

JAVA

TEMA 15:

UTILIZACIÓN AVANZADA DE CLASES



ÍNDICE

1. Las clases finales.
2. Las clases abstractas.
3. El polimorfismo.
4. Interfaces en JAVA.
5. El operador instanceof
6. Ejercicios de consolidación



JAVA

UTILIZACIÓN AVANZADA DE CLASES

1. Las clases finales



1.- Las clases finales

- La palabra reservada **final** la hemos visto antes, ya la hemos utilizado, al principio del curso, delante de identificadores para indicar que son **constantes**, esto es, variables que mantienen un valor inmutable a lo largo de toda la vida del programa.

- *Ejemplo:*

```
package constantes;

/**
 * @author OLG
 */
public class Constantes {

    public static void main(String[] args) {
        final int DIAS_SEMANA = 7;
        final int DIAS_LABORABLES = 5;

        System.out.println("El número de días de la semana son " + DIAS_SEMANA);
        System.out.println("El número de días laborables de la semana son " + DIAS_LABORABLES);
    }
}
```

1.- Las clases finales

- *Otro ejemplo:*

```
package constantes;

/**
 * @author OLG
 */
public class Constantes {

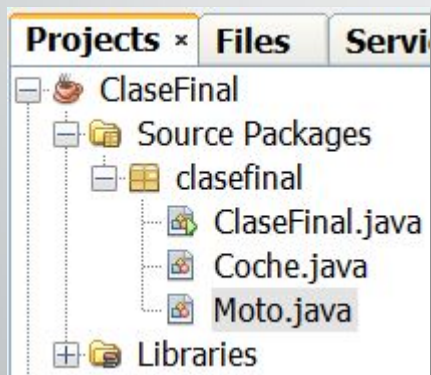
    final static int DIAS_SEMANA = 7;
    final static int DIAS_LABORABLES = 5;

    public static void main(String[] args) {
        System.out.println("El número de días de la semana son " + DIAS_SEMANA);
        System.out.println("El número de días laborables de la semana son " + DIAS_LABORABLES);
    }
}
```

1.- Las clases finales

- La palabra reservada **final** también lo podemos usar en clases y métodos.
- Cuando usamos final en una clase, significa que esa clase no puede tener clases derivadas o clases hijas, es decir, que impide la herencia.
- La sintaxis es: ***final public class nombre_clase***
- ***Ejemplo:***

1.- Las clases finales



```

1 package clasefinal;
2 /**
3  * @author OLG
4  */
5 final public class Coche {
6     String marca;
7     String modelo;
8     int velocidadMax;
9 }
    
```

```

1 package clasefinal;
2 /**
3  * @author OLG
4  */
5 public class Moto extends Coche {
6     // ...
7 }
    
```

cannot inherit from final Coche

Create Subclass

Create Test Class

(Alt-Enter shows hints)

1.- Las clases finales

- Cuando se usa **final** en un método, significa que ese método no podrá ser sobrescrito por un clase hija. La sintaxis es:
`final <modificador de acceso> tipo_dato_devuelto nombre_método (parámetros)`
- Ejemplo: `final public static void introducirDatos (int dato);`
- *Ejemplo:*

1.- Las clases finales

```

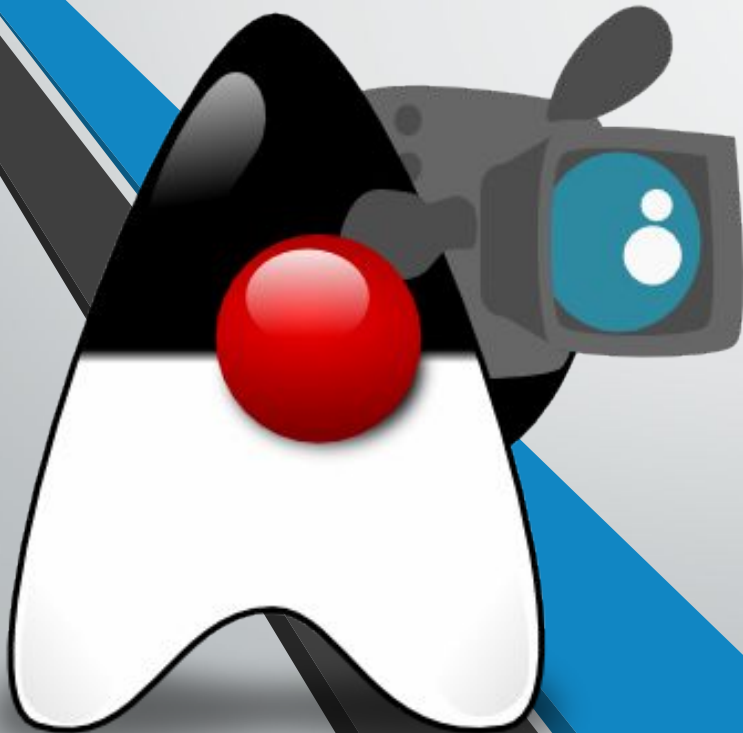
1 package metodofinal;
2 /**
3  * @author OLG
4  */
5 public class Vehiculo {
6     private String marca;
7     private int numeroRuedas;
8
9     public Vehiculo() {
10    }
11
12     public Vehiculo(String marca, int numeroRuedas) {
13         this.marca = marca;
14         this.numeroRuedas = numeroRuedas;
15     }
16
17     final public void pincharRueda() {
18         numeroRuedas--;
19     }
20
21     public int getNumeroRuedas() {
22         return numeroRuedas;
23     }
24
25     public void setNumeroRuedas(int numeroRuedas) {
26         this.numeroRuedas = numeroRuedas;
27     }
28 }
  
```

```

1 package metodofinal;
2 /**
3  * @author OLG
4  */
5 final public class Coche extends Vehiculo{
6     private int numPuertas;
7     private int velocidadMax;
8
9     public Coche() {
10        super();
11    }
12
13     public Coche(int numPuertas, int velocidadMax, String marca, int numeroRuedas) {
14         super(marca, numeroRuedas);
15         this.numPuertas = numPuertas;
16         this.velocidadMax = velocidadMax;
17     }
18
19     public int getNumPuertas() {
20         return numPuertas;
21     }
22
23     public void setNumPuertas(int numPuertas) {
24         this.numPuertas = numPuertas;
25     }
26
27     public void pincharRueda() {
28         @Override
29         public void pincharRueda() {
30         }
31     }
32
33     public int getVelocidadMax() {
34         return velocidadMax;
35     }
36 }
  
```

JAVA

UTILIZACIÓN AVANZADA DE CLASES



2. Las clases abstractas

2.- Las clases abstractas

- Una clase abstracta es una clase en la que alguno de sus métodos está declarado pero no está definido, es decir, se especifica su nombre, parámetros y tipo de devolución, pero no incluye código. A estos métodos sin código se les llama abstractos. (como a las clases)
- Además, cuando una clase es abstracta, no podemos crear objetos de estas clases.

2.- Las clases abstractas

- Para indicar que una clase es abstracta, escribimos ***abstract*** entre el modificador de acceso y class. Ejemplo:

```
public abstract class Figura
```

- Definamos un método abstracto (sin implementar), por ejemplo, el método calcularArea, para que cada clase hija OBLIGATORIAMENTE implemente su propio método de forma diferente:

```
public abstract void calcularArea (); //no lleva llaves { }
```

//también incluimos la palabra abstract (igual que en la clase)

2.- Las clases abstractas

- El uso de una clase abstracta toma sentido cuando queremos crear una clase padre que sirva como "plantilla" para las clases hijas, ya que la hijas están obligadas a sobrescribir todos los métodos abstractos que heredan.
- Veamos un ejemplo completo:

2.- Las clases abstractas

```
FiguraGeometrica.java x Triangulo.java x Circulo.java x
Source History
1 package avanzado;
2 /**
3  * @author OLG
4  */
5 public abstract class FiguraGeometrica {
6     private String color;
7
8     public FiguraGeometrica() {
9     }
10
11    public FiguraGeometrica(String color) {
12        this.color = color;
13    }
14
15    public String getColor() {
16        return color;
17    }
18
19    public void setColor(String color) {
20        this.color = color;
21    }
22
23    public abstract double calcularArea();
24    //Método abstracto.
25    //Las clases hijas estarán obligadas a implementarlo
26 }
```

```
package avanzado;
/**
 * @author OLG
 */
public class Triangulo extends FiguraGeometrica{
    private int base;
    private int altura;

    public Triangulo() {
        super();
    }

    public Triangulo(int base, int altura, String color) {
        super(color);
        this.base = base;
        this.altura = altura;
    }

    @Override
    public double calcularArea() {
        return (base*altura)/2;
    }

    public int getBase() {
        return base;
    }

    public void setBase(int base) {
        this.base = base;
    }

    public int getAltura() {
        return altura;
    }

    public void setAltura(int altura) {
        this.altura = altura;
    }
}
```

2.- Las clases abstractas

```
package avanzado;

/**
 * @author OLG
 */
public class Circulo extends FiguraGeometrica {

    private int radio;

    public Circulo() {
        super();
    }

    public Circulo(int radio, String color) {
        super(color);
        this.radio = radio;
    }

    @Override
    public double calcularArea() {
        return Math.PI*radio*radio;
    }

    public int getRadio() {
        return radio;
    }

    public void setRadio(int radio) {
        this.radio = radio;
    }
}
```

```
FiguraGeometrica.java x Triangulo.java x Circulo.java x Test.java x
Source History
1 package avanzado;
2 /**
3  * @author OLG
4  */
5 public class Test {
6
7     public static void main(String[] args) {
8         Triangulo trianguloRojo = new Triangulo (5, 10, "rojo");
9         Circulo circuloAzul = new Circulo(6, "azul");
10
11         System.out.println("Se ha creado un triangulo con los siguientes datos: ");
12         System.out.println("Base: "+trianguloRojo.getBase()+" Altura: "+trianguloRojo.getAltura());
13         System.out.println("Color: " + trianguloRojo.getColor()+" Area del triangulo: "+trianguloRojo.calcularArea());
14
15         System.out.println("\nSe ha creado un circulo con los siguientes datos: ");
16         System.out.println("Radio: "+circuloAzul.getRadio()+" Color: "+circuloAzul.getColor());
17         System.out.println("Area del circulo: "+circuloAzul.calcularArea());
18
19         //Esto daría error: FiguraGeometrica figura = new FiguraGeometrica();
20         //Ya que no podemos instanciar una clase abstracta
21     }
22 }
```

Output - Avanzado (run) x

```
run:
Se ha creado un triangulo con los siguientes datos:
Base: 5 Altura: 10
Color: rojo Area del triangulo: 25.0

Se ha creado un circulo con los siguientes datos:
Radio: 6 Color: azul
Area del circulo: 113.09733552923255
BUILD SUCCESSFUL (total time: 0 seconds)
```

JAVA

UTILIZACIÓN AVANZADA DE CLASES

3. El polimorfismo



3.- El polimorfismo

- En JAVA, es posible asignar un objeto de una clase a una variable de su superclase. Veamos un ejemplo:

Dada una variable de tipo FiguraGeometrica:

FiguraGeometrica figura;

es posible asignar a esta variable un objeto Triángulo:

figura = new Triangulo (...);

- A partir de aquí, **pueden utilizarse esta variable para invocar a aquellos métodos del objeto que también estén definidos o declarados en la superclase**, pero no a aquellos que sólo existan en la clase a la que pertenece el objeto.

3.- El polimorfismo

- Por ejemplo, puede utilizarse la variable figura para invocar a los métodos getColor() y calcularArea() del objeto Triángulo, pero no para llamar a getBase() y getAltura():

`figura.getColor(); //invoca a getColor() de Triangulo`

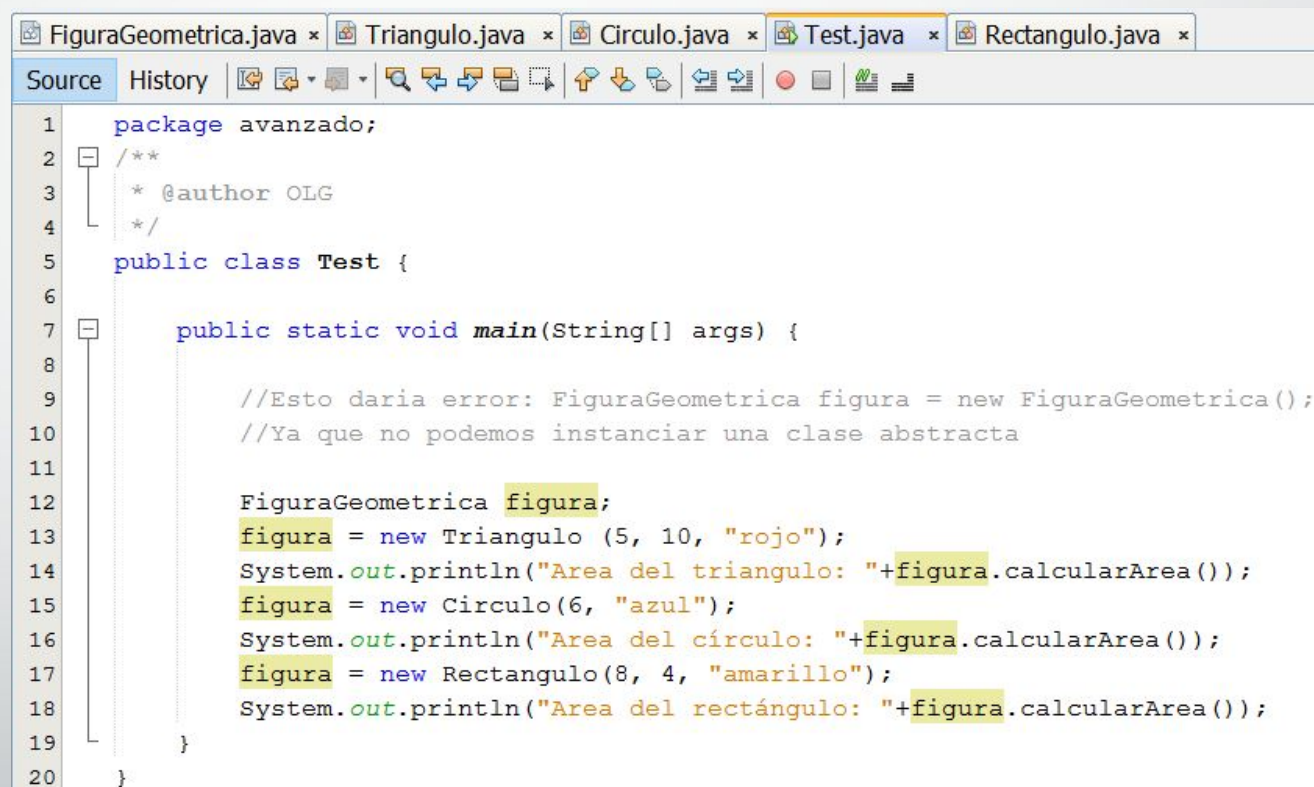
`figura.calcularArea(); //invoca a calcularArea() de Triangulo`

`figura.getBase(); // ERROR`

`figura.getAltura(); // ERROR`

3.- El polimorfismo

- Ahora, volvamos al ejemplo anterior de las clases `FiguraGeometrica`, `Triangulo`, `Circulo`, y añadimos también la clase `Rectángulo` como subclase de `FiguraGeometrica`:



```

1 package avanzado;
2 /**
3  * @author OLG
4  */
5 public class Test {
6
7     public static void main(String[] args) {
8
9         //Esto daría error: FiguraGeometrica figura = new FiguraGeometrica();
10        //Ya que no podemos instanciar una clase abstracta
11
12        FiguraGeometrica figura;
13        figura = new Triangulo (5, 10, "rojo");
14        System.out.println("Area del triangulo: "+figura.calcularArea());
15        figura = new Circulo(6, "azul");
16        System.out.println("Area del círculo: "+figura.calcularArea());
17        figura = new Rectangulo(8, 4, "amarillo");
18        System.out.println("Area del rectángulo: "+figura.calcularArea());
19    }
20 }

```

3.- El polimorfismo

- En el código anterior vemos algo muy interesante: la misma instrucción, `figura.calcularArea()`, permite llamar a distintos métodos `calcularArea()`, dependiendo del objeto almacenado en la variable `figura`.
- En esto consiste precisamente el polimorfismo: *"La posibilidad de utilizar una misma expresión para invocar a diferentes versiones de un mismo método"*

3.- El polimorfismo

- Veamos otro ejemplo:
- La principal ventaja del polimorfismo es la reutilización de código.

```
package avanzado;

/**
 * @author OLG
 */
public class Test {

    public static void mostrarColor(FiguraGeometrica figura) {
        System.out.println("El color de la figura es " + figura.getColor());
        System.out.println("El area de la figura es: "+figura.calcularArea());
    }

    public static void main(String[] args) {

        mostrarColor(new Triangulo (5, 10, "rojo"));
        mostrarColor(new Circulo(6, "azul"));
        mostrarColor(new Rectangulo(8, 4, "amarillo"));
    }
}

out - Avanzado (run) x
run:
El color de la figura es rojo
El area de la figura es: 25.0
El color de la figura es azul
El area de la figura es: 113.09733552923255
El color de la figura es amarillo
El area de la figura es: 32.0
```

3.- El polimorfismo

- Aunque no nos hayamos dado cuenta, el polimorfismo ya lo hemos utilizado en nuestros programas. Por ejemplo, hemos visto que para añadir un objeto a una colección de tipo `ArrayList` utilizamos el método `add(Object o)`. Fíjate que el parámetro recibido es de tipo `Object`, lo que significa que acepta cualquier tipo de objeto (`Object` es heredada por todas las clases)
- ***Sin la existencia del polimorfismo la clase `ArrayList` tendría que tener un método `add()` por cada tipo de objeto posible que pudiera ser añadido a la colección. (algo inviable)***

JAVA

UTILIZACIÓN AVANZADA DE CLASES

4. Interfaces en JAVA

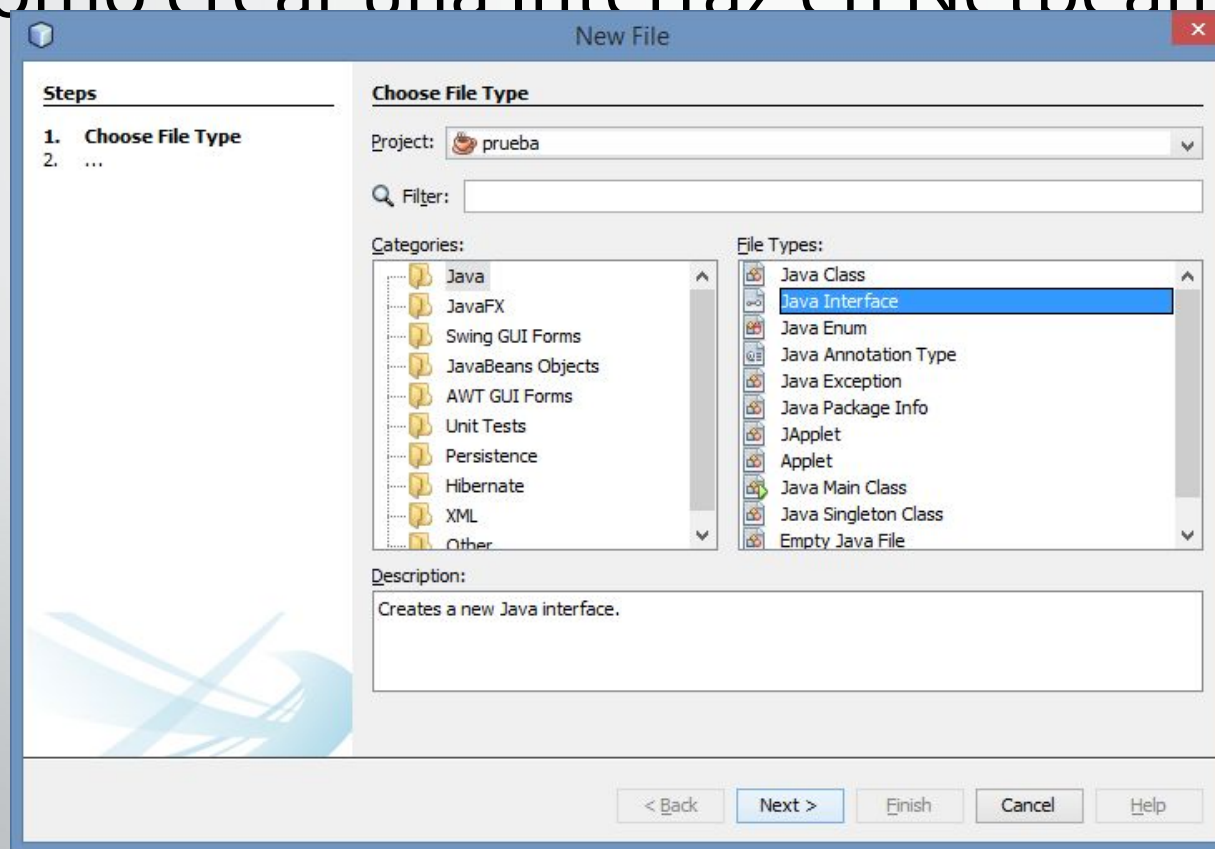


4.- Interfaces en JAVA

- Una interfaz es similar a una clase abstracta llevada al límite, en la que todos sus métodos son abstractos.
- Cuando una clase implementa una interfaz esta ha de implementar forzosamente los métodos declarados en la interfaz. La interfaz no establece lo que un método tiene que hacer y como hacerlo, sino **el formato** (nombre, parámetros y tipo de devolución) **que debe tener**.

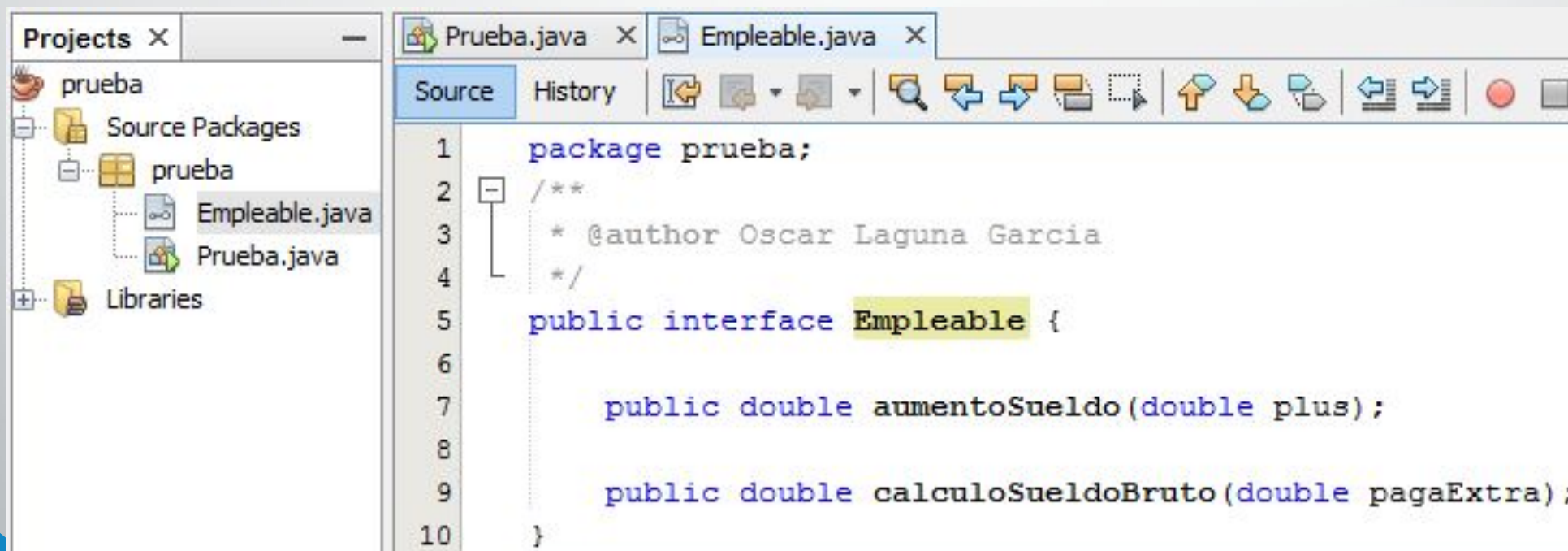
4.- Interfaces en JAVA

- Veamos como crear una Interfaz en Netbeans:



4.- Interfaces en JAVA

Ejemplo: (Fíjate que normalmente, se le suele dar un nombre acabado en -able, por ejemplo, Empleable:



```

1  package prueba;
2  /**
3   * @author Oscar Laguna Garcia
4   */
5  public interface Empleable {
6
7      public double aumentoSueldo(double plus);
8
9      public double calculoSueludoBruto(double pagaExtra);
10 }

```

4.- Interfaces en JAVA

- Para implementar una interfaz a una clase, escribimos ***implements nombre_interfaz*** después del nombre de la clase y ,si no se implementan los métodos definidos en la Interfaz, nos mostrará un error.
- Veamos un ejemplo de como quedaría:

4.- Interfaces en JAVA

```

Prueba.java x Empleable.java x
Source History
package prueba;
/**
 * @author Oscar Laguna García
 */
public class Prueba implements Empleable{
    private String nombreEmpleado;
    private double sueldoBruto;
    private double sueldoNeto;

    public Prueba(String nombreEmpleado, double sueldoBruto, double sueldoNeto) {
        this.nombreEmpleado = nombreEmpleado;
        this.sueldoBruto = sueldoBruto;
        this.sueldoNeto = sueldoNeto;
    }

    public double aumentoSueldo(double plus){
        return sueldoNeto+plus;
    }

    public double calculoSueltoBruto(double pagaExtra){
        return sueldoBruto+pagaExtra;
    }

    public static void main(String[] args) {

```

```

Prueba.java x Empleable.java x
Source History
1 package prueba;
2 /**
3  * @author Oscar Laguna García
4  */
5 public interface Empleable {
6
7     public double aumentoSueldo(double plus);
8
9     public double calculoSueltoBruto(double pagaExtra);
10 }

```


4.- Interfaces en JAVA

- Una clase pueden implementar mas de una interfaz. En ese caso deberán separarse por comas. Ejemplo:

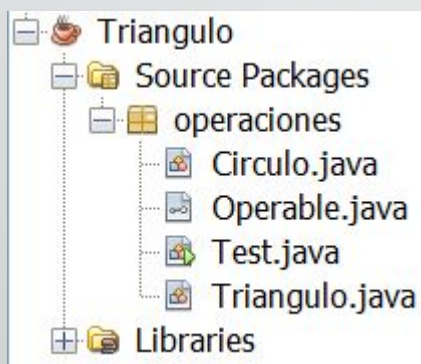
public class Prueba implements Empleable, Serializable {

- La idea de las interfaces es tener métodos comunes entre clases sin que sea necesaria la herencia (que no tienen una relación de padre-hija como pasaba con las clases abstractas). Se podría decir que es como si fueran clases *primas*.

4.- Interfaces en JAVA

- Como ya ocurriera con las clases abstractas, el principal objetivo que persiguen las interfaces con la definición de un formato común de métodos es el polimorfismo.
- Una variable de tipo interfaz puede almacenar cualquier objeto de las clases que la implementan, pudiendo utilizar esta variable para invocar a los métodos del objeto que han sido declarados en la interfaz e implementados en la clase. Veamos un ejemplo:

4.- Interfaces en JAVA



```

1 package operaciones;
2 /**
3  * @author OLG
4  */
5 public interface Operable {
6     public double calcularArea();
7 }

```

```

package operaciones;
/**
 * @author OLG
 */
public class Triangulo implements Operable {

    private int base;
    private int altura;

    public Triangulo() {
    }

    public Triangulo(int base, int altura) {
        this.base = base;
        this.altura = altura;
    }

    @Override
    public double calcularArea() {
        return (base*altura)/2;
    }

    public int getBase() {
        return base;
    }

    public void setBase(int base) {
        this.base = base;
    }
}

```

```

package operaciones;
/**
 * @author OLG
 */
public class Circulo implements Operable {

    private int radio;

    public Circulo() {
    }

    public Circulo(int radio) {
        this.radio = radio;
    }

    @Override
    public double calcularArea() {
        return Math.PI*radio*radio;
    }

    public int getRadio() {
        return radio;
    }

    public void setRadio(int radio) {
        this.radio = radio;
    }
}

```

4.- Interfaces en JAVA

```
package operaciones;

/**
 * @author OLG
 */
public class Test {

    public static void mostrarArea(Operable op) {
        System.out.println("El area de la figura es: "+op.calcularArea());
    }

    public static void main(String[] args) {
        Operable op = new Triangulo(5, 10);
        System.out.println("El area del triangulo es: " + op.calcularArea());

        mostrarArea(new Triangulo (4, 8));
        mostrarArea(new Circulo(6));
    }
}
```

4.- Interfaces en JAVA

- En la API de JAVA tenemos multitud de interfaces diseñadas para ser implementadas en las aplicaciones:
 - **java.lang.Runnable:** Contiene un método para ser implementado por aquellas aplicaciones que van a funcionar en modo multitarea.
 - **java.útil.Enumeration:** Proporciona métodos que son implementado por objetos utilizados para recorrer colecciones.
 - **java.awt.event.WindowsListener:** Proporciona métodos que deben ser implementados por las clases que van a gestionar los eventos producidos en la ventana.

4.- Interfaces en JAVA

- **java.sql.Connection:** Interfaz implementada por los objetos utilizados para manejar conexiones con bases de datos.
- **java.io.Serializable:** No contiene métodos, pero es obligatorio implementarla en aquellas clases cuyos objetos tengan que ser transferidos a algún dispositivo de almacenamiento.
- **java.lang.Comparable <Object> :** Contiene el método `compareTo` que nos va a permitir que en dicha clase se pueda realizar la comparación de objetos, permitiendo hacer ordenaciones de los mismos.

JAVA

UTILIZACIÓN AVANZADA DE CLASES

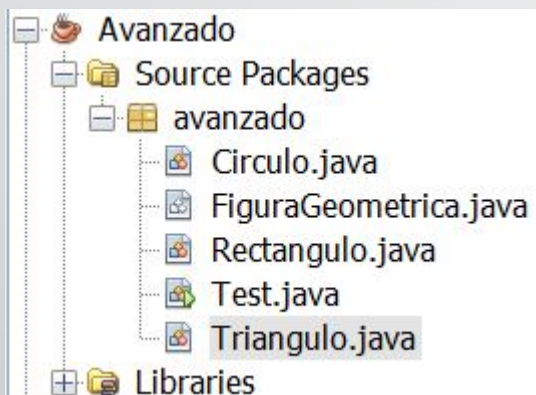


5. El operador instance of

5.- El operador instance of

- El operador ***instanceof*** nos permite comprobar si un objeto es de una clase concreta.
- Vamos a verlo con el ejemplo anterior en el que teníamos una clase abstracta *FiguraGeométrica* de la que heredaban las clases *Triangulo*, *Circulo* y *Rectángulo*:

5.- El operador instanceof



```
package avanzado;

/**
 * @author OLG
 */
public class Test {

    public static void main(String[] args) {

        FiguraGeometrica[] figuras = {new Triangulo(), new Circulo(), new Rectangulo()};
        for (int i = 0; i < figuras.length; i++) {
            if (figuras[i] instanceof Rectangulo) {
                System.out.println("El objeto en el indice " + i + " es de la clase Rectangulo");
            }
            if (figuras[i] instanceof Circulo) {
                System.out.println("El objeto en el indice " + i + " es de la clase Circulo");
            }
            if (figuras[i] instanceof Triangulo) {
                System.out.println("El objeto en el indice " + i + " es de la clase Triangulo");
            }
        }
    }
}
```

put - Avanzado (run) x

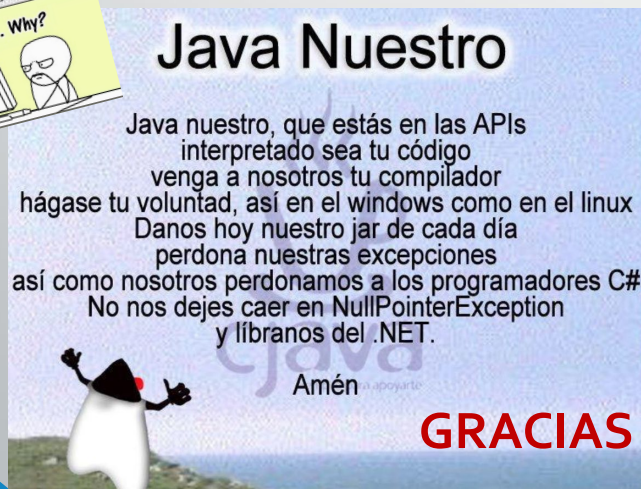
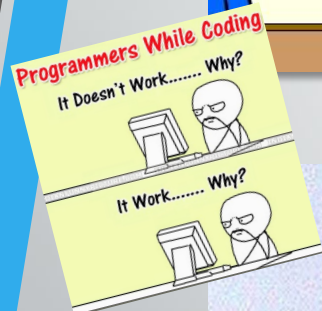
```
run:
El objeto en el indice 0 es de la clase Triangulo
El objeto en el indice 1 es de la clase Circulo
El objeto en el indice 2 es de la clase Rectangulo
```



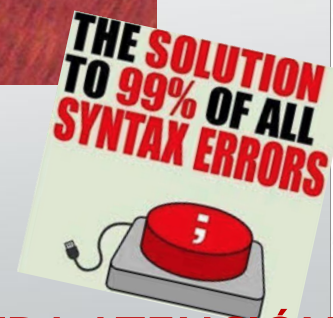

tira de linuxhispano.net

by danigm

POWERED BY TBO



FIN



GRACIAS A TODOS POR VUESTRA ATENCIÓN