);
   }
}
```

```
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(10.0 , 10.0);
        shapes[2] = new Rectangle(10.0 , 4.0);
        shapes[3] = new Triangle(“isosceles”, 7.0 , 7.0);

       for(int i=0; i< shapes.length ; i++){
          System.out.println(“Objeto Atual: ” +shapes[ i ].getName( ) );
          System.out.println(“Area: ” +shapes[ i ].getArea( ) + ” \n ”);
       }

    }
}
```

Consola IDE

```
Objeto Atual: triangle
Area: 48.0

Objeto Atual: rectangle
Area: 100.0

Objeto Atual: rectangle
Area: 40.0

Objeto Atual: triangle
Area: 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 metodo(){
        System.out.println(“Método final...”);
    }
}

class B extends A {
    @Override
    void metodo(){
        System.out.println(“Ilegal!!!”);
    }
}
```

Erro! Não pode dar Override

O final evita a Herança

```
final class A {
    //
}

class B extends A {
    //
}
```

Erro! Não pode Herdar

O final para declarar constantes

```
final class A {
    private final int x = 0;
    private final int y =10;
    private final String code = “A”

    //
}
```

Palavras Reservadas

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

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