Tutoriales Curso Java

Formación
Formación JAVA

Bloque Java Básico

Versión 2.3 del martes, 29 de enero de 2019

Destinatario(s)

Historial

Versión	Fecha	Origen de la actualización	Redactado por	Validado por
1.0				
2.0				
2.1				
2.2				
2.3	29/01/2019	Actualización tutoriales 6		

Índice

1. Tutorial Capítulo 3: Bloques Básicos

1.1. Tutorial 1 - Comprender la estructura de clases de Java

Con este tutorial se pretende mostrar cuál es la estructura común de una clase Java, a modo de ejemplo, se va a crear un paquete capitulo3.javaclassstructure, con una clase Dog con tres atributos (breed, age, color) y tres métodos (barking, hungry, sleeping). Más adelante, se entrará en detalle sobre los elementos que aquí se muestran: paquetes, atributos, métodos, etc

1. Se crea un paquete llamado capitulo3.javaclassstructure

```
package capitulo3.javaclassstructure;
```

2. Dentro del paquete, se crea la clase Dog

```
package capitulo3.javaclassstructure;
public class Dog {
}
```

3. Se crean tres atributos en la clase Dog

```
package capitulo3.javaclassstructure;
public class Dog {
    /*
     * Los tres atributos de la case Dog son breed, age y color
     */
    String breed;
    int age;
    String color;
}
```

4. Se crean tres métodos en la clase Dog

```
package capitulo3.javaclassstructure;
public class Dog {
    /*
     * Los tres atributos de la case Dog son breed, age y color
     */
    String breed;
    int age;
    String color;
    /*
     * Los tres métodos de la clase Dog son barking(), hungry() y sleeping()
```

```
     */
    public int barking(int bark) {
    bark = bark + 1;
    return bark;
    }
    void hungry() {
    }
    void sleeping() {
    }
}
```

5. Como se ha visto en esta sección, es conveniente comentar el código utilizando javadocs o comentarios de apoyo. Un ejemplo de los comentarios a realizar para el código anterior sería el siguiente:

```
/**
 * Copyright 2018.
 */

package capitulo3.javaclassstructure;

/**
 * Capítulo 3: Bloques básicos en Java
 *
 * Esta clase define objetos tipo Dog
 *
 * @author
 *
 */

public class Dog {
    /*
     * Atributos de la case Dog
     */
```

```
  String breed;
  int age;
  String color;

  /**
   * Método que realiza un ladrido
   *
   *  @param bark
   *             contiene el número de ladridos
   * @return bark
   *         un entero con el número de ladridos modificado
   */
  public int barking(int bark) {

bark = bark + 1;

return bark;

  }//Cierre método barking

   /**
    * Método en blanco
    */
  void hungry() {
  }//Cierre método hungry

   /**
    * Método en blanco
    */
```

```
    void sleeping() {
    }//Cierre método sleeping


}//Cierre de la clase Dog
```

1.2. Tutorial 2 - El método main()

El método main es, tal y como se ha visto en el bloque de teoría, uno de los métodos más importantes de Java. En este tutorial se muestra cómo se crea dicho método main y se le añade la sencilla funcionalidad de mostrar un mensaje por consola.

1. Se crea un paquete llamado capitulo3.mainmethod

```
package capitulo3.mainmethod;
```

2. Dentro del paquete, se crea la clase FirstJavaProgram

```
package capitulo3.mainmethod;

public class FirstJavaProgram {

}
```

3. Dentro de la clase FirstJavaProgram, se crea el método public static void main(String[] args) (método main)

```
package capitulo3.mainmethod;

public class FirstJavaProgram {

  public static void main(String[] args){

  }

}
```

4. Dentro del método main se escribe: System.out.println("This is my first program in java");

```
package capitulo3.mainmethod;

public class FirstJavaProgram {

  public static void main(String[] args){

    System.out.println("This is my first program in java");
```

```
  }
}
```

5. Output

```
This is my first program in java
```

1.3. Tutorial 3 - Paquetes e importaciones

A lo largo de este tutorial se crearán dos paquetes java. En uno de ellos se creará la clase Calculadora, a cuyos métodos se quiere acceder desde otra clase situada en un paquete diferente. Para ello, será necesario importar dicha clase, viendo así cómo funcionan los imports en Java.

1. Se crea un paquete llamado capitulo3.packagedeclarationsandimports

```
package capitulo3.packagedeclarationsandimports;
```

2. Se crea un paquete llamado letmecalculate dentro del paquete capitulo3.packagedeclarationsandimports

```
package capitulo3.packagedeclarationsandimports.letmecalculate;
```

3. Se crea una clase llamada Calculator dentro del paquete letmecalculate

```
package capitulo3.packagedeclarationsandimports.letmecalculate;
public class Calculator {

}
```

4. Se crea un método public int add que devuelva la suma de dos valores

```
package  capitulo3.packagedeclarationsandimports.letmecalculate;
public class Calculator {
  public int add(int a, int b){
     return a+b;
  }
}
```

5. Se crea una clase llamada Demo dentro del paquete capitulo3.packagedeclarationsandimports

```
package capitulo3.packagedeclarationsandimports;
public class Demo {
}
```

6. Se crea el método main

```
package capitulo3.packagedeclarationsandimports;
public class Demo {
   public static void main(final String args[]) {
   }
}
```

7. Se importa la clase Calculator del paquete capitulo3.packagedeclarationsandimports.letmecalculate

```
package capitulo3.packagedeclarationsandimports;
import capitulo3.packagedeclarationsandimports.letmecalculate.Calculator;
public class Demo {
   public static void main(final String args[]) {
   }
}
```

8. Se crea un objeto Calculator

```
package capitulo3.packagedeclarationsandimports;
import capitulo3.packagedeclarationsandimports.letmecalculate.Calculator;
public class Demo {
  public static void main(final String args[]) {
    Calculator calc= new Calculator();
  }
}
```

9. También es posible importar todas las clases del paquete capitulo3.packagedeclarationsandimports.letmecalculate utilizando la wildcard *

```
package capitulo3.packagedeclarationsandimports;
import capitulo3.packagedeclarationsandimports.letmecalculate.*;
public class Demo {
  public static void main(final String args[]) {
    Calculator calc = new Calculator();
  }
}
```

10. El método main imprime por pantalla el restultado del método sum del objeto del tipo Calculator que se acaba de crear

```
package capitulo3.packagedeclarationsandimports;
import capitulo3.packagedeclarationsandimports.letmecalculate.Calculator;
```

```
public class Demo {
  public static void main(final String args[]) {
    Calculator calc= new Calculator();
    System.out.println(calc.add(100, 200));
  }
}
```

11. Output

```
300
```

1.4. Tutorial 4 - Crear objetos

En este tutorial se pretende mostrar cómo se crean los métodos constructores de una clase Java. Para ello, se crearán dos constructores: uno de ellos sin parámetros, y el otro, con un parámetro de tipo String.

1. Se crea un paquete llamado capitulo3.creatingobjetcs

```
package capitulo3.creatingobjetcs;
```

2. Se crea una clase llamada Puppy dentro del paquete capitulo3.creatingobjetcs

```
package capitulo3.creatingobjetcs;
public class Puppy {
}
```

3. Se crean dos constructores: un constructor vacío y un constructor con un parámetro de tipo String llamado name

```
package capitulo3.creatingobjetcs;
public class Puppy {
    /*
     * Constructor vacío
     */
    public Puppy() {
    }
    /*
     * Constructor con un parámetro de tipo String llamado name
     */
    public Puppy(String name) {
        System.out.println("Passed Name is :" + name );
```

```
    }
 }
```

4. Se crea el método main

```
package capitulo3.creatingobjetcs;
public class Puppy {
    /*
     * Constructor vacío
     */
    public Puppy() {
    }
    /*
     * Constructor con un parámetro de tipo String llamado name
     */
    public Puppy(String name) {
        System.out.println("Passed Name is :" + name );
    }
    public static void main(String []args) {
    }
 }
```

5. Se crea un objeto Puppy y se usa su constructor con parámetros, al que se le pasa el parámetro name con valor "tommy"

```
package capitulo3.creatingobjetcs;
public class Puppy {
    /*
     * Constructor vacío
     */
    public Puppy() {
    }
    /*
     * Constructor con un parámetro de tipo String llamado name
     */
    public Puppy(String name) {
        System.out.println("Passed Name is :" + name );
    }
    public static void main(String []args) {
       /*
        * Se crea un objeto Puppy al que se llama myPuppy y se hace uso para ello del
constructor con parámetros
        * Es importante remarcar que siempre que se cree un objeto, se hará uso de la
palabra reservada new
```

```
        */
       Puppy myPuppy = new Puppy( "tommy" );
    }
 }
```

6. Output

```
Passed Name is :tommy
```

1.5. Tutorial 5 - Diferencia entre referencias a objetos y primitivas de datos

Existen dos tipos de variables en Java: primitivas de datos y variables referencia. A lo largo de este tutorial se pretende mostrar cuáles son las primitivas de datos de Java, así como un ejemplo de referencias a objetos, haciendo uso de la clase String.

1. Se crea un paquete llamado capitulo3.objectreferencesandprimitives

```
package capitulo3.objectreferencesandprimitives;
```

2. Dentro del paquete, se crea la clase PrimitiveDataTypes

```
package capitulo3.objectreferencesandprimitives;
public class PrimitiveDataTypes {
}
```

3. Dentro de la clase PrimitiveDataTypes, se crea el método main

```
package capitulo3.objectreferencesandprimitives;
public class PrimitiveDataTypes {
  public static void main(final String[] args) {
  }
}
```

4. Se crea un ejemplo de cada una de las primitivas de datos de Java: byte, short, long, double, float, boolean y char

```
package capitulo3.objectreferencesandprimitives;
public class PrimitiveDataTypes {
    public static void main(final String[] args) {
        byte numByte = 113;
        System.out.println(numByte);
        short numShort = 150;
        System.out.println(numShort);
```

```
        int numInt = 7;
        System.out.println(numInt);
        long numLong = -12332252626L;
        System.out.println(numLong);
        double numDouble = -42937737.9d;
        System.out.println(numDouble);
        float numFloat = 19.98f;
        System.out.println(numFloat);
        boolean b = false;
        System.out.println(b);
        char ch = 'Z';
        System.out.println(ch);
    }
}
```

5. Output:

```
113
150
7
-12332252626
-4.29377379E7
19.98
false
Z
```

6. Dentro del paquete capitulo3.objectreferencesandprimitives, se crea la clase ReferenceTypes

```
package capitulo3.objectreferencesandprimitives;
public class ReferenceTypes {
}
```

7. Se crea el método main:

```
package capitulo3.objectreferencesandprimitives;
public class ReferenceTypes {
    public static void main(final String[] args) {
    }
}
```

8. Dentro de la clase ReferenceTypes se crea una variable de un tipo de dato que no sea una primitiva de datos. Por ejemplo, String.

```
package capitulo3.objectreferencesandprimitives;
public class ReferenceTypes {
    public static void main(final String[] args) {
        String text1;
        String text2;

        text1 = "I'm text1";
        text2 = null;

        System.out.println(text1);
        System.out.println(text2);
    }
}
```

9. Output

```
I'm text1
null
```

1.6. Tutorial 6 - Declarando e inicializando variables

Existen diferentes maneras de declarar e inicializar variables en Java, por lo que el siguiente tutorial muestra un conjunto de ejemplos de ello.

1. Se crea un paquete llamado capitulo3.declaringandinitializingvariables

```
package capitulo3.declaringandinitializingvariables;
```

2. Dentro del paquete, se crea la clase DeclaringAndInitializingVariables

```
package capitulo3.declaringandinitializingvariables;
public class DeclaringAndInitializingVariables {
}
```

3. Dentro de la clase DeclaringAndInitializingVariables se crea el método main

```
package capitulo3.declaringandinitializingvariables;
public class DeclaringAndInitializingVariables {
  public static void main(final String[] args) {
  }
}
```

4. Se pretenden hacer diferentes pruebas para poder comprobar cuáles son las distintas formas de crear e inicializar variables en Java: crear una variable e inicializarla en una sentencia diferente, crear varias variables e inicializarlas en una sola línea, etc

```
package capitulo3.declaringandinitializingvariables;
public class DeclaringAndInitializingVariables {
  public static void main(final String[] args) {
        /*
         * Se crea una variable que es inicializada en una sentencia diferente
         */
        int num;
        num = 20;

        /*
         * Se crea una variable y se inicializa
         */
        char ch = 'A';
        int number = 100;

        /*
         * Se crean varias variables sin inicializar en una sola sentencia
         */
        boolean b1, b2;

        /*
         * Se crea una variable inicializada y una sin inicializar en la misma senten
cia
         */
        String s1 = "1", s2;

        /*
         * Se crean dos variables en sentencias diferentes, pero en la misma línea
         */
        int n1; int n2;
  }
}
```

1.7. Tutorial 7 - Variables en Java: Comprensión de la inicialización por defecto y el ámbito de las variables

El siguiente tutorial pretende mostrar los tres ámbitos de las variables en Java dependiendo de dónde estas sean creadas: variables de instancia, variables locales y variables de clase. Para ello, se crearán una serie de clases donde se declararán dichos tipos de variable y se comprobará cuál es el ámbito de estas.

1.7.1. Variables de instancia

1. Se crea un paquete llamado capitulo3.javavariables

```
package capitulo3.javavariables;
```

2. Dentro del paquete, se crea la clase InstanceVariables

```
package capitulo3.javavariables;
public class InstanceVariables {
}
```

3. Dentro de la clase InstanceVariables se crea una variable de instancia inicializada (myInstanceVar en este caso)

```
package capitulo3.javavariables;
public class InstanceVariables {
    /*
     * Cada uno de los objetos de la clase InstanceVariables que se cree
     * tendrá su propia variable de instancia myInstanceVar
     */
    String myInstanceVar = "instance variable";
}
```

4. Se crea el método main

```
package capitulo3.javavariables;
public class InstanceVariables {
    /*
     * Cada uno de los objetos de la clase InstanceVariables que se cree
     * tendrá su propia variable de instancia myInstanceVar
     */
    String myInstanceVar = "instance variable";
    public static void main(final String args[]) {
    }
}
```

5. Dentro del método main, se instancian una serie de objetos de la clase InstanceVariables

```
package capitulo3.javavariables;
public class InstanceVariables {
    /*
     * Cada uno de los objetos de la clase InstanceVariables que se cree
     * tendrá su propia variable de instancia myInstanceVar
     */
    String myInstanceVar = "instance variable";
    public static void main(final String args[]) {
        InstanceVariables obj = new InstanceVariables();
        InstanceVariables obj2 = new InstanceVariables();
        InstanceVariables obj3 = new InstanceVariables();
    }
}
```

6. Se imprime por pantalla la variable myInstanceVar para cada uno de los objetos

```
package capitulo3.javavariables;
public class InstanceVariables {
    /*
     * Cada uno de los objetos de la clase InstanceVariables que se cree
     * tendrá su propia variable de instancia myInstanceVar
     */
    String myInstanceVar = "instance variable";
    public static void main(final String args[]) {
        InstanceVariables obj = new InstanceVariables();
        InstanceVariables obj2 = new InstanceVariables();
        InstanceVariables obj3 = new InstanceVariables();

        System.out.println(obj.myInstanceVar);
        System.out.println(obj2.myInstanceVar);
        System.out.println(obj3.myInstanceVar);
    }
}
```

7. Se cambia la variable myInstanceVar de uno de los objetos (obj2 en este caso)

```
package capitulo3.javavariables;
public class InstanceVariables {
    /*
     * Cada uno de los objetos de la clase InstanceVariables que se cree
     * tendrá su propia variable de instancia myInstanceVar
     */
    String myInstanceVar = "instance variable";
```

```
    public static void main(final String args[]) {
        InstanceVariables obj = new InstanceVariables();
        InstanceVariables obj2 = new InstanceVariables();
        InstanceVariables obj3 = new InstanceVariables();


        System.out.println(obj.myInstanceVar);
        System.out.println(obj2.myInstanceVar);
        System.out.println(obj3.myInstanceVar);


        /*
         * Se cambia la variable de instancia myInstanceVar del objeto obj2,
         * pero no del resto.
         */
        obj2.myInstanceVar = "Changed Text";
    }
}
```

8. Al imprimir de nuevo por pantalla las variables myInstanceVar de cada uno de los objetos, se podrá ver como la variable de obj2 es ahora diferente, ya que ha sido modificada previamente.

```
package capitulo3.javavariables;
public class InstanceVariables {
    /*
     * Cada uno de los objetos de la clase InstanceVariables que se cree
     * tendrá su propia variable de instancia myInstanceVar
     */
    String myInstanceVar = "instance variable";
    public static void main(final String args[]) {
        InstanceVariables obj = new InstanceVariables();
        InstanceVariables obj2 = new InstanceVariables();
        InstanceVariables obj3 = new InstanceVariables();


        System.out.println(obj.myInstanceVar);
        System.out.println(obj2.myInstanceVar);
        System.out.println(obj3.myInstanceVar);


        /*
         * Se cambia la variable de instancia myInstanceVar del objeto obj2,
         * pero no del resto.
         */
        obj2.myInstanceVar = "Changed Text";
```

```
        System.out.println(obj.myInstanceVar);
        System.out.println(obj2.myInstanceVar);
        System.out.println(obj3.myInstanceVar);
    }
}
```

9. Output

```
instance variable
instance variable
instance variable
instance variable
Changed Text

instance variable
```

1.7.2. Variables locales

10. Dentro del paquete capitulo3.javavariables, se crea la clase LocalVariables

```
package capitulo3.javavariables;
public class LocalVariables {
}
```

11. Dentro de la clase LocalVariables se crea una variable de instancia inicializada (myVar en este caso)

```
package capitulo3.javavariables;
public class LocalVariables {
    /*
     * Variable de instancia myVar
     */
    public String myVar = "instance variable";
}
```

12. Se crea un método (myMethod()) que crea una variable local también llamada myVar y se inicializa

```
package capitulo3.javavariables;
public class LocalVariables {
    /*
     * Variable de instancia myVar
     */
    public String myVar = "instance variable";
```

```
    public void myMethod() {
        /*
         * Variable local myVar
         */
        String myVar = "Inside Method";
        System.out.println(myVar);
    }
}
```

13. Se crea el método main

```
package capitulo3.javavariables;
public class LocalVariables {
    /*
     * Variable de instancia myVar
     */
    public String myVar = "instance variable";
    public void myMethod() {
        /*
         * Variable local myVar
         */
        String myVar = "Inside Method";
        System.out.println(myVar);
    }
    public static void main(final String args[]) {
    }
}
```

14. Dentro del método main, se instancian un objeto de la clase LocalVariables

```
package capitulo3.javavariables;
public class LocalVariables {
    /*
     * Variable de instancia myVar
     */
    public String myVar = "instance variable";
    public void myMethod() {
        /*
         * Variable local myVar
         */
        String myVar = "Inside Method";
        System.out.println(myVar);
    }
    public static void main(final String args[]) {
```

```
        /*
         * Se crea un objeto LocalVariables llamado obj
         */
        LocalVariables obj = new LocalVariables();
    }
}
```

15. Se llama al método myMethod(), que imprime por pantalla el valor de la variable local myVar

```
package capitulo3.javavariables;
public class LocalVariables {
    /*
     * Variable de instancia myVar
     */
    public String myVar = "instance variable";
    public void myMethod() {
        /*
         * Variable local myVar
         */
        String myVar = "Inside Method";
        System.out.println(myVar);
    }
    public static void main(final String args[]) {
        /*
         * Se crea un objeto LocalVariables llamado obj
         */
        LocalVariables obj = new LocalVariables();
        System.out.println("Calling Method");
        obj.myMethod();
    }
}
```

16. Se imprime por pantalla el valor de la variable de instancia myVar

```
package capitulo3.javavariables;
public class LocalVariables {
    /*
     * Variable de instancia myVar
     */
    public String myVar = "instance variable";
    public void myMethod() {
        /*
         * Variable local myVar
         */
```

```
        String myVar = "Inside Method";
        System.out.println(myVar);
    }
    public static void main(final String args[]) {
        /*
         * Se crea un objeto LocalVariables llamado obj
         */
        LocalVariables obj = new LocalVariables();
        System.out.println("Calling Method");
        obj.myMethod();
        System.out.println(obj.myVar);
    }
}
```

17. Output

```
Calling Method
Inside Method
instance variable
```

1.7.3. Variables de clase

18. Dentro del paquete capitulo3.javavariables, se crea la clase ClassVariables

```
package capitulo3.javavariables;
public class ClassVariables {
}
```

19. Dentro de la clase ClassVariables se crea una variable de clase o estática inicializada (myClassVar en este caso)

```
package capitulo3.javavariables;
public class ClassVariables {
    /*
     * Cada uno de los objetos de la clase ClassVariables que se cree
     * tendrá la misma variable myClassVar
     */
    public static String myClassVar="class or static variable";
}
```

20. Se crea el método main

```
package capitulo3.javavariables;
public class ClassVariables {
```

```
    /*
     * Cada uno de los objetos de la clase ClassVariables que se cree
     * tendrá la misma variable myClassVar
     */
    public static String myClassVar="class or static variable";


    public static void main(String args[]){
    }
}
```

21. Dentro del método main, se instancian una serie de objetos de la clase ClassVariables

```
package capitulo3.javavariables;
public class ClassVariables {
    /*
     * Cada uno de los objetos de la clase ClassVariables que se cree
     * tendrá la misma variable myClassVar
     */
    public static String myClassVar="class or static variable";


    public static void main(String args[]){
       ClassVariables obj = new ClassVariables();
       ClassVariables obj2 = new ClassVariables();
       ClassVariables obj3 = new ClassVariables();
    }
}
```

22. Se imprime por pantalla la variable myClassVar para cada uno de los objetos

```
package capitulo3.javavariables;
public class ClassVariables {
    /*
     * Cada uno de los objetos de la clase ClassVariables que se cree
     * tendrá la misma variable myClassVar
     */
    public static String myClassVar="class or static variable";


    public static void main(String args[]){
       ClassVariables obj = new ClassVariables();
       ClassVariables obj2 = new ClassVariables();
       ClassVariables obj3 = new ClassVariables();
```

```
        System.out.println(obj.myClassVar);
        System.out.println(obj2.myClassVar);
        System.out.println(obj3.myClassVar);
    }
}
```

23. Se cambia la variable myClassVar de uno de los objetos (obj2 en este caso)

```
package capitulo3.javavariables;
public class ClassVariables {
    /*
     * Cada uno de los objetos de la clase ClassVariables que se cree
     * tendrá la misma variable myClassVar
     */
    public static String myClassVar="class or static variable";

    public static void main(String args[]){
        ClassVariables obj = new ClassVariables();
        ClassVariables obj2 = new ClassVariables();
        ClassVariables obj3 = new ClassVariables();

        System.out.println(obj.myClassVar);
        System.out.println(obj2.myClassVar);
        System.out.println(obj3.myClassVar);

        /*
         * Se cambia la variable de instancia myClassVar del objeto obj2,
         * y por tanto, puesto que es única, cambiará también para obj y obj3
         */
        obj2.myClassVar = "Changed Text";
    }
}
```

24. Al imprimir de nuevo por pantalla las variables myClassVar de cada uno de los objetos, se podrá ver como el resultado ha cambiado para todos los objetos a pesar de haber modificado solo myClassVar en obj2. Esto se debe a que es una variable estática o de clase

```
package capitulo3.javavariables;
public class ClassVariables {
    /*
     * Cada uno de los objetos de la clase ClassVariables que se cree
```

```
     * tendrá la misma variable myClassVar
     */
    public static String myClassVar="class or static variable";


    public static void main(String args[]){
       ClassVariables obj = new ClassVariables();
       ClassVariables obj2 = new ClassVariables();
       ClassVariables obj3 = new ClassVariables();


       System.out.println(obj.myClassVar);
       System.out.println(obj2.myClassVar);
       System.out.println(obj3.myClassVar);


       /*
        * Se cambia la variable de instancia myClassVar del objeto obj2,
        * y por tanto, puesto que es única, cambiará también para obj y obj3
        */
       obj2.myClassVar = "Changed Text";


       System.out.println(obj.myClassVar);
       System.out.println(obj2.myClassVar);
       System.out.println(obj3.myClassVar);
    }
}
```

25. Output

```
class or static variable
class or static variable
class or static variable
Changed Text
Changed Text
Changed Text
```

1.8. Tutorial 8 - Ordenando elementos en una clase

Este tutorial pretende hacer ver cuál es el orden correcto de los elementos de una clase Java. Esta estructura debe de mantenerse siempre, y merece la pena prestar especial atención a los comentarios para poder ver dónde se ha de escribir cada parte del código.

1. El siguiente código muestra los elementos básicos de una clase Java:

```
/*
 * Declaración del package (primera línea de la clase).
 */
package capitulo3.orderingelementsinaclass;


/*
 * Imports, después de la declaración del paquete.
 */
import java.util.Calendar;
import java.util.Date;


/*
 * Capítulo 3: Bloques básicos en Java
 *
 * @author
 *
 *         Esta clase pretende mostrar la estructura de una clase Java sencilla.
 *         Como se puede ver, los comentarios pueden estar en cualquier parte de la clase.
 *
 */


/*
 * Declaración de la clase
 */
public class Website {
    /*
     * Variables de instancia
     */
    String webName;
    int webAge;


    /*
     * Constructor de la clase
     */
    Website(String name, int age){
       this.webName = name;
       this.webAge = age;
    }

    /*
```

```
     * Métodos de la clase
     */
    public Date fechaActual() {
        Calendar c1 = Calendar.getInstance();
        return c1.getTime();
    }
}
```

1.9. Tutorial 9 - Destruyendo Objetos

En este tutorial se pretende mostrar cómo funciona el Garbage Collector, el cual elimina de forma automática de la memoria objetos creados ya no referenciados en ninguna parte del código.

1. Se crea un paquete llamado capitulo3.destroyingobjects

```
package capitulo3.destroyingobjects;
```

2. Dentro del paquete, se crea la clase DestroyingObjects

```
package capitulo3.destroyingobjects;
public class DestroyingObjects {
}
```

3. Dentro de la clase DestroyingObjects se crea el método main

```
package capitulo3.destroyingobjects;
public class DestroyingObjects {
  public static void main(final String args[]) {
  }
}
```

4. Se crea un objeto DestroyingObjects obj al que posteriormente se le asigna un valor NULL. Este es por lo tanto eliminado por el Garbage Collector.

```
package capitulo3.destroyingobjects;
public class DestroyingObjects {
  public static void main(final String args[]) {
        DestroyingObjects obj = new DestroyingObjects();
        /*
         * El objeto al que hace referencia obj es destruído y recogido por e
l GC
         */
        obj = null;
```

```
  }
}
```

5. Se craen dos objetos DestroyingObjects a y b. Posteriormente, la referencia de b se actualiza para que apunte a a, por lo tanto, el objeto creado en DestroyingObjects b = new DestroyingObjects() es eliminado por el Garbage Collector.

```
package capitulo3.destroyingobjects;
public class DestroyingObjects {
  public static void main(final String args[]) {
        DestroyingObjects obj = new DestroyingObjects();
        /*
         * El objeto al que hace referencia obj es destruído y recogido por el GC
         */
        obj = null;


        DestroyingObjects a = new DestroyingObjects();

        DestroyingObjects b = new DestroyingObjects();


        /*

         * El objeto al que apunaba b es recogido por el GB al cambiar su referencia
al objeto al que apunta a

         */

        b = a;

  }

}
```

6. Se invoca al Garbage Collector

```
package capitulo3.destroyingobjects;
public class DestroyingObjects {
  public static void main(final String args[]) {
        DestroyingObjects obj = new DestroyingObjects();
        /*
         * El objeto al que hace referencia obj es destruído y recogido por el GC
         */
        obj = null;
```

```
        DestroyingObjects a = new DestroyingObjects();
        DestroyingObjects b = new DestroyingObjects();

        /*
         * El objeto al que apunaba b es recogido por el GB al cambiar su referencia
al objeto
         * al que apunta a
         */
        b = a;

        /*
         * Se invoca al Garbage Collector
         */
        System.gc();
  }
}
```

7. Cada vez que el GC deshecha un objeto, el método finalize() es invocado.

```
package capitulo3.destroyingobjects;
public class DestroyingObjects {
  public static void main(final String args[]) {
        DestroyingObjects obj = new DestroyingObjects();
        /*
         * El objeto al que hace referencia obj es destruído y recogido por el GC
         */
        obj = null;

        DestroyingObjects a = new DestroyingObjects();
        DestroyingObjects b = new DestroyingObjects();

        /*
         * El objeto al que apunaba b es recogido por el GB al cambiar su referencia
al objeto
         * al que apunta a
         */
         b = a;

         /*
          * Se invoca al Garbage Collector
```

```
          */
         System.gc();
  }


    @Override
    protected void finalize() throws Throwable {
        System.out.println("Garbage collection is performed by JVM");
    }
}
```

8. Output

```
Garbage collection is performed by JVM
Garbage collection is performed by JVM
```

2. Tutorial Capítulo 4: Operadores y sentencias

2.1. Tutorial 1 - Trabajando con operadores binarios aritméticos - Operadores aritméticos

En este tutorial se pretende mostrar cuáles son los operadores aritméticos de Java haciendo un uso práctico de ellos.

1. Se crea un paquete llamado capitulo4.basicoperators

```
package capitulo4.basicoperators;
```

2. Dentro se crea una clase BasicOperators

```
package capitulo4.basicoperators;
public class BasicOperators {
}
```

3. Se crea un método main

```
package capitulo4.basicoperators;
public class BasicOperators {

  public static void main(String args[]) {

  }
}
```

4. Se inicializan 2 variables int

```
package capitulo4.basicoperators;
public class BasicOperators {

  public static void main(String args[]) {

    int num1 = 100;

    int num2 = 20;

  }
}
```

5. Se muestra por pantalla las operaciones básicas de aritmética

```
package capitulo4.basicoperators;
public class BasicOperators {

  public static void main(String args[]) {

    int num1 = 100;

    int num2 = 20;


    System.out.println("num1 + num2: " + (num1 + num2));

    System.out.println("num1 - num2: " + (num1 - num2));

    System.out.println("num1 * num2: " + num1 * num2);

    System.out.println("num1 / num2: " + num1 / num2);

    System.out.println("num1 % num2: " + num1 % num2);

  }
}
```

6. Output

```
num1 + num2: 120
num1 - num2: 80
num1 * num2: 2000
num1 / num2: 5
num1 % num2: 0
```

2.2. Tutorial 2 - Trabajando con operadores unarios - Operadores logicos (5 min)

En este tutorial se pretende mostrar cuáles son los operadores lógicos de Java haciendo un uso práctico de ellos.

1. Se crea un paquete llamado capitulo4.logicaloperators

```
package capitulo4.logicaloperators;
```

2. Dentro se crea una clase LogicalOperators

```
package capitulo4.logicaloperators;

public class LogicalOperators {

}
```

3. Se crea un método main

```
package capitulo4.logicaloperators;

public class LogicalOperators {

  public static void main(String args[]) {

  }

}
```

4. Se inicializan 2 variables boolean, una a false y la otra a true

```
package capitulo4.logicaloperators;

public class LogicalOperators {

  public static void main(String args[]) {

    boolean b1 = true;

    boolean b2 = false;

  }

}
```

5. Se muestra por pantalla la comparación de las dos variables con todos los operadores lógicos

```
package capitulo4.logicaloperators;

public class LogicalOperators {

  public static void main(String args[]) {

    boolean b1 = true;
```

```
    boolean b2 = false;


    System.out.println("b1 && b2: " + (b1&&b2));
    System.out.println("b1 || b2: " + (b1||b2));
    System.out.println("!(b1 && b2): " + !(b1&&b2));
    System.out.println("b1 & b2: "+ (b1 & b2));
    System.out.println("b1 | b2: "+ (b1 | b2));
    System.out.println("b1 ^ b2: "+ (b1 ^ b2));
  }
}
```

6. Output

```
b1 && b2: false
b1 || b2: true
!(b1 && b2): true
b1 & b2: false
b1 | b2: true
b1 ^ b2: true
```

2.3. Tutorial 3 - Trabajando con operadores unarios - Operadores de incremento

En este tutorial se pretende mostrar cuáles son los operadores de incremento de Java haciendo un uso práctico de ellos.

1. Se crea un paquete llamado capitulo4.autoincrementoperators

```
package capitulo4.autoincrementoperators;
```

2. Dentro se crea una clase AutoincrementOperators

```
package capitulo4.autoincrementoperators;

public class AutoincrementOperators {

}
```

3. Se crea un método main

```
package capitulo4.autoincrementoperators;

public class AutoincrementOperators {
```

```
  public static void main(String args[]){
  }
}
```

4. Se inicializan 2 variables int

```
package capitulo4.autoincrementoperators;
public class AutoincrementOperators {
  public static void main(String args[]){
    int num1=100;
    int num2=200;
  }
}
```

5. Se auto-incrementa la primera variable

```
package capitulo4.autoincrementoperators;
public class AutoincrementOperators {
  public static void main(String args[]){
    int num1=100;
    int num2=200;
    num1++;
  }
}
```

6. Se auto-decrementa la segunda variable

```
package capitulo4.autoincrementoperators;
public class AutoincrementOperators {
  public static void main(String args[]){
    int num1=100;
    int num2=200;
    num1++;
    num2--;
```

```
  }
}
```

7. Se muestran las dos variables por pantalla

```
package capitulo4.autoincrementoperators;

public class AutoincrementOperators {

  public static void main(String args[]){

    int num1=100;

    int num2=200;

    num1++;

    num2--;

    System.out.println("num1++ is: "+num1);

    System.out.println("num2-- is: "+num2);

  }
}
```

8. Output

```
num1++ is: 101
num2-- is: 199
```

2.4. Tutorial 4 - Usando los operadores binarios adicionales - Operadores de asignación

En este tutorial se pretende mostrar cuáles son los operadores de asignación de Java haciendo un uso práctico de ellos.

1. Se crea un paquete llamado capitulo4.assignmentoperators

```
package capitulo4.assignmentoperators;
```

2. Dentro se crea una clase AssignmentOperators

```
package capitulo4.assignmentoperators;

public class AssignmentOperators {

}
```

3. Se crea un método main

```
package capitulo4.assignmentoperators;

public class AssignmentOperators {

  public static void main(String args[]) {

  }

}
```

4. Se inicializan 2 variables int

```
package capitulo4.assignmentoperators;

public class AssignmentOperators {

  public static void main(String args[]) {

    int num1 = 10;

    int num2 = 20;

  }

}
```

5. Se muestra por pantalla las operaciones de asignación y las operaciones compuestas de asignación

```
package capitulo4.assignmentoperators;

public class AssignmentOperators {

  public static void main(String args[]) {

    int num1 = 10;

    int num2 = 20;


    num2 = num1;

    System.out.println("= Output: "+num2);


    num2 += num1;

    System.out.println("+= Output: "+num2);


    num2 -= num1;
```

```
    System.out.println("-= Output: "+num2);


    num2 *= num1;
    System.out.println("*= Output: "+num2);


    num2 /= num1;
    System.out.println("/= Output: "+num2);


    num2 %= num1;
    System.out.println("%= Output: "+num2);
  }
}
```

6. Output

```
= Output: 10
+= Output: 20
-= Output: 10
*= Output: 100
/= Output: 10
%= Output: 0
```

2.5. Tutorial 5 - Usando los operadores binarios adicionales - Operadores relacionales

En este tutorial se pretende mostrar cuáles son los operadores relacionales de Java haciendo un uso práctico de ellos.

1. Se crea un paquete llamado capitulo4.comparisonoperators

```
package capitulo4.comparisonoperators;
```

2. Dentro se crea una clase ComparisonOperators

```
package capitulo4.comparisonoperators;

public class ComparisonOperators {

}
```

3. Se crea un método main

```
package capitulo4.comparisonoperators;

public class ComparisonOperators {

  public static void main(String args[]) {

  }

}
```

4. Se inicializan 2 variables int

```
package capitulo4.comparisonoperators;

public class ComparisonOperators {

  public static void main(String args[]) {

    int num1 = 10;

    int num2 = 50;

  }

}
```

5. Se realizan sentencias if donde se compara las dos variables con todos los operadores de comparación

```
package capitulo4.comparisonoperators;

public class ComparisonOperators {

  public static void main(String args[]) {

    int num1 = 10;

    int num2 = 50;


    if(num1==num2) {

      System.out.println("num1 and num2 are equal");

    }

    else{

      System.out.println("num1 and num2 are not equal");

    }
```

```
if( num1 != num2 ){
  System.out.println("num1 and num2 are not equal");
}
else{
  System.out.println("num1 and num2 are equal");
}

if( num1 > num2 ){
  System.out.println("num1 is greater than num2");
}
else{
  System.out.println("num1 is not greater than num2");
}

if( num1 >= num2 ){
  System.out.println("num1 is greater than or equal to num2");
}
else{
  System.out.println("num1 is less than num2");
}

if( num1 < num2 ){
  System.out.println("num1 is less than num2");
}
else{
  System.out.println("num1 is not less than num2");
}
```

```
    if( num1 <= num2){
      System.out.println("num1 is less than or equal to num2");
    }
    else{
      System.out.println("num1 is greater than num2");
    }
  }
}
```

6. Output

```
num1 and num2 are not equal
num1 and num2 are not equal
num1 is not greater than num2
num1 is less than num2
num1 is less than num2
num1 is less than or equal to num2
```

2.6. Tutorial 6 - Comprender las sentencias - Sentencia If-then

En este tutorial se pretende mostrar cómo funciona la sentencia if a través de una serie de ejemplos.

1. Se crea un paquete llamado capitulo4.ifthen

```
package capitulo4.ifthen;
```

2. Dentro se crea una clase IfThen

```
package capitulo4.ifthen;

public class IfThen {

}
```

3. Se crea un método main

```
package capitulo4.ifthen;

public class IfThen {

  public static void main(String args[]){

  }
```

```
}
```

4. Se inicializa una variable int

```
package capitulo4.ifthen;

public class IfThen {

  public static void main(String args[]){

    int num=70;

  }

}
```

5. Se realiza una sentencia if que compara la variable con otro valor y muestra por pantalla un mensaje cuando la comparación es verdadera

```
package capitulo4.ifthen;

public class IfThen {

  public static void main(String args[]){

    int num=70;

    if( num < 100 ){

      System.out.println("number is less than 100");

    }

  }

}
```

6. A continuación se realizará una sentencia if anidada dentro de otra. La primera sentencia compara la variable con otro valor y se muestra por pantalla un mensaje cuando la comparación es verdadera

```
package capitulo4.ifthen;

public class IfThen {

  public static void main(String args[]){

    int num=70;

    if( num < 100 ){

      System.out.println("number is less than 100");

    }
```

```
    if( num < 100 ){
      System.out.println("number is less than 100");
    }
  }
}
```

7. En la segunda sentencia if, se compara la variable con otro valor distinto al anterior y se muestra por pantalla un mensaje cuando la comparación sea cierta

```
package capitulo4.ifthen;

public class IfThen {

  public static void main(String args[]){

    int num=70;

    if( num < 100 ){

      System.out.println("number is less than 100");

    }


    if( num < 100 ){

      System.out.println("number is less than 100");

      if(num > 50){

        System.out.println("number is greater than 50");

      }

    }

  }

}
```

8. Output

```
number is less than 100
number is less than 100
number is greater than 50
```

2.7. Tutorial 7 - Comprender las sentencias - Sentencia If-then-else

En este tutorial se pretende mostrar cómo funciona la sentencia if-else a través de una serie de ejemplos.

1. Se crea un paquete llamado capitulo4.ifthenelse

```
package capitulo4.ifthenelse;
```

2. Dentro se crea una clase IfThenElse

```
package capitulo4.ifthenelse;

public class IfThenElse {

}
```

3. Se crea un método main

```
package capitulo4.ifthenelse;

public class IfThenElse {

  public static void main(String args[]){

  }

}
```

4. Se inicializa una variable int

```
package capitulo4.ifthenelse;

public class IfThenElse {

  public static void main(String args[]){

    int num=1234;

  }

}
```

5. Se realiza una sentencia if-then-else que compara la variable con otro valor y muestra por pantalla un mensaje cuando la comparación es verdadera

```
package capitulo4.ifthenelse;

public class IfThenElse {

  public static void main(String args[]){

    int num=1234;
```

```
    if( num < 50 ){
      System.out.println("num is less than 50");
    }
  }
}
```

6. Si la comparación es falsa se muestra otro mensaje por pantalla.

```
package capitulo4.ifthenelse;
public class IfThenElse {
  public static void main(String args[]){
    int num=1234;
    if( num < 50 ){
      System.out.println("num is less than 50");
    }
    else {
      System.out.println("num is greater than or equal 50");
    }
  }
}
```

7. A continuación se realiza una sentencia if-then-else, se anidan tantas sentencias if-then-else como se desea y se muestra un mensaje cuando la comparación es verdadera o falsa

```
package capitulo4.ifthenelse;
public class IfThenElse {
  public static void main(String args[]){
    int num=1234;
    if( num < 50 ){
      System.out.println("num is less than 50");
    }
    else {
      System.out.println("num is greater than or equal 50");
```

```
    }


    if(num <100 && num>=1) {
      System.out.println("Its a two digit number");
    }
    else if(num <1000 && num>=100) {
      System.out.println("Its a three digit number");
    }
    else if(num <10000 && num>=1000) {
      System.out.println("Its a four digit number");
    }
    else if(num <100000 && num>=10000) {
      System.out.println("Its a five digit number");
    }
    else {
      System.out.println("number is not between 1 & 99999");
    }
  }
}
```

8. Output

```
num is greater than or equal 50
Its a four digit number
```

2.8. Tutorial 8 - Comprender las sentencias - Sentencia switch

En este tutorial se pretende mostrar cómo funciona la sentencia switch a través de una serie de ejemplos.

1. Se crea un paquete llamado capitulo4.switchstatement

```
package capitulo4.switchstatement;
```

2. Dentro se crea una clase Switch

```
package capitulo4.switchstatement;

public class Switch {

}
```

3. Se crea un método main

```
package capitulo4.switchstatement;

public class Switch {

  public static void main(String args[]){

  }

}
```

4. Se inicializa una variable int

```
package capitulo4.switchstatement;

public class Switch {

  public static void main(String args[]){

    int i=2;

  }

}
```

5. Se realiza una sentencia switch con 4 casos y uno por defecto. En cada caso se muestra un mensaje del caso que se ha ejecutado. No hay que olvidar la anotación "break"

```
package capitulo4.switchstatement;

public class Switch {

  public static void main(String args[]){

    int i=2;


    switch(i){

      case 1:

        System.out.println("Case1 ");

        break;
```

```
      case 2:
        System.out.println("Case2 ");
        break;
      case 3:
        System.out.println("Case3 ");
        break;
      case 4:
        System.out.println("Case4 ");
        break;
      default:
        System.out.println("Default ");
    }
  }
}
```

6. Output

```
Case2
```

2.9. Tutorial 9 - Comprender las sentencias - Sentencia while

En este tutorial se pretende mostrar cómo funciona la sentencia while a través de una serie de ejemplos.

1. Se crea un paquete llamado capitulo4.whileloop

```
package capitulo4.whileloop;
```

2. Dentro se crea una clase While

```
package capitulo4.whileloop;

public class While {

}
```

3. Se crea un método main

```
package capitulo4.whileloop;

public class While {

  public static void main(String args[]){

  }

}
```

4. Se inicializa una variable int

```
package capitulo4.whileloop;

public class While {

  public static void main(String args[]){

    int i=10;

  }

}
```

5. Se realiza una sentencia while que imprime por pantalla el valor de la variable y después se decrementa en uno. El bucle se ejecutará mientras la variable sea mayor que 1

```
package capitulo4.whileloop;

public class While {

  public static void main(String args[]){

    int i=10;

    while(i>1){

      System.out.println(i);

      i--;

    }

  }

}
```

6. A continuación se crea un array de int con cuatro valores

```
package capitulo4.whileloop;

public class While {

  public static void main(String args[]){

    int i=10;

    while(i>1){

      System.out.println(i);

      i--;

    }


    int arr[]={2,11,45,9};

  }

}
```

7. Se crea una variable para recorrer el array

```
package capitulo4.whileloop;

public class While {

  public static void main(String args[]){

    int i=10;

    while(i>1){

      System.out.println(i);

      i--;

    }


    int arr[]={2,11,45,9};

    int y=0;

  }

}
```

8. Se recorre el array mediante un bucle while y la variable creada anteriormente

```
package capitulo4.whileloop;

public class While {

  public static void main(String args[]){

    int i=10;

    while(i>1){

      System.out.println(i);

      i--;

    }


    int arr[]={2,11,45,9};

    int y=0;

    while(y<4){

    }

  }

}
```

9. Se muestra el valor de las posiciones del array

```
package capitulo4.whileloop;

public class While {

  public static void main(String args[]){

    int i=10;

    while(i>1){

      System.out.println(i);

      i--;

    }


    int arr[]={2,11,45,9};

    int y=0;

    while(y<4){
```

```
      System.out.println("pos "+ y +": "+ arr[y]);
      y++;
    }
  }
}
```

10. Output

```
10
9
8
7
6
5
4
3
2
pos 0: 2
pos 1: 11
pos 2: 45
pos 3: 9
```

2.10. Tutorial 10 - Comprender las sentencias - Sentencia do-while

En este tutorial se pretende mostrar cómo funciona la sentencia do-while a través de una serie de ejemplos.

1. Se crea un paquete llamado capitulo4.whiledoloop

```
package capitulo4.whiledoloop;
```

2. Dentro se crea una clase WhileDo

```
package capitulo4.whiledoloop;

public class WhileDo {

}
```

3. Se crea un método main

```
package capitulo4.whiledoloop;
```

```
public class WhileDo {

  public static void main(String args[]){

  }

}
```

4. Se inicializa una variable int

```
package capitulo4.whiledoloop;

public class WhileDo {

  public static void main(String args[]){

    int i=10;

  }

}
```

5. Se realiza una sentencia do-while que imprime por pantalla el valor de la variable y después se decrementa en uno. El bucle se ejecutará mientras la variable sea mayor que 1

```
package capitulo4.whiledoloop;

public class WhileDo {

  public static void main(String args[]){

    int i=10;

    do{

      System.out.println(i);

      i--;

    }while(i>1);

  }

}
```

6. A continuación se crea un array de int con cuatro valores

```
package capitulo4.whiledoloop;

public class WhileDo {

  public static void main(String args[]){

    int i=10;
```

```
    do{
      System.out.println(i);
      i--;
    }while(i>1);

    int arr[]={2,11,45,9};
  }
}
```

7. Se crea una variable para recorrer el array

```
package capitulo4.whiledoloop;

public class WhileDo {
  public static void main(String args[]){
    int i=10;
    do{
      System.out.println(i);
      i--;
    }while(i>1);

    int arr[]={2,11,45,9};
    int y=0;
  }
}
```

8. Se recorre el array mediante un bucle do-while y la variable creada anteriormente

```
package capitulo4.whiledoloop;

public class WhileDo {
  public static void main(String args[]){
    int i=10;
    do{
```

```
      System.out.println(i);
      i--;
    }while(i>1);

    int arr[]={2,11,45,9};
    int y=0;
    do{
    }while(y<4);
  }
}
```

9. Se muestra el valor de las posiciones del array

```
package capitulo4.whiledoloop;

public class WhileDo {
  public static void main(String args[]){
    int i=10;
    do{
      System.out.println(i);
      i--;
    }while(i>1);

    int arr[]={2,11,45,9};
    int y=0;
    do{
      System.out.println("pos "+ y +": "+ arr[y]);
      y++;
    }while(y<4);
  }
}
```

10. Output

```
10
9
8
7
6
5
4
3
2
pos 0: 2
pos 1: 11
pos 2: 45
pos 3: 9
```

2.11. Tutorial 11 - Comprender las sentencias - Sentencia for

En este tutorial se pretende mostrar cómo funciona la sentencia for a través de una serie de ejemplos.

1. Se crea un paquete llamado capitulo4.forloop

```
package capitulo4.forloop;
```

2. Dentro se crea una clase For

```
package capitulo4.forloop;

public class For {

}
```

3. Se crea un método main

```
package capitulo4.forloop;

public class For {

  public static void main(String args[]){

  }

}
```

4. Se realiza un bucle for que imprime por pantalla el valor de la variable de inicialización y después se decrementa en uno. El bucle se ejecutará mientras la variable sea mayor que 1

```
package capitulo4.forloop;

public class For {

  public static void main(String args[]){

    for(int i=10; i>1; i--){

      System.out.println("The value of i is: "+i);

    }

  }

}
```

5. A continuación se crea un array de int con cuatro valores

```
package capitulo4.forloop;

public class For {

  public static void main(String args[]){

    for(int i=10; i>1; i--){

      System.out.println("The value of i is: "+i);

    }


    int arr[]={2,11,45,9};

  }

}
```

6. Se recorre el array mediante un bucle for

```
package capitulo4.forloop;

public class For {

  public static void main(String args[]){

    for(int i=10; i>1; i--){

      System.out.println("The value of i is: "+i);

    }
```

```
    int arr[]={2,11,45,9};
    for(int i=0; i<arr.length; i++){
    }
  }
}
```

7. Se muestra el valor de las posiciones del array

```
package capitulo4.forloop;
public class For {
  public static void main(String args[]){
    for(int i=10; i>1; i--){
      System.out.println("The value of i is: "+i);
    }

    int arr[]={2,11,45,9};
    for(int i=0; i<arr.length; i++){
      System.out.println("For: " + arr[i]);
    }
  }
}
```

8. A continuación se recorre el array creado anteriormente con un bucle forEach y se muestra por pantalla sus valores

```
package capitulo4.forloop;
public class For {
  public static void main(String args[]){
    for(int i=10; i>1; i--){
      System.out.println("The value of i is: "+i);
    }

    int arr[]={2,11,45,9};
```

```
    for(int i=0; i<arr.length; i++){
      System.out.println("For: " + arr[i]);
    }

    for (int num : arr) {
      System.out.println("forEach: " + num);
    }
  }
}
```

9. Output

```
The value of i is: 10
The value of i is: 9
The value of i is: 8
The value of i is: 7
The value of i is: 6
The value of i is: 5
The value of i is: 4
The value of i is: 3
The value of i is: 2
For: 2
For: 11
For: 45
For: 9
forEach: 2
forEach: 11
forEach: 45
forEach: 9
```

2.12. Tutorial 12 - Comprendiendo el control de flujo avanzado - Sentencia break

En este tutorial se pretende mostrar cómo funciona la sentencia break a través de una serie de ejemplos.

1. Se crea un paquete llamado capitulo4.breakstatement.

```
package capitulo4.breakstatement;
```

2. Dentro se crea una clase Break.

```
package capitulo4.breakstatement;

public class Break {

}
```

3. Se crea un método main.

```
package capitulo4.breakstatement;

public class Break {

  public static void main(String args[]){

  }

}
```

4. Se crea una variable y se inicializa.

```
package capitulo4.breakstatement;

public class Break {

  public static void main(String args[]){

    int num1 =0;

  }

}
```

5. Se realiza un bucle while que compara esa variable con un valor. Mientras la comparación sea verdadera, entrará dentro del bucle.

```
package capitulo4.breakstatement;

public class Break {

  public static void main(String args[]){

    int num1 =0;

    while(num1<=100){

    }

  }

}
```

6. Se imprime por pantalla el valor de la variable.

```
package capitulo4.breakstatement;

public class Break {

  public static void main(String args[]){

    int num1 =0;

    while(num1<=100){

      System.out.println("Value of variable is: "+num1);

    }

  }

}
```

7. Se realiza un sentencia if-then-else que la compara con otro valor.

```
package capitulo4.breakstatement;

public class Break {

  public static void main(String args[]){

    int num1 =0;

    while(num1<=100){

      System.out.println("Value of variable is: "+num1);

      if (num1==2){

      }

    }

  }

}
```

8. Si la comparación es verdadera se utiliza la sentencia break para salir del bucle while.

```
package capitulo4.breakstatement;

public class Break {

  public static void main(String args[]){

    int num1 =0;

    while(num1<=100){

      System.out.println("Value of variable is: "+num1);
```

```
      if (num1==2){
        break;
      }
    }
  }
}
```

9. Se incremente en uno el valor de la variable.

```
package capitulo4.breakstatement;

public class Break {
  public static void main(String args[]){
    int num1 =0;
    while(num1<=100){
      System.out.println("Value of variable is: "+num1);
      if (num1==2){
        break;
      }
      num1++;
    }
  }
}
```

10. Una vez salga del bucle se muestra por pantalla un mensaje de salida del bucle.

```
package capitulo4.breakstatement;

public class Break {
  public static void main(String args[]){
    int num1 =0;
    while(num1<=100){
      System.out.println("Value of variable is: "+num1);
      if (num1==2){
```

```
        break;
      }
      num1++;
    }
    System.out.println("Out of while-loop");
  }
}
```

11. A continuación, se crea una variable.

```
package capitulo4.breakstatement;

public class Break {
  public static void main(String args[]){
    int num1 =0;
    while(num1<=100){
      System.out.println("Value of variable is: "+num1);
      if (num1==2){
        break;
      }
      num1++;
    }
    System.out.println("Out of while-loop");

    int var;
  }
}
```

12. Se realiza un bucle for que compara esa variable con un valor mientras el resultado de la comparación sea verdadero.

```
package capitulo4.breakstatement;

public class Break {

  public static void main(String args[]){

    int num1 =0;

    while(num1<=100){

      System.out.println("Value of variable is: "+num1);

      if (num1==2){

        break;

      }

      num1++;

    }

    System.out.println("Out of while-loop");


    int var;

    for (var =100; var>=10; var --){

    }

  }

}
```

13. Se imprime por pantalla el valor de la variable.

```
package capitulo4.breakstatement;

public class Break {

  public static void main(String args[]){

    int num1 =0;

    while(num1<=100){

      System.out.println("Value of variable is: "+num1);

      if (num1==2){

        break;
```

```
      }
      num1++;
    }
    System.out.println("Out of while-loop");

    int var;
    for (var =100; var>=10; var --){
      System.out.println("var: "+var);
    }
  }
}
```

14. Se realiza una sentencia if-then-else que la compara con otro valor.

```
package capitulo4.breakstatement;

public class Break {
  public static void main(String args[]){
    int num1 =0;
    while(num1<=100){
      System.out.println("Value of variable is: "+num1);
      if (num1==2){
        break;
      }
      num1++;
    }
    System.out.println("Out of while-loop");

    int var;
    for (var =100; var>=10; var --){
      System.out.println("var: "+var);
      if (var==99){
```

```
      }
    }
  }
}
```

15. Si la comparación es verdadera se utiliza la sentencia break para salir del bucle for, si no lo es, se decrementa en uno el valor de la variable.

```
package capitulo4.breakstatement;

public class Break {
  public static void main(String args[]){
    int num1 =0;
    while(num1<=100){
      System.out.println("Value of variable is: "+num1);
      if (num1==2){
        break;
      }
      num1++;
    }
    System.out.println("Out of while-loop");

    int var;
    for (var =100; var>=10; var --){
      System.out.println("var: "+var);
      if (var==99){
        break;
      }
    }
  }
}
```

16. Una vez salga del bucle se muestra por pantalla un mensaje de salida del bucle.

```
package capitulo4.breakstatement;

public class Break {

  public static void main(String args[]){

    int num1 =0;

    while(num1<=100){

      System.out.println("Value of variable is: "+num1);

      if (num1==2){

        break;

      }

      num1++;

    }

    System.out.println("Out of while-loop");


    int var;

    for (var =100; var>=10; var --){

      System.out.println("var: "+var);

      if (var==99){

        break;

      }

    }

    System.out.println("Out of for-loop");

  }

}
```

17. Después, se crea una variable.

```
package capitulo4.breakstatement;

public class Break {

  public static void main(String args[]){

    int num1 =0;
```

```
    while(num1<=100){
      System.out.println("Value of variable is: "+num1);
      if (num1==2){
        break;
      }
      num1++;
    }
    System.out.println("Out of while-loop");

    int var;
    for (var =100; var>=10; var --){
      System.out.println("var: "+var);
      if (var==99){
        break;
      }
    }
    System.out.println("Out of for-loop");

    int num2=2;
  }
}
```

18. Se realiza una sentencia switch que compara esa variable con un caso del switch.

```
package capitulo4.breakstatement;

public class Break {
  public static void main(String args[]){
    int num1 =0;
    while(num1<=100){
      System.out.println("Value of variable is: "+num1);
      if (num1==2){
```

```
        break;
      }
      num1++;
    }
    System.out.println("Out of while-loop");

    int var;
    for (var =100; var>=10; var --){
      System.out.println("var: "+var);
      if (var==99){
        break;
      }
    }
    System.out.println("Out of for-loop");

    int num2=2;
    switch (num2){
    }
  }
}
```

19. Se imprime por pantalla el caso al que se ha accedido. Se utiliza la sentencia break para salir del switch.

```
package capitulo4.breakstatement;

public class Break {
  public static void main(String args[]){
    int num1 =0;
    while(num1<=100){
      System.out.println("Value of variable is: "+num1);
      if (num1==2){
```

```
      break;
    }
    num1++;
  }
  System.out.println("Out of while-loop");

  int var;
  for (var =100; var>=10; var --){
    System.out.println("var: "+var);
    if (var==99){
      break;
    }
  }
  System.out.println("Out of for-loop");

  int num2=2;
  switch (num2){
    case 1:
      System.out.println("Case 1 ");
      break;
    case 2:
      System.out.println("Case 2 ");
      break;
    case 3:
      System.out.println("Case 3 ");
      break;
    default:
      System.out.println("Default ");
  }
```

```
  }
}
```

20. Output

```
Value of variable is: 0
Value of variable is: 1
Value of variable is: 2
Out of while-loop
var: 100
var: 99
Out of for-loop
Case 2
```

2.13. Tutorial 13 - Comprendiendo el control de flujo avanzado - Sentencia continue

En este tutorial se pretende mostrar cómo funciona la sentencia continue a través de una serie de ejemplos.

1. Se crea un paquete llamado capitulo4.continuestatement

```
package capitulo4.continuestatement;
```

2. Dentro se crea una clase Continue

```
package capitulo4.continuestatement;

public class Continue {

}
```

3. Se crea un método main

```
package capitulo4.continuestatement;

public class Continue {

  public static void main(String args[]){

  }
}
```

4. Se crea una variable y se inicializa

```
package capitulo4.continuestatement;
```

```
public class Continue {

  public static void main(String args[]){

    int counter=10;

  }

}
```

5. Se realiza un bucle while que compara esa variable con un valor mientras el resultado de la comparación sea verdadero

```
package capitulo4.continuestatement;

public class Continue {

  public static void main(String args[]){

    int counter=10;

    while (counter >=0){

    }

  }

}
```

6. Se realiza un sentencia if-then que la compara con otro valor

```
package capitulo4.continuestatement;

public class Continue {

  public static void main(String args[]){

    int counter=10;

    while (counter >=0){

      if (counter==7){

      }

    }

  }

}
```

7. Si es verdadera la comparación se decrementa el valor de la variable y se ejecuta la sentencia continue para pasar a la siguiente iteración

```
package capitulo4.continuestatement;

public class Continue {

  public static void main(String args[]){

    int counter=10;

    while (counter >=0){

      if (counter==7){

        counter--;

        continue;

      }

    }

  }

}
```

8. Se imprime por pantalla el valor de la variable y se decrementa la variable

```
package capitulo4.continuestatement;

public class Continue {

  public static void main(String args[]){

    int counter=10;

    while (counter >=0){

      if (counter==7){

        counter--;

        continue;

      }

      System.out.print(counter+" ");

      counter--;

    }

  }

}
```

9. A continuación, se realiza un bucle for que compara una variable con un valor

```
package capitulo4.continuestatement;

public class Continue {

  public static void main(String args[]){

    int counter=10;

    while (counter >=0){

      if (counter==7){

        counter--;

        continue;

      }

      System.out.print(counter+" ");

      counter--;

    }


    for (int j=0; j<=6; j++){

    }

  }

}
```

10. Si la comparación es verdadera se realiza un sentencia if-then-else que la compara con otro valor

```
package capitulo4.continuestatement;

public class Continue {

  public static void main(String args[]){

    int counter=10;

    while (counter >=0){

      if (counter==7){

        counter--;

        continue;

      }

      System.out.print(counter+" ");
```

```
      counter--;
    }

    for (int j=0; j<=6; j++){
      if (j==4){
      }
    }
  }
}
```

11. Si la comparación es verdadera se utiliza la sentencia continue para pasar a la siguiente iteración del bucle for

```
package capitulo4.continuestatement;

public class Continue {
  public static void main(String args[]){
    int counter=10;
    while (counter >=0){
      if (counter==7){
        counter--;
        continue;
      }
      System.out.print(counter+" ");
      counter--;
    }

    for (int j=0; j<=6; j++){
      if (j==4){
        continue;
      }
    }
```

```
  }
}
```

12. Se imprime por pantalla el valor de la variable

```
package capitulo4.continuestatement;

public class Continue {

  public static void main(String args[]){

    int counter=10;

    while (counter >=0){

      if (counter==7){

        counter--;

        continue;

      }

      System.out.print(counter+" ");

      counter--;

    }


    for (int j=0; j<=6; j++){

      if (j==4){

        continue;

      }

      System.out.print(j+" ");

    }

  }

}
```

13. Después, se crea una variable

```
package capitulo4.continuestatement;

public class Continue {

  public static void main(String args[]){
```

```
    int counter=10;
    while (counter >=0){
      if (counter==7){
        counter--;
        continue;
      }
      System.out.print(counter+" ");
      counter--;
    }

    for (int j=0; j<=6; j++){
      if (j==4){
        continue;
      }
      System.out.print(j+" ");
    }

    int j=0;
  }
}
```

14. Se realiza un bucle do-while que compara esa variable con un valor

```
package capitulo4.continuestatement;

public class Continue {
  public static void main(String args[]){
    int counter=10;
    while (counter >=0){
      if (counter==7){
        counter--;
        continue;
```

```
      }
      System.out.print(counter+" ");
      counter--;
    }

    for (int j=0; j<=6; j++){
      if (j==4){
        continue;
      }
      System.out.print(j+" ");
    }

    int j=0;
    do{
    }while(j<10);
  }
}
```

15. Se realiza un sentencia if-then-else que la compara con otro valor

```
package capitulo4.continuestatement;

public class Continue {
  public static void main(String args[]){
    int counter=10;
    while (counter >=0){
      if (counter==7){
        counter--;
        continue;
      }
      System.out.print(counter+" ");
      counter--;
```

```
    }


    for (int j=0; j<=6; j++){
      if (j==4){
        continue;
      }
      System.out.print(j+" ");
    }


    int j=0;
    do{
      if (j==7){
      }
    }while(j<10);
  }
}
```

16. Si la comparación es verdadera se incrementa el valor de la variable en uno y se utiliza la sentencia continue para pasar a la siguiente iteración del bucle do-while

```
package capitulo4.continuestatement;

public class Continue {
  public static void main(String args[]){
    int counter=10;
    while (counter >=0){
      if (counter==7){
        counter--;
        continue;
      }
      System.out.print(counter+" ");
      counter--;
```

```
    }


    for (int j=0; j<=6; j++){
      if (j==4){
        continue;
      }
      System.out.print(j+" ");
    }


    int j=0;
    do{
      if (j==7){
        j++;
        continue;
      }
    }while(j<10);
  }
}
```

17. Se imprime por pantalla el valor de la variable y se incrementa en uno

```
package capitulo4.continuestatement;

public class Continue {
  public static void main(String args[]){
    int counter=10;
    while (counter >=0){
      if (counter==7){
        counter--;
        continue;
      }
      System.out.print(counter+" ");
```

```
      counter--;
    }


    for (int j=0; j<=6; j++){
      if (j==4){
        continue;
      }
      System.out.print(j+" ");
    }


    int j=0;
    do{
      if (j==7){
        j++;
        continue;
      }
      System.out.print(j+ " ");
      j++;
    }while(j<10);
  }
}
```

18. Output

```
10 9 8 6 5 4 3 2 1 0 0 1 2 3 5 6 0 1 2 3 4 5 6 8 9
```

2.14. Tutorial 14 - Ejercicio global - Sentencias 1

En este tutorial, vamos a ver en un primer momento como generar datos, calcular la media de números negativos y positivos y calcular la cantidad de ceros.

1. Se crea un paquete llamado capitulo4.sentences

```
package capitulo4.sentences;
```

2. Dentro se crea una clase Sentences1

```
package capitulo4.sentences;

public class Sentences1{}
```

3. Se crea un método main

```
package capitulo4.sentences;

public class Sentences1 {

        public static void main(String[] args) {}

}
```

4. Se declaran y se inicializan las variables necesarias para almacenar los contadores de ceros, negativos y positivos

```
package capitulo4.sentences;

public class Sentences1 {
   public static void main(String[] args) {
            int num;
            int cont_zeros; // the zero counter
            int cont_pos; // positive counter
            int cont_neg; //  negative counter
            int sum_pos,sum_neg; // sum of the positive and negative numbers
            float average_pos,average_neg; // positive and negative averages can have
decimal places
            cont_zeros=0;
            cont_pos=0;
            cont_neg=0;
            sum_pos=0;
            sum_neg=0;
}
}
```

5. Se realiza una sentencia for de 10 iteraciones

```
package capitulo4.sentences;

public class Sentences1 {
   public static void main(String[] args) {
```

```
            int num;
            int cont_zeros; // the zero counter
            int cont_pos; // positive counter
            int cont_neg; //  negative counter
            int sum_pos,sum_neg; // sum of the positive and negative numbers
            float average_pos,average_neg; // positive and negative averages can have
decimal places
            cont_zeros=0;
            cont_pos=0;
            cont_neg=0;
            sum_pos=0;
            sum_neg=0;
            for (int i=1;i<=10;i++) {}
       }
}
```

6. Se declara una variable num con un valor aleatorio y se muestra el valor generado

```
package capitulo4.sentences;

public class Sentences1 {
   public static void main(String[] args) {
            int num;
            int cont_zeros; // the zero counter
            int cont_pos; // positive counter
            int cont_neg; //  negative counter
            int sum_pos,sum_neg; // sum of the positive and negative numbers
            float average_pos,average_neg; //  positive and negative averages can have
decimal places
            cont_zeros=0;
            cont_pos=0;
            cont_neg=0;
            sum_pos=0;
            sum_neg=0;
            for (int i=1;i<=10;i++)
            {
                    num=(int) (Math.random() * 7) + 1;
                    System.out.println("Random number: "+ num);
                }
        }
}
```

7. Con operadores de incremento y operaciones compuestas de asignación, se incrementan los contadores.

```
package capitulo4.sentences;

public class Sentences1 {
   public static void main(String[] args) {
            int num;
            int cont_zeros; // the zero counter
            int cont_pos; // positive counter
            int cont_neg; //  negative counter
            int sum_pos,sum_neg; // sum of the positive and negative numbers
            float average_pos,average_neg; // positive and negative averages can have
decimal places
            cont_zeros=0;
            cont_pos=0;
            cont_neg=0;
            sum_pos=0;
            sum_neg=0;
            for (int i=1;i<=10;i++)
            {
                    num=(int) (Math.random() * 7) + 1;
                    System.out.println("Random number: "+ num);
                    if(num==0)
                            cont_zeros++;
                    else {
                            if(num>0)
                            {
                                    cont_pos++;
                                    sum_pos+=num;
                            } else {
                                    cont_neg++;
                                    sum_neg+=num;
                            }
                    }
            }
        }
}
```

8. Se muestran por pantalla los valores

```
package capitulo4.sentences;

public class Sentences1 {
   public static void main(String[] args) {
            int num;
```

```
            int cont_zeros; // the zero counter
            int cont_pos; // positive counter
            int cont_neg; //  negative counter
            int sum_pos,sum_neg; // sum of the positive and negative numbers
            float average_pos,average_neg; // positive and negative averages can have
decimal places
            cont_zeros=0;
            cont_pos=0;
            cont_neg=0;
            sum_pos=0;
            sum_neg=0;
            for (int i=1;i<=10;i++)
            {
                    num=(int) (Math.random() * 7) + 1;
                    System.out.println("Random number: "+ num);
                    if(num==0)
                            cont_zeros++;
                    else {
                            if(num>0)
                            {
                                    cont_pos++;
                                    sum_pos+=num;
                            } else {
                                    cont_neg++;
                                    sum_neg+=num;
                            }
                    }
            }
            // Zeros
            System.out.println("The number of zeros entered is: "+cont_zeros);
            //Positives
            if (cont_pos ==0)
                    System.out.println("You can't do the average of the positives");
            else {
                    average_pos= (float)sum_pos/cont_pos;
                    System.out.println("Average of positives: "+ average_pos);
            }
            // Negatives
            if (cont_pos ==0)
                    System.out.println("You can't do the average of the negatives.")
;
            else {
                    average_neg= (float)sum_neg/cont_neg;
```

```
                    System.out.println("Average of negatives: "+ average_neg);
            }
    }
}
```

9. Output:

(Es un ejemplo de Output, puede diferir ya que se generan valores aleatorios)

```
Random number: 5
Random number: 2
Random number: 3
Random number: 1
Random number: 6
Random number: 6
Random number: 1
Random number: 6
Random number: 7
Random number: 6
The number of zeros entered is: 0
Average of  positives:  4.3
Average of negatives: NaN
```

2.15. Tutorial 15 - Ejercicio global - Sentencias 2

En este tutorial, vamos a ver cómo usar operadores de asignación, aritméticos, unario y generar datos random.

Una empresa que se dedica a la venta de desinfectantes necesita un programa para gestionar las facturas. En cada factura figura: el código del artículo, la cantidad vendida en litros y el precio por litro. Se pide de 5 facturas introducidas: Facturación total, cantidad en litros vendidos del artículo 1 y cuantas facturas se emitieron de más de 600 €.

1. Se crea un paquete llamado capitulo4.sentences

```
package capitulo4.sentences;
```

2. Dentro se crea una clase Sentences2

```
package capitulo4.sentences;

public class Sentences2{}
```

3. Se crea un método main

```
package capitulo4.sentences;

public class Sentences2 {

        public static void main(String[] args) {}

}
```

4. Se declaran y se inicializan las variables necesarias para almacenar los contadores de ceros, negativos y positivos

```
package capitulo4.sentences;
public class Sentences2 {
   public static void main(String[] args) {
            int code; // the item code on each invoice
            int liters; // the number of liters in each invoice
            float price; // now the price is not asked per keyboard
            float invoice_amount; // This saves the amount of the current invoice.
            float total_invoicing; // the amount of all invoices
            int liters_cod1; // the total litres sold of product 1 on all invoices
            int over_600; // counter to keep track of all invoices over 600 €
            total_invoicing = 0;
            liters_cod1 = 0; over_600 = 0;
            total_invoicing = 0;
            code = 0; over_600 = 0;
                 }
        }
}
```

5. Se realiza una sentencia for de 5 iteraciones

```
package capitulo4.sentences;
public class Sentences2 {
   public static void main(String[] args) {
            int code; // the item code on each invoice
            int liters; // the number of liters in each invoice
            float price; // now the price is not asked per keyboard
            float invoice_amount; // This saves the amount of the current invoice.
            float total_invoicing; // the amount of all invoices
            int liters_cod1; // the total litres sold of product 1 on all invoices
            int over_600; // counter to keep track of all invoices over 600 €
            total_invoicing = 0;
            liters_cod1 = 0; over_600 = 0;
            total_invoicing = 0;
```

```
            code = 0; over_600 = 0;
            for (int i=1;i<=5;i++) { }
        }
}
```

6. Se generan valores aleatorios para el código, los litros y el precio por litro

```
package capitulo4.sentences;
public class Sentences2 {
   public static void main(String[] args) {
            int code; // the item code on each invoice
            int liters; // the number of liters in each invoice
            float price; // now the price is not asked per keyboard
            float invoice_amount; // This saves the amount of the current invoice.
            float total_invoicing; // the amount of all invoices
            int liters_cod1; // the total litres sold of product 1 on all invoices
            int over_600; // counter to keep track of all invoices over 600 €
            total_invoicing = 0;
            liters_cod1 = 0; over_600 = 0;
            total_invoicing = 0;
            code = 0; over_600 = 0;
            for (int i=1;i<=5;i++) {
                    System.out.println("Invoice nº " + i);
                    code=(int) (Math.random() * 2) + 1;
                    System.out.println("product code: "+code );

                    liters=(int) (Math.random() * 10) + 1;
                    System.out.println("number (liters): "+liters);

                    price= (int) (Math.random() * 300) + 200;
                    System.out.println("price (litre): "+price);
            }
        }
}
```

7. Se calcula el importe de la factura, el total de facturación y la cantidad de litros para el artículo 1

```
package capitulo4.sentences;
public class Sentences2 {
   public static void main(String[] args) {
            int code; // the item code on each invoice
            int liters; // the number of liters in each invoice
            float price; // now the price is not asked per keyboard
            float invoice_amount; // This saves the amount of the current invoice.
            float total_invoicing; // the amount of all invoices
            int liters_cod1; // the total litres sold of product 1 on all invoices
            int over_600; // counter to keep track of all invoices over 600 €
            total_invoicing = 0;
            liters_cod1 = 0; over_600 = 0;
            total_invoicing = 0;
            code = 0; over_600 = 0;
            for (int i=1;i<=5;i++) {
                    System.out.println("Invoice nº " + i);
                    code=(int) (Math.random() * 2) + 1;
                    System.out.println("product code: "+code );

                    liters=(int) (Math.random() * 10) + 1;
                    System.out.println("number (liters): "+liters);

                    price= (int) (Math.random() * 300) + 200;
                    System.out.println("price (litre): "+price);
            }

                        invoice_amount = liters*price;
                    total_invoicing += invoice_amount;
                    if (code == 1)
                            liters_cod1 += liters;
                    if(invoice_amount >= 600)
                            over_600 ++;
```

```
            }


        }
}
```

8. Se muestran los resultados por pantalla

```
package capitulo4.sentences;
public class Sentences2 {
   public static void main(String[] args) {
            int code; // the item code on each invoice
            int liters; // the number of liters in each invoice
            float price; // now the price is not asked per keyboard
            float invoice_amount; // This saves the amount of the current invoice.
            float total_invoicing; // the amount of all invoices
            int liters_cod1; // the total litres sold of product 1 on all invoices
            int over_600; // counter to keep track of all invoices over 600 €
            total_invoicing = 0;
            liters_cod1 = 0; over_600 = 0;
            total_invoicing = 0;
            code = 0; over_600 = 0;
            for (int i=1;i<=5;i++) {
                    System.out.println("Invoice nº " + i);
                    code=(int) (Math.random() * 2) + 1;
                    System.out.println("product code: "+code );


                    liters=(int) (Math.random() * 10) + 1;
                    System.out.println("number (liters): "+liters);


                    price= (int) (Math.random() * 300) + 200;
                    System.out.println("price (litre): "+price);
            }


                        invoice_amount = liters*price;
                    total_invoicing += invoice_amount;
```

```
                    if (code == 1)
                            liters_cod1 += liters;
                    if(invoice_amount >= 600)
                            over_600 ++;
            }
                 System.out.println("\n\n\nSales Overview\n");
            // total invoicing
            System.out.println("Total invoicing: " +total_invoicing + "€");
            // liters of article 1
            System.out.println("Product sales 1: " + liters_cod1 + " liters");
            // Invoice of more than 600 euros
            System.out.println("Invoice over 600€: " + over_600);
        }
}
```

9. Output

(Es un ejemplo de Output, puede diferir ya que se generan valores aleatorios)

```
Invoice nº 1
product code: 2
number (liters): 9
price (litre): 480.0
Invoice nº 2
product code: 2
number (liters): 6
price (litre): 228.0
Invoice nº 3
product code: 1
number (liters): 7
price (litre): 214.0
Invoice nº 4
product code: 1
number (liters): 1
price (litre): 396.0
Invoice nº 5
product code: 2
number (liters): 2
```

```
price (litre): 298.0
Sales Overview
Total invoicing: 8178.0€
Product sales 1: 8 liters
Invoice over 600€: 3
```

2.16.Tutorial 16 - Ejercicio global - Sentencias 3

En este tutorial, vamos a ver cómo usar operadores de asignación, aritméticos, unario, la sentencia case y generar datos aleatorios.

Igual que el anterior pero suponiendo que no se introduce el precio por litro. Solo existen tres productos con precios:

1- 0,6 €/litro, 2- 30 €/litro y 3- 1,25 €/litro.

1. Se crea un paquete llamado capitulo4.sentencies

```
package capitulo4.sentences;
```

2. Dentro se crea una clase Sentences3

```
package capitulo4.sentencies;

public class Sentences3{}
```

3. Se crea un método main

```
package capitulo4.sentencies;

public class Sentences3 {

        public static void main(String[] args) {}

}
```

4. Se declaran y se inicializan las variables necesarias para almacenar los contadores de ceros, negativos y positivos

```
package capitulo4.sentencies;
public class Sentences3 {
   public static void main(String[] args) {
            int code; // the item code on each invoice
            int liters; // the number of liters in each invoice
            float price; // now the price is not asked per keyboard
            float invoice_amount; // This saves the amount of the current invoice.
            float total_invoicing; // the amount of all invoices
```

```
            int liters_cod1; // the total litres sold of product 1 on all invoices
            int over_600; // counter to keep track of all invoices over 600 €
            total_invoicing = 0;
            liters_cod1 = 0; over_600 = 0;
            total_invoicing = 0;
                 }
        }
}
```

5. Se realiza una sentencia for de 5 iteraciones

```
package capitulo4.sentencies;
public class Sentences3 {
   public static void main(String[] args) {
            int code; // the item code on each invoice
            int liters; // the number of liters in each invoice
            float price; // now the price is not asked per keyboard
            float invoice_amount; // This saves the amount of the current invoice.
            float total_invoicing; // the amount of all invoices
            int liters_cod1; // the total litres sold of product 1 on all invoices
            int over_600; // counter to keep track of all invoices over 600 €
            total_invoicing = 0;
            liters_cod1 = 0; over_600 = 0;
            total_invoicing = 0;
            for (int i=1;i<=5;i++) { }
        }
}
```

6. Se generan valores aleatorios para el código, los litros y el case con los precios por litro según el código

```
package capitulo4.sentencies;
public class Sentences3 {
   public static void main(String[] args) {
            int code; // the item code on each invoice
            int liters; // the number of liters in each invoice
            float price; // now the price is not asked per keyboard
            float invoice_amount; // This saves the amount of the current invoice.
            float total_invoicing; // the amount of all invoices
            int liters_cod1; // the total litres sold of product 1 on all invoices
            int over_600; // counter to keep track of all invoices over 600 €
            total_invoicing = 0;
            liters_cod1 = 0; over_600 = 0;
```

```
            total_invoicing = 0;
            for (int i=1;i<=5;i++) {
                    System.out.println("Invoice nº " + i);
                    code=(int) (Math.random() * 2) + 1;
                    System.out.println("product code: "+code );

                    liters=(int) (Math.random() * 100) + 1;
                    System.out.println("number (liters): "+liters);

                    switch (code) {
                            case 1:
                                    price = 0.6f;
                                    break;
                            case 2:
                                    price = 30f;
                                    break;
                            case 3:
                                    price = 101.25f;
                                    break;
                            default:
                                    price = 0; // this case should not occur
                    }
            }
        }
}
```

7. Se calcula el importe de la factura, el total de facturación y la cantidad de litros para el artículo 1

```
package capitulo4.sentencies;
public class Sentences3 {
   public static void main(String[] args) {
            int code; // the item code on each invoice
```

```
int liters; // the number of liters in each invoice
float price; // now the price is not asked per keyboard
float invoice_amount; // This saves the amount of the current invoice.
float total_invoicing; // the amount of all invoices
int liters_cod1; // the total litres sold of product 1 on all invoices
int over_600; // counter to keep track of all invoices over 600 €
total_invoicing = 0;
liters_cod1 = 0; over_600 = 0;
total_invoicing  =  0;
for (int i=1;i<=5;i++) {
        System.out.println("Invoice nº " + i);
        code=(int) (Math.random() * 2) + 1;
        System.out.println("product code: "+code );

        liters=(int) (Math.random() * 10) + 1;
        System.out.println("number (liters): "+liters);
        switch (code) {
                case 1:
                        price = 0.6f;
                        break;
                case 2:
                        price = 30f;
                        break;
                case 3:
                        price = 101.25f;
                        break;
                default:
                        price = 0; // this case should not occur
        }
```

```
                        invoice_amount = liters*price;
                    total_invoicing += invoice_amount;
                    if (code == 1)
                            liters_cod1 += liters;
                    if(invoice_amount >= 600)
                            over_600 ++;



        }
}
```

8. Se muestran los resultados por pantalla

```
package capitulo4.sentencies;
public class Sentencies2 {
   public static void main(String[] args) {
            int code; // the item code on each invoice
            int liters; // the number of liters in each invoice
            float price; // now the price is not asked per keyboard
            float invoice_amount; // This saves the amount of the current invoice.
            float total_invoicing; // the amount of all invoices
            int liters_cod1; // the total litres sold of product 1 on all invoices
            int over_600; // counter to keep track of all invoices over 600 €
            total_invoicing = 0;
            liters_cod1 = 0; over_600 = 0;
            total_invoicing = 0;
            code = 0; over_600 = 0;
            for (int i=1;i<=5;i++) {
                    System.out.println("Invoice nº " + i);
                    code=(int) (Math.random() * 2) + 1;
                    System.out.println("product code: "+code );


                    liters=(int) (Math.random() * 10) + 1;
                    System.out.println("number (liters): "+liters);
                    switch (code) {
```

```
                        case 1:
                                price = 0.6f;
                                break;
                        case 2:
                                price = 30f;
                                break;
                        case 3:
                                price = 101.25f;
                                break;
                        default:
                                price = 0; // this case should not occur
                }



        invoice_amount = liters*price;
total_invoicing += invoice_amount;
if (code == 1)
        liters_cod1 += liters;
if(invoice_amount >= 600)
        over_600 ++;
}
     System.out.println("\n\n\nSales Overview\n");
// total invoicing
System.out.println("Total invoicing: " +total_invoicing + "€");
// liters of article 1
System.out.println("Liters Article 1: " + liters_cod1 + " liters");
// Invoice of more than 600 euros
```

```
            System.out.println("Invoice over 600€: " + over_600);
        }
}
```

9. Output

```
Invoice nº 1
Product code: 2
Number (liters): 51
Invoice nº 2
Product code: 1
Number (liters): 87
Invoice nº 3
Product code: 1
Number (liters): 68
 Invoice nº 4
Product code: 2
Number (liters): 97
Invoice nº 5
 Product code: 2
Number (liters): 70
Sales Overview
The total invoicing is: 6633.0€
Liters Article 1 : 155 liters
Invoice over €600: 3
```

3. Tutorial Capitulo 5: API de Java

3.1. Tutorial 1 - Creando y Manipulando Strings - Strings

Se pretende en este tutorial hacer uso de las clases Java String y StringBuilder, teniendo un primer contacto con algunos de sus métodos predefinidos más útiles.

1. Se crea un paquete llamado capitulo5.string

```
package capitulo5.string;
```

2. Dentro se crea una clase StringType

```
package capitulo5.string;

public class StringType {
```

```
}
```

3. Se crea un método main

```
package capitulo5.string;

public class StringType {

  public static void main(final String args[]) {

  }

}
```

4. Se crea un array de tipo char que representa una palabra

```
package capitulo5.string;

public class StringType {

  public static void main(final String args[]) {

    char[] helloArray = { 'H', 'e', 'l', 'l', 'o'};

  }

}
```

5. Se convierte el array en un objeto String y se imprime por pantalla

```
package capitulo5.string;

public class StringType {

  public static void main(final String args[]) {

    char[] helloArray = { 'H', 'e', 'l', 'l', 'o'};

    String helloString = new String(helloArray);

    System.out.println(helloString);

  }

}
```

6. Se crea una variable len y se le asigna la longitud del String. Después se imprime por pantalla

```
package capitulo5.string;

public class StringType {

  public static void main(final String args[]) {
```

```
    char[] helloArray = { 'H', 'e', 'l', 'l', 'o'};
    String helloString = new String(helloArray);
    System.out.println(helloString);

    int len = helloString.length();
    System.out.println("String Length is : " + len);
  }
}
```

7. Se añade al String otro String y se imprime por pantalla

```
package capitulo5.string;
public class StringType {
  public static void main(final String args[]) {
    char[] helloArray = { 'H', 'e', 'l', 'l', 'o'};
    String helloString = new String(helloArray);
    System.out.println(helloString);

    int len = helloString.length();
    System.out.println("String Length is : " + len);

    helloString = helloString.concat(" World ");
    System.out.println(helloString);
  }
}
```

8. Se imprime por pantalla el carácter de la posición 6

```
package capitulo5.string;
public class StringType {
  public static void main(final String args[]) {
    char[] helloArray = { 'H', 'e', 'l', 'l', 'o'};
```

```
    String helloString = new String(helloArray);
    System.out.println(helloString);

    int len = helloString.length();
    System.out.println("String Length is : " + len);

    helloString = helloString.concat(" World ");
    System.out.println(helloString);

    System.out.println(helloString.charAt(6));
  }
}
```

9. Se imprime por pantalla la posición del primer carácter "l"

```
package capitulo5.string;

public class StringType {
  public static void main(final String args[]) {
    char[] helloArray = { 'H', 'e', 'l', 'l', 'o'};
    String helloString = new String(helloArray);
    System.out.println(helloString);

    int len = helloString.length();
    System.out.println("String Length is : " + len);

    helloString = helloString.concat(" World ");
    System.out.println(helloString);

    System.out.println(helloString.charAt(6));

    System.out.println(helloString.indexOf('l'));
```

```
  }
}
```

10. Se imprime por pantalla el substring del String inicial desde la posición 3 hasta el final

```
package capitulo5.string;

public class StringType {

  public static void main(final String args[]) {

    char[] helloArray = { 'H', 'e', 'l', 'l', 'o'};

    String helloString = new String(helloArray);

    System.out.println(helloString);


    int len = helloString.length();

    System.out.println("String Length is : " + len);


    helloString = helloString.concat(" World ");

    System.out.println(helloString);


    System.out.println(helloString.charAt(6));


    System.out.println(helloString.indexOf('l'));


    System.out.println(helloString.substring(3));

  }
}
```

11. Se imprime por pantalla el String en mayúsculas

```
package capitulo5.string;

public class StringType {

  public static void main(final String args[]) {

    char[] helloArray = { 'H', 'e', 'l', 'l', 'o'};
```

```
    String helloString = new String(helloArray);
    System.out.println(helloString);

    int len = helloString.length();
    System.out.println("String Length is : " + len);

    helloString = helloString.concat(" World ");
    System.out.println(helloString);

    System.out.println(helloString.charAt(6));

    System.out.println(helloString.indexOf('l'));

    System.out.println(helloString.substring(3));

    System.out.println(helloString.toUpperCase());
  }
}
```

12. Se imprime por pantalla si el String es igual al String "ABC"

```
package capitulo5.string;

public class StringType {

  public static void main(final String args[]) {
    char[] helloArray = { 'H', 'e', 'l', 'l', 'o'};
    String helloString = new String(helloArray);
    System.out.println(helloString);

    int len = helloString.length();
    System.out.println("String Length is : " + len);
```

```
    helloString = helloString.concat(" World ");
    System.out.println(helloString);

    System.out.println(helloString.charAt(6));

    System.out.println(helloString.indexOf('l'));

    System.out.println(helloString.substring(3));

    System.out.println(helloString.toUpperCase());

    System.out.println(helloString.equals("ABC"));
  }
}
```

13. Se imprime por pantalla si el String empieza por la letra "a"

```
package capitulo5.string;

public class StringType {

  public static void main(final String args[]) {
    char[] helloArray = { 'H', 'e', 'l', 'l', 'o'};
    String helloString = new String(helloArray);
    System.out.println(helloString);

    int len = helloString.length();
    System.out.println("String Length is : " + len);

    helloString = helloString.concat(" World ");
    System.out.println(helloString);

    System.out.println(helloString.charAt(6));
```

```
    System.out.println(helloString.indexOf('l'));

    System.out.println(helloString.substring(3));

    System.out.println(helloString.toUpperCase());

    System.out.println(helloString.equals("ABC"));

    System.out.println(helloString.startsWith("a"));
  }
}
```

14. Se imprime por pantalla si el String contiene la letra "o"

```
package capitulo5.string;

public class StringType {

  public static void main(final String args[]) {

    char[] helloArray = { 'H', 'e', 'l', 'l', 'o'};

    String helloString = new String(helloArray);

    System.out.println(helloString);

    int len = helloString.length();

    System.out.println("String Length is : " + len);

    helloString = helloString.concat(" World ");

    System.out.println(helloString);

    System.out.println(helloString.charAt(6));

    System.out.println(helloString.indexOf('l'));
```

```
    System.out.println(helloString.substring(3));

    System.out.println(helloString.toUpperCase());

    System.out.println(helloString.equals("ABC"));

    System.out.println(helloString.startsWith("a"));

    System.out.println(helloString.contains("o"));
  }
}
```

15. Se imprime por pantalla el cambio de la letra "e" por la letra "a"

```
package capitulo5.string;

public class StringType {

  public static void main(final String args[]) {

    char[] helloArray = { 'H', 'e', 'l', 'l', 'o'};

    String helloString = new String(helloArray);

    System.out.println(helloString);

    int len = helloString.length();

    System.out.println("String Length is : " + len);

    helloString = helloString.concat(" World ");

    System.out.println(helloString);

    System.out.println(helloString.charAt(6));

    System.out.println(helloString.indexOf('l'));
```

```
    System.out.println(helloString.substring(3));

    System.out.println(helloString.toUpperCase());

    System.out.println(helloString.equals("ABC"));

    System.out.println(helloString.startsWith("a"));

    System.out.println(helloString.contains("o"));

    System.out.println(helloString.replace('e', 'a'));
  }
}
```

16. Se imprime por pantalla el String sin espacios en blanco delante y detrás

```
package capitulo5.string;

public class StringType {

  public static void main(final String args[]) {

    char[] helloArray = { 'H', 'e', 'l', 'l', 'o'};

    String helloString = new String(helloArray);

    System.out.println(helloString);


    int len = helloString.length();

    System.out.println("String Length is : " + len);


    helloString = helloString.concat(" World ");

    System.out.println(helloString);


    System.out.println(helloString.charAt(6));
```

```
    System.out.println(helloString.indexOf('l'));

    System.out.println(helloString.substring(3));

    System.out.println(helloString.toUpperCase());

    System.out.println(helloString.equals("ABC"));

    System.out.println(helloString.startsWith("a"));

    System.out.println(helloString.contains("o"));

    System.out.println(helloString.replace('e', 'a'));

    System.out.println(helloString.trim());
  }
}
```

17. Se crea un String que contenga una palabra

```
package capitulo5.string;

public class StringType {

  public static void main(final String args[]) {

    char[] helloArray = { 'H', 'e', 'l', 'l', 'o'};

    String helloString = new String(helloArray);

    System.out.println(helloString);


    int len = helloString.length();

    System.out.println("String Length is : " + len);
```

```
    helloString = helloString.concat(" World ");
    System.out.println(helloString);

    System.out.println(helloString.charAt(6));

    System.out.println(helloString.indexOf('l'));

    System.out.println(helloString.substring(3));

    System.out.println(helloString.toUpperCase());

    System.out.println(helloString.equals("ABC"));

    System.out.println(helloString.startsWith("a"));

    System.out.println(helloString.contains("o"));

    System.out.println(helloString.replace('e', 'a'));

    System.out.println(helloString.trim());

    String str = "world";
  }
}
```

18. Se crea un objeto StringBuilder con ese String

```
package capitulo5.string;

public class StringType {

  public static void main(final String args[]) {

    char[] helloArray = { 'H', 'e', 'l', 'l', 'o'};
```

```
String helloString = new String(helloArray);
System.out.println(helloString);

int len = helloString.length();
System.out.println("String Length is : " + len);

helloString = helloString.concat(" World ");
System.out.println(helloString);

System.out.println(helloString.charAt(6));

System.out.println(helloString.indexOf('l'));

System.out.println(helloString.substring(3));

System.out.println(helloString.toUpperCase());

System.out.println(helloString.equals("ABC"));

System.out.println(helloString.startsWith("a"));

System.out.println(helloString.contains("o"));

System.out.println(helloString.replace('e', 'a'));

System.out.println(helloString.trim());

String str = "world";
StringBuilder sb = new StringBuilder(str);
```

```
  }
}
```

19. Se utiliza un método del objeto Stringbuilder para invertir el orden de las letras

```
package capitulo5.string;

public class StringType {

  public static void main(final String args[]) {

    char[] helloArray = { 'H', 'e', 'l', 'l', 'o'};

    String helloString = new String(helloArray);

    System.out.println(helloString);


    int len = helloString.length();

    System.out.println("String Length is : " + len);


    helloString = helloString.concat(" World ");

    System.out.println(helloString);


    System.out.println(helloString.charAt(6));


    System.out.println(helloString.indexOf('l'));


    System.out.println(helloString.substring(3));


    System.out.println(helloString.toUpperCase());


    System.out.println(helloString.equals("ABC"));


    System.out.println(helloString.startsWith("a"));


    System.out.println(helloString.contains("o"));
```

```
    System.out.println(helloString.replace('e', 'a'));

    System.out.println(helloString.trim());

    String str = "world";
    StringBuilder sb = new StringBuilder(str);
    sb.reverse();
  }
}
```

20. Se imprime por pantalla

```
package capitulo5.string;

public class StringType {
  public static void main(final String args[]) {
    char[] helloArray = { 'H', 'e', 'l', 'l', 'o'};
    String helloString = new String(helloArray);
    System.out.println(helloString);

    int len = helloString.length();
    System.out.println("String Length is : " + len);

    helloString = helloString.concat(" World ");
    System.out.println(helloString);

    System.out.println(helloString.charAt(6));

    System.out.println(helloString.indexOf('l'));

    System.out.println(helloString.substring(3));
```

```
    System.out.println(helloString.toUpperCase());

    System.out.println(helloString.equals("ABC"));

    System.out.println(helloString.startsWith("a"));

    System.out.println(helloString.contains("o"));

    System.out.println(helloString.replace('e', 'a'));

    System.out.println(helloString.trim());

    String str = "world";
    StringBuilder sb = new StringBuilder(str);
    sb.reverse();
    System.out.println(sb);
  }
}
```

21. Output

```
Hello
String Length is : 5
Hello World
W
2
lo World
HELLO WORLD
false
false
true
Hallo World
Hello World
dlrow
```

3.2. Tutorial 2 - Entender Arrays de Java - Arrays

Se pretende en este tutorial hacer uso de los arrays de Java, teniendo un primer contacto con la forma de trabajar con ellos y su forma de recorrerlos.

1. Se crea un paquete llamado capitulo5.arrays

```
package capitulo5.arrays;
```

2. Dentro se crea una clase ArrayType

```
package capitulo5.arrays;

public class ArrayType {

}
```

3. Se importa el paquete java.util.Arrays

```
package capitulo5.arrays;

import java.util.Arrays;

public class ArrayType {

}
```

4. Se crea un método main

```
package capitulo5.arrays;

import java.util.Arrays;

public class ArrayType {

  public static void main(String[] args) {

  }

}
```

5. Se crea un array de int con cuatro valores

```
package capitulo5.arrays;

import java.util.Arrays;

public class ArrayType {

  public static void main(String[] args) {

    int[] myList = {3, 5, 2, 1};
```

```
  }
}
```

6. Se crea una función printArray que dado un array imprima el valor de sus posiciones mediante un bucle for y pasándole el array creado anteriormente

```
package capitulo5.arrays;

import java.util.Arrays;

public class ArrayType {

  public static void main(String[] args) {

    int[] myList = {3, 5, 2, 1};


     printArray(myList);

  }


  public static void printArray(int[] myList) {

    for (int i = 0; i < myList.length; i++) {

      System.out.println(myList[i] + " ");

    }

  }
}
```

7. Se crea una función reverse que dado un array coja sus valores y los inserte de forma invertida en otro array y después lo devuelva

```
package capitulo5.arrays;

import java.util.Arrays;

public class ArrayType {

  public static void main(String[] args) {

    int[] myList = {3, 5, 2, 1};


     printArray(myList);
```

```
     reverse(myList);
  }

  public static void printArray(int[] myList) {
    for (int i = 0; i < myList.length; i++) {
      System.out.println(myList[i] + " ");
    }
  }

  public static int[] reverse(int[] list) {
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
    }
  }
}
```

8. Para ello se crea un array auxiliar con el tamaño del array actual. Se recorre con un bucle for el array y se insertan en el array auxiliar los valores del array actual

```
package capitulo5.arrays;
import java.util.Arrays;
public class ArrayType {
  public static void main(String[] args) {
    int[] myList = {3, 5, 2, 1};

     printArray(myList);

     reverse(myList);
  }

  public static void printArray(int[] myList) {
    for (int i = 0; i < myList.length; i++) {
```

```
      System.out.println(myList[i] + " ");
    }
  }

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
      result[j] = list[i];
    }
  }
}
```

9. Por último se devuelve el resultado

```
package capitulo5.arrays;

import java.util.Arrays;

public class ArrayType {
  public static void main(String[] args) {
    int[] myList = {3, 5, 2, 1};

     printArray(myList);

     reverse(myList);
  }

  public static void printArray(int[] myList) {
    for (int i = 0; i < myList.length; i++) {
       System.out.println(myList[i] + " ");
    }
  }
```

```
  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
      result[j] = list[i];
    }
    return result;
  }
}
```

10. Se ordena el array inicial y se imprimen sus valores por pantalla

```
package capitulo5.arrays;
import java.util.Arrays;
public class ArrayType {
  public static void main(String[] args) {
    int[] myList = {3, 5, 2, 1};

     printArray(myList);

     reverse(myList);

     Arrays.sort(myList);
     for (int num : myList)
       System.out.println(num + " ");
  }

  public static void printArray(int[] myList) {
    for (int i = 0; i < myList.length; i++) {
      System.out.println(myList[i] + " ");
    }
  }
```

```
  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
      result[j] = list[i];
    }
    return result;
  }
}
```

11. Finalmente se imprime por pantalla la posición donde se encuentra el número 3

```
package capitulo5.arrays;
import java.util.Arrays;
public class ArrayType {
  public static void main(String[] args) {
    int[] myList = {3, 5, 2, 1};

     printArray(myList);

     reverse(myList);

     Arrays.sort(myList);
     for (int num : myList)
       System.out.println(num + " ");

     System.out.println(Arrays.binarySearch(myList, 3));
  }

  public static void printArray(int[] myList) {
    for (int i = 0; i < myList.length; i++) {
```

```
      System.out.println(myList[i] + " ");
    }
  }

  public static int[] reverse(int[] list) {
    int[] result = new int[list.length];
    for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
      result[j] = list[i];
    }
    return result;
  }
}
```

12. Output

```
3
5
2
1
1
2
3
5
2
```

3.3. Tutorial 3 - Entender un ArrayList - ArrayList

Se pretende en este tutorial hacer uso de los ArrayList de Java, teniendo un primer contacto con la forma de trabajar con ellos y algunos de sus métodos.

1. Se crea un paquete llamado capitulo5.arraylist

```
package capitulo5.arraylist;
```

2. Dentro se crea una clase ArrayListType

```
package capitulo5.arraylist;

public class ArrayListType {
```

```
}
```

3. Se crea un método main

```
package capitulo5.arraylist;

public class ArrayListType {

  public static void main(String args[]) {

  }

}
```

4. Se importan los paquetes java.util.ArrayList y java.util.Collections

```
package capitulo5.arraylist;

import java.util.ArrayList;

import java.util.Collections;

public class ArrayListType {

  public static void main(String args[]) {

  }

}
```

5. Se crea un ArrayList de String

```
package capitulo5.arraylist;

import java.util.ArrayList;

import java.util.Collections;

public class ArrayListType {

  public static void main(String args[]) {

    ArrayList<String> obj = new ArrayList<String>();

  }

}
```

6. Se añaden 5 nombres al ArrayList

```
package capitulo5.arraylist;

import java.util.ArrayList;

import java.util.Collections;

public class ArrayListType {

  public static void main(String args[]) {

    ArrayList<String> obj = new ArrayList<String>();


    obj.add("Juan");

    obj.add("Harry");

    obj.add("Carlos");

    obj.add("David");

    obj.add("Pablo");

  }

}
```

7. Se imprime por pantalla el contenido del ArrayList

```
package capitulo5.arraylist;

import java.util.ArrayList;

import java.util.Collections;

public class ArrayListType {

  public static void main(String args[]) {

    ArrayList<String> obj = new ArrayList<String>();


    obj.add("Juan");

    obj.add("Harry");

    obj.add("Carlos");

    obj.add("David");

    obj.add("Pablo");
```

```
    System.out.println("Elements of the array:"+obj);
  }
}
```

8. Se añaden 2 nombres más al ArrayList

```
package capitulo5.arraylist;
import java.util.ArrayList;
import java.util.Collections;
public class ArrayListType {
  public static void main(String args[]) {
    ArrayList<String> obj = new ArrayList<String>();

    obj.add("Juan");
    obj.add("Harry");
    obj.add("Carlos");
    obj.add("David");
    obj.add("Pablo");

    System.out.println("Elements of the array:"+obj);

    obj.add(4, "Elena");
    obj.add(2, "Carla");
  }
}
```

9. Se borran 2 nombres del ArrayList

```
package capitulo5.arraylist;
import java.util.ArrayList;
import java.util.Collections;
public class ArrayListType {
```

```
  public static void main(String args[]) {
    ArrayList<String> obj = new ArrayList<String>();

    obj.add("Juan");
    obj.add("Harry");
    obj.add("Carlos");
    obj.add("David");
    obj.add("Pablo");

    System.out.println("Elements of the array:"+obj);

    obj.add(4, "Elena");
    obj.add(2, "Carla");

    obj.remove("David");
    obj.remove("Harry");
  }
}
```

10. Se cambia el nombre de la posición 0

```
package capitulo5.arraylist;
import java.util.ArrayList;
import java.util.Collections;
public class ArrayListType {
  public static void main(String args[]) {
    ArrayList<String> obj = new ArrayList<String>();

    obj.add("Juan");
    obj.add("Harry");
    obj.add("Carlos");
```

```
    obj.add("David");
    obj.add("Pablo");

    System.out.println("Elements of the array:"+obj);

    obj.add(4, "Elena");
    obj.add(2, "Carla");

    obj.remove("David");
    obj.remove("Harry");

    obj.set(0, "Robin");
  }
}
```

11. Se imprime por pantalla si el ArrayList está vacío

```
package capitulo5.arraylist;
import java.util.ArrayList;
import java.util.Collections;
public class ArrayListType {
  public static void main(String args[]) {
    ArrayList<String> obj = new ArrayList<String>();

    obj.add("Juan");
    obj.add("Harry");
    obj.add("Carlos");
    obj.add("David");
    obj.add("Pablo");
```

```
    System.out.println("Elements of the array:"+obj);


    obj.add(4, "Elena");
    obj.add(2, "Carla");


    obj.remove("David");
    obj.remove("Harry");


    obj.set(0, "Robin");


    System.out.println(obj.isEmpty());
  }
}
```

12. Se imprime por pantalla el tamañano del ArrayList

```
package capitulo5.arraylist;
import java.util.ArrayList;
import java.util.Collections;
public class ArrayListType {
  public static void main(String args[]) {
    ArrayList<String> obj = new ArrayList<String>();


    obj.add("Juan");
    obj.add("Harry");
    obj.add("Carlos");
    obj.add("David");
    obj.add("Pablo");


    System.out.println("Elements of the array:"+obj);
```

```
    obj.add(4, "Elena");
    obj.add(2, "Carla");

    obj.remove("David");
    obj.remove("Harry");

    obj.set(0, "Robin");

    System.out.println(obj.isEmpty());

    System.out.println(obj.size());
  }
}
```

13. Se imprime por pantalla si el ArrayList contiene el nombre "Carla"

```
package capitulo5.arraylist;
import java.util.ArrayList;
import java.util.Collections;
public class ArrayListType {
  public static void main(String args[]) {
    ArrayList<String> obj = new ArrayList<String>();

    obj.add("Juan");
    obj.add("Harry");
    obj.add("Carlos");
    obj.add("David");
    obj.add("Pablo");

    System.out.println("Elements of the array:"+obj);
```

```
    obj.add(4, "Elena");
    obj.add(2, "Carla");

    obj.remove("David");
    obj.remove("Harry");

    obj.set(0, "Robin");

    System.out.println(obj.isEmpty());

    System.out.println(obj.size());

    System.out.println(obj.contains("Carla"));
  }
}
```

14. Se imprime por pantalla el contenido del ArrayList

```
package capitulo5.arraylist;
import java.util.ArrayList;
import java.util.Collections;
public class ArrayListType {
  public static void main(String args[]) {
    ArrayList<String> obj = new ArrayList<String>();

    obj.add("Juan");
    obj.add("Harry");
    obj.add("Carlos");
    obj.add("David");
    obj.add("Pablo");
```

```
    System.out.println("Elements of the array:"+obj);


    obj.add(4, "Elena");
    obj.add(2, "Carla");


    obj.remove("David");
    obj.remove("Harry");


    obj.set(0, "Robin");


    System.out.println(obj.isEmpty());


    System.out.println(obj.size());


    System.out.println(obj.contains("Carla"));


    System.out.println("Current array list is:"+obj);
  }
}
```

15. Se ordenan los elementos del ArrayList

```
package capitulo5.arraylist;

import java.util.ArrayList;

import java.util.Collections;

public class ArrayListType {

  public static void main(String args[]) {

    ArrayList<String> obj = new ArrayList<String>();


    obj.add("Juan");

    obj.add("Harry");
```

```
    obj.add("Carlos");
    obj.add("David");
    obj.add("Pablo");

    System.out.println("Elements of the array:"+obj);

    obj.add(4, "Elena");
    obj.add(2, "Carla");

    obj.remove("David");
    obj.remove("Harry");

    obj.set(0, "Robin");

    System.out.println(obj.isEmpty());

    System.out.println(obj.size());

    System.out.println(obj.contains("Carla"));

    System.out.println("Current array list is:"+obj);

    Collections.sort(obj);
  }
}
```

16. Se vuelve a imprimir el contenido del ArrayList para ver que se han ordenado

```
package capitulo5.arraylist;
import java.util.ArrayList;
import java.util.Collections;
```

```
public class ArrayListType {
  public static void main(String args[]) {
    ArrayList<String> obj = new ArrayList<String>();

    obj.add("Juan");
    obj.add("Harry");
    obj.add("Carlos");
    obj.add("David");
    obj.add("Pablo");

    System.out.println("Elements of the array:"+obj);

    obj.add(4, "Elena");
    obj.add(2, "Carla");

    obj.remove("David");
    obj.remove("Harry");

    obj.set(0, "Robin");

    System.out.println(obj.isEmpty());

    System.out.println(obj.size());

    System.out.println(obj.contains("Carla"));

    System.out.println("Current array list is:"+obj);

    Collections.sort(obj);
```

```
    System.out.println("Current array list is:"+obj);
  }
}
```

17. Se borra completamente el contenido del ArrayList y se imprime por pantalla si el ArrayList está vacío

```
package capitulo5.arraylist;
import java.util.ArrayList;
import java.util.Collections;
public class ArrayListType {
  public static void main(String args[]) {
    ArrayList<String> obj = new ArrayList<String>();

    obj.add("Juan");
    obj.add("Harry");
    obj.add("Carlos");
    obj.add("David");
    obj.add("Pablo");

    System.out.println("Elements of the array:"+obj);

    obj.add(4, "Elena");
    obj.add(2, "Carla");

    obj.remove("David");
    obj.remove("Harry");

    obj.set(0, "Robin");
```

```
    System.out.println(obj.isEmpty());

    System.out.println(obj.size());

    System.out.println(obj.contains("Carla"));

    System.out.println("Current array list is:"+obj);

    Collections.sort(obj);

    System.out.println("Current array list is:"+obj);

    obj.clear();
    System.out.println(obj.isEmpty());
  }
}
```

18. Output

```
Elements of the array:[Juan, Harry, Carlos, David, Pablo]
false
5
true
Current array list is:[Robin, Carla, Carlos, Elena, Pablo]
Current array list is:[Carla, Carlos, Elena, Pablo, Robin]
true
```

3.4. Tutorial 4 - Trabajando con Fechas y Horas - DateTime

Se pretende en este tutorial trabajar de múltiples formas con fechas en Java, haciendo uso de las clases Date y SimpleDateFormat.

1. Se crea un paquete llamado capitulo5.datetime;

```
package capitulo5.datetime;
```

2. Dentro se crea una clase DateTimeType

```
package capitulo5.datetime;

public class DateTimeType {

}
```

3. Se importan los paquetes java.text.ParseException, java.text.SimpleDateFormat y java.util.Date

```
package  capitulo5.datetime;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class DateTimeType {

}
```

4. Se crea un método main

```
package capitulo5.datetime;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class DateTimeType {

  public static void main(String args[]) {

  }

}
```

5. Se crea un objeto Date()

```
package  capitulo5.datetime;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class DateTimeType {

  public static void main(String args[]) {
```

```
    Date date1 = new Date();
  }
}
```

6. Se muestra por pantalla el objeto date creado

```
package capitulo5.datetime;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
public class DateTimeType {
  public static void main(String args[]) {
     Date date1 = new Date();
     System.out.println(date1.toString());
  }
}
```

7. Se crea un objeto Date()

```
package  capitulo5.datetime;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
public class DateTimeType {
  public static void main(String args[]) {
    Date date1 = new Date();
    System.out.println(date1.toString());

    Date dNow = new Date();
  }
}
```

8. Se crea un objeto SimpleDateFormat con un formato de hora (E yyyy.MM.dd 'at' hh:mm:ss a zzz)

```
package  capitulo5.datetime;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class DateTimeType {

  public static void main(String args[]) {

    Date date1 = new Date();

    System.out.println(date1.toString());


    Date dNow = new Date();

    SimpleDateFormat ft1 = new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");

  }

}
```

9. Se imprime por pantalla el objeto Date creado con el formato creado anteriormente

```
package  capitulo5.datetime;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class DateTimeType {

  public static void main(String args[]) {

    Date date1 = new Date();

    System.out.println(date1.toString());


    Date dNow = new Date();

    SimpleDateFormat ft1 = new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");

    System.out.println("Current Date: " + ft1.format(dNow));

  }
```

```
}
```

10. Se crea un objeto Date()

```
package  capitulo5.datetime;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class DateTimeType {

  public static void main(String args[]) {

    Date date1 = new Date();

    System.out.println(date1.toString());


    Date dNow = new Date();

    SimpleDateFormat ft1 = new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");

    System.out.println("Current Date: " + ft1.format(dNow));


    Date date2 = new Date();

  }

}
```

11. Se crea un String que con la función format formatee el objeto Date creado

```
package  capitulo5.datetime;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class DateTimeType {

  public static void main(String args[]) {

    Date date1 = new Date();

    System.out.println(date1.toString());
```

```
    Date dNow = new Date();
    SimpleDateFormat ft1 = new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");
    System.out.println("Current Date: " + ft1.format(dNow));

    Date date2 = new Date();
    String str = String.format("%1$s %2$tB %2$td, %2$tY", "Due date:", date2 );
  }
}
```

12. Y se imprime por pantalla

```
package  capitulo5.datetime;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
public class DateTimeType {
  public static void main(String args[]) {
    Date date1 = new Date();
    System.out.println(date1.toString());

    Date dNow = new Date();
    SimpleDateFormat ft1 = new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");
    System.out.println("Current Date: " + ft1.format(dNow));

    Date date2 = new Date();
    String str = String.format("%1$s %2$tB %2$td, %2$tY", "Due date:", date2 );
    System.out.println(str);
  }
}
```

13. Se crea un objeto SimpleDateFormat con un formato (yyyy-MM-dd)

```
package  capitulo5.datetime;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class DateTimeType {

  public static void main(String args[]) {

    Date date1 = new Date();

    System.out.println(date1.toString());


    Date dNow = new Date();

    SimpleDateFormat ft1 = new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");

    System.out.println("Current Date: " + ft1.format(dNow));


    Date date2 = new Date();

    String str = String.format("%1$s %2$tB %2$td, %2$tY", "Due date:", date2 );

    System.out.println(str);


    SimpleDateFormat ft2 = new SimpleDateFormat ("yyyy-MM-dd");

  }

}
```

14. Se crea un String que represente una fecha en el formato anterior

```
package  capitulo5.datetime;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class DateTimeType {

  public static void main(String args[]) {

    Date date1 = new Date();
```

```
    System.out.println(date1.toString());

    Date dNow = new Date();
    SimpleDateFormat ft1 = new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");
    System.out.println("Current Date: " + ft1.format(dNow));

    Date date2 = new Date();
    String str = String.format("%1$s %2$tB %2$td, %2$tY", "Due date:", date2 );
    System.out.println(str);

    SimpleDateFormat ft2 = new SimpleDateFormat ("yyyy-MM-dd");
    String input = "1818-11-11";
  }
}
```

15. Se imprime el contenido del String

```
package capitulo5.datetime;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
public class DateTimeType {
  public static void main(String args[]) {
    Date date1 = new Date();
    System.out.println(date1.toString());

    Date dNow = new Date();
    SimpleDateFormat ft1 = new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");
    System.out.println("Current Date: " + ft1.format(dNow));

    Date date2 = new Date();
```

```
    String str = String.format("%1$s %2$tB %2$td, %2$tY", "Due date:", date2 );
    System.out.println(str);

    SimpleDateFormat ft2 = new SimpleDateFormat ("yyyy-MM-dd");
    String input = "1818-11-11";
    System.out.print(input + " Parses as ");
  }
}
```

16. Se declara una variable Date

```
package  capitulo5.datetime;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
public class DateTimeType {
  public static void main(String args[]) {
    Date date1 = new Date();
    System.out.println(date1.toString());

    Date dNow = new Date();
    SimpleDateFormat ft1 = new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");
    System.out.println("Current Date: " + ft1.format(dNow));

    Date date2 = new Date();
    String str = String.format("%1$s %2$tB %2$td, %2$tY", "Due date:", date2 );
    System.out.println(str);

    SimpleDateFormat ft2 = new SimpleDateFormat ("yyyy-MM-dd");
    String input = "1818-11-11";
    System.out.print(input + " Parses as ");
```

```
    Date t;
  }
}
```

17. Se hace un parsing del String con el formato creado anteriormente y se guarda en la variable Date creada

```
package  capitulo5.datetime;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
public class DateTimeType {
   public static void main(String args[]) {
     Date date1 = new Date();
     System.out.println(date1.toString());

     Date dNow = new Date();
     SimpleDateFormat ft1 = new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz")
;
     System.out.println("Current Date: " + ft1.format(dNow));

     Date date2 = new Date();
     String str = String.format("%1$s %2$tB %2$td, %2$tY", "Due date:", date2 );
     System.out.println(str);

     SimpleDateFormat ft2 = new SimpleDateFormat ("yyyy-MM-dd");
     String input = "1818-11-11";
     System.out.print(input + " Parses as ");
     Date t;
     t = ft2.parse(input);
  }
```

```
}
```

18. Se imprime el objeto Date

```
package capitulo5.datetime;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class DateTimeType {

  public static void main(String args[]) {

    Date date1 = new Date();

    System.out.println(date1.toString());


    Date dNow = new Date();

    SimpleDateFormat ft1 = new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");

    System.out.println("Current Date: " + ft1.format(dNow));


    Date date2 = new Date();

    String str = String.format("%1$s %2$tB %2$td, %2$tY", "Due date:", date2 );

    System.out.println(str);


    SimpleDateFormat ft2 = new SimpleDateFormat ("yyyy-MM-dd");

    String input = "1818-11-11";

    System.out.print(input + " Parses as ");

    Date t;


    t = ft2.parse(input);

    System.out.println(t);

  }

}
```

19. Esta operación puede lanzar una excepción, por lo que se utiliza un try/catch para controlar la excepción

```
package capitulo5.datetime;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class DateTimeType {

  public static void main(String args[]) {

    Date date1 = new Date();

    System.out.println(date1.toString());


    Date dNow = new Date();

    SimpleDateFormat ft1 = new SimpleDateFormat ("E yyyy.MM.dd 'at' hh:mm:ss a zzz");

    System.out.println("Current Date: " + ft1.format(dNow));


    Date date2 = new Date();

    String str = String.format("%1$s %2$tB %2$td, %2$tY", "Due date:", date2 );

    System.out.println(str);


    SimpleDateFormat ft2 = new SimpleDateFormat ("yyyy-MM-dd");

    String input = "1818-11-11";

    System.out.print(input + " Parses as ");

    Date t;

    try {

      t = ft2.parse(input);

      System.out.println(t);

    } catch (ParseException e) {

      System.out.println("Unparseable using " + ft2);

    }

  }
```

```
}
```

20. Output

```
Thu Apr 12 10:04:34 CEST 2018
Current Date: jue 2018.04.12 at 10:04:34 AM CEST
Due date: abril 12, 2018
1818-11-11 Parses as Wed Nov 11 00:00:00 CET 1818
```

4. Tutorial Capitulo 6: Métodos y encapsulamiento

4.1. Tutorial 1 - Diseñando métodos

En este breve tutorial se pretende mostrar cuál es la estructura de los métodos en Java, prestando especial atención a cuáles son los elementos que los componen y el orden en el que deben estar dichos elementos.

1. Se crea un paquete llamado capitulo6.designingmethods

```
package capitulo6.designingmethods;
```

2. Se crea una clase llamada DesigningMethods dentro del paquete capitulo6.designingmethods

```
package capitulo6.designingmethods;
public class DesigningMethods {
}
```

3. Se crean un método dentro de la clase que contenga: Un modificador de acceso, un tipo de retorno, un nombre y una lista de parámetros.

```
package capitulo6.designingmethods;
public class DesigningMethods {
    /**
     * Declaración del método methodName
     *
     * Modificadores de acceso: public static
     * Tipo de retorno: int
     * Nombre del método: methodName
     * Lista de parámetros: int a, int b
     *
```

```
     */
    public static int methodName(int a, int b) {
        /**
         * Cuerpo del método
         */
        return 0;
    }
}
```

4.2. Tutorial 2 - Trabajando con Varargs (15 min)

En este tutorial se muestra un método que utiliza varargs para que se pueda ver de forma práctica cómo se hace uso de una entrada de parámetros variable a un método, así como la forma de llamar a este método.

1. Se crea un paquete llamado capitulo6.varargs

```
package capitulo6.varargs;
```

2. Se crea una clase llamada VarargsDemo dentro del paquete capitulo6.varargs

```
package capitulo6.varargs;
public class VarargsDemo {
}
```

3. Se crea un método dentro de la clase que contenga un parámetro de entrada de talla variable (En este caso, se crea un método printMax que devuelve el mayor de los números recibidos como parámetros)

```
package capitulo6.varargs;
public class VarargsDemo {
    /**
     * Método que devuelve el mayor de los números recibidos como parámetro
     */
    public static void printMax(double... numbers) {
        /**
         * En caso de que no haya parámetros, se imprimirá por pantalla "No argument
passed"
         */
        if (numbers.length == 0) {
            System.out.println("No argument passed");
            return;
        }
        double result = numbers[0];
```

```
        for (int i = 1; i < numbers.length; i++) {
            if (numbers[i] > result) {
                result = numbers[i];
            }
        }
        System.out.println("The max value is " + result);
    }
}
```

4. Se crea un método main para poder probar el método printMax

```
package capitulo6.varargs;
public class VarargsDemo {
    /**
     * Método que devuelve el mayor de los números recibidos como parámetro
     */
    public static void printMax(double... numbers) {
        /**
         * En caso de que no haya parámetros, se imprimirá por pantalla "No argument
passed"
         */
        if (numbers.length == 0) {
            System.out.println("No argument passed");
            return;
        }
        double result = numbers[0];
        for (int i = 1; i < numbers.length; i++) {
            if (numbers[i] > result) {
                result = numbers[i];
            }
        }
        System.out.println("The max value is " + result);
    }


    public static void main(String args[]) {
        /**
         * Se llama al método dos veces con diferentes parámetros
         */
        VarargsDemo.printMax(34, 3, 3, 2, 56.5);
        VarargsDemo.printMax(new double[] { 1, 2, 3 });
    }
}
```

5. Output

```
The max value is 56.5
The max value is 3.0
```

4.3. Tutorial 3 - Aplicando Modificadores de Acceso

Este tutorial muestra una serie de ejemplos referentes a los ámbitos desde los cuales se pueden acceder a los métodos y atributos de una clase dependiendo de sus modificadores de acceso.

1. Se crea un paquete llamado capitulo6.accessmodifiers

```
package capitulo6.accessmodifiers;
```

2. Dentro de capitulo6.accessmodifiers se crea el package1

```
package capitulo6.accessmodifiers.package1;
```

3. En el paquete capitulo6.accessmodifiers.package1 se crea la clase Default

```
package capitulo6.accessmodifiers.package1;
public class Default {
}
```

4. Se crea un método con el modificiador por defecto que suma dos números a y b pasados como parámetros

```
package capitulo6.accessmodifiers.package1;
public class Default {

    /**
     * Al no poner ningún modificador de manera explícita, se aplica el modificador p
or defecto
     */
    int addTwoNumbers(final int a, final int b) {
        return a + b;
    }
}
```

Este método solo podrá ser accesible por clases dentro de su mismo paquete, si no, se obtendrá un error de compilación.

5. En el paquete capitulo6.accessmodifiers se crea la clase Test

```
package capitulo6.accessmodifiers;
public class Test{
}
```

6. Se crea el método main

```
package capitulo6.accessmodifiers;
public class Test{
  public static void main(String args[]) {
  }
}
```

7. Se crea un objeto de la clase Default (para ello habrá que importarla)

```
package capitulo6.accessmodifiers;
import capitulo6.accessmodifiers.package1.*;
public class Test{
  public static void main(String args[]) {
        Default objDefault = new Default();
  }
}
```

8. Si se intenta llamar al método .addTwoNumbers del objeto de la clase Default se podrá ver un error de compilación, ya que este método es únicamente accesible desde clases que estén en el mismo paquete que Default, y TestDefault está en un paquete diferente. Esto se debe a que su modificador de acceso es default

```
package capitulo6.accessmodifiers;
import capitulo6.accessmodifiers.package1.*;
public class Test {
    public static void main(String args[]) {
        Default objDefault = new Default();
        objDefault.addTwoNumbers(10, 21); //Error de compilación
    }
}
```

9. En el paquete capitulo6.accessmodifiers se crea la clase Private

```
package capitulo6.accessmodifiers;
class Private {
}
```

10. Se crea una propiedad double constante con el modificador private cuyo valor sea 100

```
package capitulo6.accessmodifiers;
class Private {
    private final double num = 100;
}
```

11. Se crea un método con el modificiador private que calcule el cuadrado de un número

```
package capitulo6.accessmodifiers;
class Private {
    private final double num = 100;
    private int square(int a) {
        return a * a;
    }
}
```

Ni el método ni la propiedad podrán ser accesibles por ninguna otra clase, esté en el mismo paquete que la clase Private o no.

Volviendo a la clase Test creada previamente.

12. Se crea un objeto de la clase Private

```
package capitulo6.accessmodifiers;
import capitulo6.accessmodifiers.package1.*;
public class Test{
  public static void main(String args[]) {
        Default objDefault = new Default();
       objDefault.addTwoNumbers(10, 21); //Error de compilación


       Private objPrivate = new Private();
  }
}
```

13. Si se intenta leer el valor de la propiedad num de la clase Private o si se intenta llamar al método .square se podrá ver un error de compilación, ya que el modificador private impide que tanto la propiedad como el método sean utilizados fuera de la clase Private.

```
package capitulo6.accessmodifiers;
import capitulo6.accessmodifiers.package1.*;
public class Test{
  public static void main(String args[]) {
        Default objDefault = new Default();
       objDefault.addTwoNumbers(10, 21); //Error de compilación
```

```
        Private objPrivate = new Private();
        double num = objPrivate.num; //Error de compilación
        objPrivate.square(10); //Error de compilación
  }
}
```

14. En el paquete capitulo6.accessmodifiers.package1 se crea la clase Protected

```
package capitulo6.accessmodifiers.package1;
public class Protected {
}
```

15. Se crea un método con el modificiador protected que reste dos números a y b pasados como parámetros

```
package capitulo6.accessmodifiers.package1;
public class Protected {
    protected int subctractTwoNumbers(int a, int b) {
        return a - b;
    }
}
```

Este método solo podrá ser accesible por clases que extiendan a la clase Protected.

Volviendo a la clase Test.

16. Se extiende la clase Test para que extienda a la clase Protected

```
package capitulo6.accessmodifiers;
import capitulo6.accessmodifiers.package1.*;
public class Test extends Protected{
  public static void main(String args[]) {
        Default objDefault = new Default();
        objDefault.addTwoNumbers(10, 21); //Error de compilación

        Private objPrivate = new Private();
        double num = objPrivate.num; //Error de compilación
        objPrivate.square(10); //Error de compilación
  }
}
```

17. Se crea un objeto de la clase Test

```
package capitulo6.accessmodifiers;
import capitulo6.accessmodifiers.package1.*;
public class Test extends Protected{
  public static void main(String args[]) {
       Default objDefault = new Default();
       objDefault.addTwoNumbers(10, 21); //Error de compilación


       Private objPrivate = new Private();
       double num = objPrivate.num; //Error de compilación
       objPrivate.square(10); //Error de compilación


       Test obj = new Test();
  }
}
```

18. Puesto que Test extiende a Protected, el objeto de clase Test podrá llamar al método con el modificador de acceso protected .subctractTwoNumbers de la clase Protected

```
package capitulo6.accessmodifiers;
import capitulo6.accessmodifiers.package1.*;
public class Test extends Protected{
  public static void main(String args[]) {
       Default objDefault = new Default();
       objDefault.addTwoNumbers(10, 21); //Error de compilación


       Private objPrivate = new Private();
       double num = objPrivate.num; //Error de compilación
       objPrivate.square(10); //Error de compilación


       Test obj = new Test();
        /**
         * Se imprime por pantalla el restultado del método .subctractTwoNumbers
         * de la clase Protected.java
         */
        System.out.println(obj.subctractTwoNumbers(20, 10));
  }
}
```

19. En el paquete capitulo6.accessmodifiers.package1 se crea la clase Public

```
package capitulo6.accessmodifiers.package1;
public class Public {
}
```

20. Se crea un método con el modificiador public que divida dos números a y b pasados como parámetros

```
package capitulo6.accessmodifiers.package1;
public class Public {
    public int divideTwoNumbers(int a, int b){
     return a/b;
    }
}
```

Este método podrá ser accesible por cualquier clase, se encuentre en el mismo paquete que la clase Public o en un paquete diferente.

Volviendo a la clase Test.

21. Se crea un objeto de la clase Public

```
package capitulo6.accessmodifiers;
import capitulo6.accessmodifiers.package1.*;
public class Test extends Protected{
  public static void main(String args[]) {
       Default objDefault = new Default();
       objDefault.addTwoNumbers(10, 21); //Error de compilación

       Private objPrivate = new Private();
       double num = objPrivate.num; //Error de compilación
       objPrivate.square(10); //Error de compilación

       Test obj = new Test();
        /**
         * Se imprime por pantalla el restultado del método .subctractTwoNumbers
         * de la clase Protected.java
         */
        System.out.println(obj.subctractTwoNumbers(20, 10));

        Public objPublic = new Public();
  }
}
```

22. Puesto que el método .divideTwoNumbers de la clase Public tiene el modificador de acceso public, se podrá llamar sin problema desde la clase Test

```
package capitulo6.accessmodifiers;
import capitulo6.accessmodifiers.package1.*;
public class Test extends Protected{
  public static void main(String args[]) {
       Default objDefault = new Default();
       objDefault.addTwoNumbers(10, 21); //Error de compilación


       Private objPrivate = new Private();
       double num = objPrivate.num; //Error de compilación
       objPrivate.square(10); //Error de compilación


       Test obj = new Test();
        /**
         * Se imprime por pantalla el restultado del método .subctractTwoNumbers
         * de la clase Protected.java
         */
        System.out.println(obj.subctractTwoNumbers(20, 10));


        Public objPublic = new Public();
        /**
         * Se imprime por pantalla el restultado del método .divideTwoNumbers
         * de la clase Public.java
         */
        System.out.println(objPublic.divideTwoNumbers(20, 10));
  }
}
```

Para poder ejecutar el programa, será necesario comentar las líneas de código que no compilen.

23. Output

```
10
2
```

4.4. Tutorial 4 - Pasando Datos a Través de Métodos

Este tutorial pretende exponer diferentes ejemplos sobre la forma de pasar datos (variables) a métodos y de qué forma estos datos permanecen modificados o no tras finalizar la ejecución de dichos métodos.

1. Se crea un paquete llamado capitulo6.passingdataamongmethods

```
package capitulo6.passingdataamongmethods;
```

2. En el paquete capitulo6.passingdataamongmethods se crea la clase PassingDataAmongMethods

```
package capitulo6.passingdataamongmethods;
public class PassingDataAmongMethods {
}
```

3. Se crea el método main

```
package capitulo6.passingdataamongmethods;
public class PassingDataAmongMethods {
  public static void main(final String[] args) {
  }
}
```

4. Dentro del método main se han implementado una serie de ejemplos que permiten ver cuál es el verdadero valor de ciertas variables a lo largo del código

```
package capitulo6.passingdataamongmethods;
public class PassingDataAmongMethods {
  public static void main(final String[] args) {
        /**
         * Se crea una variable de tipo String con el valor "Hello!"
         */
        String s1 = "Hello!";

        /**
         * Se llama al método bye y se le pasa s1 como parámetro
         */
        bye(s1);

        /**
         * Se imprime en la consola el valor de s1, que será
         * Hello!
         * Esto se debe a que a pesar de que en el método bye
         * hay una variable que se llama s1, es una variable
         * de ámbito local completamente diferente a la variable
         * s1 creada en el método main.
```

```
         */
        System.out.println(s1);

        /**
         * Se crea una variable de tipo StringBuilder
         */
        StringBuilder s2 = new StringBuilder();

        /**
         * Se llama al método goodMorning y se le pasa s2 como parámetro
         */
        goodMorning(s2);

        /**
         * Se imprime en la consola el valor de s1, que será
         * Good Morning
         * Esto se debe a que s sí que es el mismo objeto que
         * se le pasa como parámetro.
         */
        System.out.println(s2);
    }
    public static void bye(String s1) {
        s1 = "Bye!";
    }

    public static void goodMorning(StringBuilder s) {
        s.append("Good Morning");
  }
}
```

5. Output

```
Hello!
Good Morning
```

4.5. Tutorial 5 - Sobrecargando Métodos

En el siguiente tutorial se muestra la forma en la que es posible sobrecargar métodos en Java, creando métodos diferentes pero con el mismo nombre.

1. Se crea un paquete llamado capitulo6.overloadingmethods

```
package capitulo6.overloadingmethods;
```

2. Se crea una clase llamada OverloadingMethods dentro del paqute capitulo6.overloadingmethods

```
package capitulo6.overloadingmethods;
public class OverloadingMethods {
}
```

3. Se crean una serie de métodos llamados disp, con diferentes parámetros (overlad al método disp)

```
package capitulo6.overloadingmethods;
public class OverloadingMethods {
    public void disp(char c) {
        System.out.println(c);
    }

    public void disp(char c, int num) {
        System.out.println(c + " " + num);
    }

    public void disp(int c) {
        System.out.println(c);
    }

    public void disp(int num, char c) {
        System.out.println(num + " " + c);
    }
}
```

4. Se crea el método main para poder hacer llamadas a los métodos y ver sus resultados

```
package capitulo6.overloadingmethods;
public class OverloadingMethods {
    public void disp(char c) {
        System.out.println(c);
    }
```

```
    public void disp(char c, int num) {
        System.out.println(c + " " + num);
    }
    public void disp(int c) {
        System.out.println(c);
    }


    public void disp(int num, char c) {
        System.out.println(num + " " + c);
    }


    public static void main(String args[]) {
        OverloadingMethods obj = new OverloadingMethods();
        obj.disp('a');
        obj.disp('a', 10);
        obj.disp(2);
        obj.disp(10, 'a');
    }
}
```

5. Output

```
a
a 10
2
10 a
```

4.6. Tutorial 6 - Creando Constructores

Este tutorial pretende mostrar cuál es la forma de crear diferentes constructores para una misma clase Java, y cómo se puede crear un objeto de una clase haciendo uso de uno u otro de sus constructores.

1. Se crea un paquete llamado capitulo6.constructors

```
package capitulo6.constructors;
```

2. En el paquete capitulo6.constructors se crea la clase Constructors

```
package capitulo6.constructors;
public class Constructors {
}
```

3. Se crea una propiedad que sea un número entero llamada var

```
package capitulo6.constructors;
public class Constructors {
    private int var;
}
```

4. Se crea un constructor vacío que modifique var a 10

```
package capitulo6.constructors;
public class Constructors {
    private int var;


    /**
     * Constructor vacío
     */
    public Constructors(){
           var = 10;
    }
}
```

5. Se crea un constructor con un parámetro que da a var el valor que se le pase como parámetro

```
package capitulo6.constructors;
public class Constructors {
    private int var;

    /**
     * Constructor vacío
     */
    public Constructors(){
           var = 10;
    }

    /**
     * Constructor con parámetros
     */
    public Constructors(int num){
```

```
            var = num;
    }
}
```

6. Para poder hacer pruebas, se crea también el método getValue(), que devolverá el valor de la propiedad var

```
package capitulo6.constructors;
public class Constructors {
    private int var;


    /**
     * Constructor vacío
     */
    public Constructors(){
            var = 10;
    }


    /**
     * Constructor con parámetros
     */
    public Constructors(int num){
            var = num;
    }


    public int getValue(){
             return var;
    }
}
```

7. Se crea el método main

```
package capitulo6.constructors;
public class Constructors {
    private int var;


    /**
     * Constructor vacío
     */
    public Constructors(){
            var = 10;
    }
```

```
    /**
     * Constructor con parámetros
     */
    public Constructors(int num){
           var = num;
    }

    public int getValue(){
            return var;
    }

    public static void main(String args[]){
    }
}
```

8. Dentro del método main, se crean dos objetos Constructors, uno con el constructor vacío y otro con el constructor con parámetros, y se comprueba el valor de sus propiedades var

```
package capitulo6.constructors;
public class Constructors {
    private int var;

    /**
     * Constructor vacío
     */
    public Constructors(){
           var = 10;
    }

    /**
     * Constructor con parámetros
     */
    public Constructors(int num){
           var = num;
    }

    public int getValue(){
            return var;
    }
```

```
    public static void main(String args[]){
        Constructors obj = new Constructors();
        Constructors obj2 = new Constructors(100);
        System.out.println("var is: "+obj.getValue());
        System.out.println("var is: "+obj2.getValue());
    }
}
```

9. Output

```
var is: 10
var is: 100
```

4.7. Tutorial 7 - Encapsulando Datos

Este tutorial pretende mostrar cómo se deben encapsular los atributos de una clase para dar acceso a la lectura y modificación de los mismos a través de métodos get y set (encapsulación).

1. Se crea un paquete llamado capitulo6.encapsulatingdata

```
package capitulo6.encapsulatingdata;
```

2. Se crea una clase llamada EncapsulatingData dentro del paquete capitulo6.encapsulatingdata

```
package capitulo6.encapsulatingdata;
public class EncapsulatingData {
}
```

3. Se crean tres propiedades privadas: ssn(int), empName(String) y empAge(int)

```
package capitulo6.encapsulatingdata;
public class EncapsulatingData {
    /**
     * Propiedades de la clase EncapsulatingData
     */
    private int ssn;
    private String empName;
    private int empAge;
}
```

4. Se crean todos los métodos get y set para las tres propiedades

```
package capitulo6.encapsulatingdata;
public class EncapsulatingData {
    /**
     * Propiedades de la clase EncapsulatingData
     */
    private int ssn;
    private String empName;
    private int empAge;


    /**
     * Métodos getterss y setterss
     */
    public int getEmpSSN() {
        return ssn;
    }
    public String getEmpName() {
        return empName;
    }
    public int getEmpAge() {
        return empAge;
    }


    public void setEmpAge(int newValue) {
        empAge = newValue;
    }
    public void setEmpName(String newValue) {
        empName = newValue;
    }
    public void setEmpSSN(int newValue) {
        ssn = newValue;
    }
}
```

Desde la clase EncapsulatingDataTest.java se podrán recuperar y modificar las propiedades accediendo a los métodos get y set de cada una de ellas

5. Se crea una clase llamada EncapsulatingDataTest dentro del paqute capitulo6.encapsulatingdata

```
package capitulo6.encapsulatingdata;
public class EncapsulatingDataTest {
}
```

6. Se crea el médoto main

```
package capitulo6.encapsulatingdata;
public class EncapsulatingDataTest {
  public static void main(final String args[]) {
  }
}
```

7. Se crea un objeto del tipo EncapsulatingData

```
package capitulo6.encapsulatingdata;
public class EncapsulatingDataTest {
  public static void main(final String args[]) {
    EncapsulatingData obj = new EncapsulatingData();
  }
}
```

8. Se modifican las propiedades del objeto con los métodos set implemetados en la clase EncapsulatingData

```
package capitulo6.encapsulatingdata;
public class EncapsulatingDataTest {
  public static void main(final String args[]) {
        EncapsulatingData obj = new EncapsulatingData();
        obj.setEmpName("Mario");
        obj.setEmpAge(32);
        obj.setEmpSSN(112233);
  }
}
```

9. Se leen las propiedades del objeto con los métodos get implementados en la clase EncapsulatingData

```
package capitulo6.encapsulatingdata;
public class EncapsulatingDataTest {
  public static void main(final String args[]) {
        EncapsulatingData obj = new EncapsulatingData();


        obj.setEmpName("Mario");
        obj.setEmpAge(32);
        obj.setEmpSSN(112233);


        System.out.println("Employee Name: " + obj.getEmpName());
        System.out.println("Employee SSN: " + obj.getEmpSSN());
```

```
        System.out.println("Employee Age: " + obj.getEmpAge());
  }
}
```

10. Output

```
Employee Name: Mario
Employee SSN: 112233
Employee Age: 32
```

4.8. Tutorial 8 - Ejercicio de ejemplo: Home

En este tutorial se pretende implementar un ejemplo que se pueda usar como guía para hacer el ejercicio del capítulo 6. En este ejercicio se implementarán tres clases con unos atributos específicos, que se inicializarán dentro de los constructores correspondientes a dichas clases. También se implementarán los getters y setters en cada clase necesarios para hacer uso de dichos atributos.

1. Se crea un paquete capitulo6.ejercicehomeexample.

```
package capitulo6.exercisehomeexample;
```

2. Se definen tres clases dentro del paquete creado: House, LivingRoom y Kitchen.

```
package capitulo6.exercisehomeexample;
public class House { }
```

```
package capitulo6.exercisehomeexample;
public class Kitchen { }
```

```
package capitulo6.exercisehomeexample;
public class LivingRoom { }
```

3. La clase LivingRoom debe tener como atributos privados numberOfTV (int) y livingRoomType (String).

```
package capitulo6.exercisehomeexample;
public class LivingRoom {
    //Atributos de la clase LivingRoom
    private int numberOfTV;
    private String livingRoomType;
}
```

4. Se crea un constructor que los inicialice a 0 y "Unknown".

```
package capitulo6.exercisehomeexample;
public class LivingRoom {
    //Atributos de la clase LivingRoom
    private int numberOfTV;
    private String livingRoomType;
    //Constructor
    public LivingRoom() {
        numberOfTV = 0;
        livingRoomType = "Unknown";
    }
}
```

5. Se crean los getters y setters de la clase LivingRoom, para quedar la clase implementada:

```
package capitulo6.exercisehomeexample;
public class LivingRoom {
    //Atributos de la clase LivingRoom
    private int numberOfTV;
    private String livingRoomType;
    //Constructor
    public LivingRoom() {
        numberOfTV = 0;
        livingRoomType = "Unknown";
    }
    //Metodos getters y setters
    public void setNumberOfTV ( int valueNumberOfTV ) {
        numberOfTV = valueNumberOfTV;
    }
    public void setLivingRoomType ( String valueLivingRoomType ) {
        livingRoomType = valueLivingRoomType;
    }
    public int getNumberOfTV () {
        return numberOfTV;
    }
    public String getLivingRoomType () {
        return livingRoomType;
    }
}
```

6. La clase Kitchen debe tener como atributos privados isIndependent (boolean) y numberOfStoves (int).

```
package capitulo6.exercisehomeexample;
public class Kitchen {
    //Atributos de la clase Kitchen
    private boolean isIndependent;
```

```
    private int numberOfStoves;
}
```

7. Crear un constructor e inicializar los atributos anteriores a false y 0.

```
package capitulo6.exercisehomeexample;
public class Kitchen {
    //Atributos de la clase Kitchen
    private boolean isIndependent;
    private int numberOfStoves;
    //Constructor
    public Kitchen() {
        isIndependent = false;
        numberOfStoves = 0;
    }
}
```

8. Se implementan los getters y setters de la clase Kitchen, para quedar la clase implementada:

```
package capitulo6.exercisehomeexample;
public class Kitchen {
    //Atributos de la clase Kitchen
    private boolean isIndependent;
    private int numberOfStoves;
    //Constructor
    public Kitchen() {
        isIndependent = false;
        numberOfStoves = 0;
    }
    //Metodos getters y setters
    public void setIsIndependent ( boolean valueIsIndependent ) {
        isIndependent = valueIsIndependent;
    }
    public void setNumberOfStoves ( int valueNumberOfStoves ) {
        numberOfStoves = valueNumberOfStoves;
    }
    public boolean getIsIndependent () {
        return isIndependent;
    }
    public int getNumberOfStoves () {
        return numberOfStoves;
    }
}
```

9. La clase House tendrá los siguientes atributos públicos de clase: area (double), address (String), livingRoom (tipo LivingRoom) y kitchen (tipo Kitchen).

```
package capitulo6.exercisehomeexample;
public class House {
    //Atributos de la clase House
    public static double area;
    public static String address;
    public static LivingRoom livingRoom;
    public static Kitchen kitchen;
}
```

10. Se deberá definir un constructor para la clase House que establezca a unos valores por defecto a sus atributos y que cree nuevos objetos si se trata de atributos objeto.

```
package capitulo6.exercisehomeexample;
public class House {
    //Atributos de la clase House
    public static double area;
    public static String address;
    public static LivingRoom livingRoom;
    public static Kitchen kitchen;
    //Constructor
    public House() {
        area = 0.0d;
        address = "Unknown";
        livingRoom = new LivingRoom();
        kitchen = new Kitchen();
    }
}
```

11. Se implementarán los métodos getters y setters necesarios para poder trabajar con todos los atributos de la clase.

```
package capitulo6.exercisehomeexample;
public class House {
    //Atributos de la clase House
    public static double area;
    public static String address;
    public static LivingRoom livingRoom;
    public static Kitchen kitchen;
    //Constructor
    public House() {
        area = 0.0d;
        address = "Unknown";
        livingRoom = new LivingRoom();
        kitchen = new Kitchen();
    }
    //Metodos getters y setters
```

```
    public void setArea ( double valueArea ) {
        area = valueArea;
    }
    public void setAddress ( String valueAddress ) {
        address = valueAddress;
    }
    public void setLivingRoom ( LivingRoom valueLivingRoom ) {
        livingRoom.setNumberOfTV(valueLivingRoom.getNumberOfTV());
        livingRoom.setLivingRoomType(valueLivingRoom.getLivingRoomType());
    }
    public void setKitchen( Kitchen valueKitchen ) {
        kitchen.setIsIndependent(valueKitchen.getIsIndependent());
        kitchen.setNumberOfStoves(valueKitchen.getNumberOfStoves());
    }
    public double getArea () {
        return area;
    }
    public String getAddress () {
        return address;
    }
    public static LivingRoom getLivingRoom () {
        return livingRoom;
    }
    public static Kitchen getKitchen () {
        return kitchen;
    }
}
```

12. Después se creara un método main en la propia clase House donde se instancie un objeto de la clase House, quedando la clase implementada:

```
package capitulo6.exercisehomeexample;
public class House {
    //Atributos de la clase House
    public static double area;
    public static String address;
    public static LivingRoom livingRoom;
    public static Kitchen kitchen;
    //Constructor
    public House() {
        area = 0.0d;
        address = "Unknown";
        livingRoom = new LivingRoom();
        kitchen = new Kitchen();
    }
```

```
    //Metodos getters y setters
    public void setArea ( double valueArea ) {
        area = valueArea;
    }
    public void setAddress ( String valueAddress ) {
        address = valueAddress;
    }
    public void setLivingRoom ( LivingRoom valueLivingRoom ) {
        livingRoom.setNumberOfTV(valueLivingRoom.getNumberOfTV());
        livingRoom.setLivingRoomType(valueLivingRoom.getLivingRoomType());
    }
    public void setKitchen( Kitchen valueKitchen ) {
        kitchen.setIsIndependent(valueKitchen.getIsIndependent());
        kitchen.setNumberOfStoves(valueKitchen.getNumberOfStoves());
    }
    public double getArea () {
        return area;
    }
    public String getAddress () {
        return address;
    }
    public static LivingRoom getLivingRoom () {
        return livingRoom;
    }
    public static Kitchen getKitchen () {
        return kitchen;
    }
public static void main(String[] args) {
      House house = new House();
      String message = "House area: " + house.area + "\n";
      message = message + "House address: " + house.address + ".\n";
      message = message + "Let's go inside the house:\n";
      message = message + "The house has a livingroom of type "
                        + house.getLivingRoom().getLivingRoomType()
                        + " and with  " + house.getLivingRoom().getNumberOfTV()
                        + " TV's \n";
      message = message + "The house has a kitchen has  "
                        + house.getKitchen().getNumberOfStoves()
                        +" stoves and the kitchen is "
                        + house.getKitchen().getIsIndependent()
```

```
                        +" independent \n";
        message = message + "\n\n\tEnd!";
        System.out.println ( message ) ;
    }
}
```

13. Output

```
House area: 0.0
House address: Unknown.
Let's go inside the house:
The house has a livingroom of type Unknown and with  0 TV's
The house has a kitchen has  0 stoves and the kitchen is false independent
   End!
```

5. Tutorial Capítulo 7: Diseño de clases

5.1. Tutorial 1 - Introducción a la herencia de clases - Herencia

Este tutorial muestra un ejemplo sencillo de herencia en Java, haciendo uso de una clase padre y una clase hija, así como de una serie de métodos que se definen en ambas clases.

1. Se crea un paquete llamado capitulo7.inheritance

```
package capitulo7.inheritance;
```

2. Dentro se crea una clase ParentClass

```
package capitulo7.inheritance;
class ParentClass{
 }
}
```

3. Se crea un constructor vacío que muestre un mensaje por pantalla y otro con dos argumentos proctected (*name* y *age*)

```
package capitulo7.inheritance;
class ParentClass{
  protected String name;
  protected int age;
  ParentClass(){
     System.out.println("Constructor of Parent");
```

```
  }
  ParentClass(String name, int age){
     this.name = name;
     this.age = age;
  }
}
```

4. Se crea un método disp() que muestre por pantalla un mensaje indicando que es el método del padre

```
package capitulo7.inheritance;
class ParentClass{
  protected String name;
  protected int age;
  ParentClass(){
     System.out.println("Constructor of Parent");
  }
  ParentClass(String name, int age){
     this.name = name;
     this.age = age;
  }
}
```

5. Se crea una nueva clase public ChildClass que herede de la clase ParentClass

```
package capitulo7.inheritance;
public class ChildClass extends ParentClass{
}
```

6. Se crea un constructor vacío que muestre por pantalla que es el constructor de la clase hijo. Se crea también un constructor con un parámetro (*age*) en la clase hijo que utiliza el segundo constructor heredado de la clase padre.

```
package capitulo7.inheritance;
public class ChildClass extends ParentClass{
  ChildClass(){
     System.out.println("Constructor of Child");
  }
  public ChildClass(int age){
     super(“Child”, age);
  }
}
```

7. Se crea un método disp() que muestre un mensaje por pantalla indicando que es el método del hijo.

```
package capitulo7.inheritance;
public class ChildClass extends ParentClass{
  ChildClass(){
     System.out.println("Constructor of Child");
  }
  public ChildClass(int age){
     super(“Child”, age);
  }
  void disp(){
     System.out.println("Child Method");
  }
}
```

8. Se realiza una llamada al método disp() heredado del padre.

```
package capitulo7.inheritance;
public class ChildClass extends ParentClass{
   ChildClass(){
       System.out.println("Constructor of Child");
   }
   public ChildClass(int age){
       super(“Child”, age);
   }
   void disp(){
       System.out.println("Child Method");
       //Calling the disp() method of parent class
       super.disp();
   }
}
```

9. Se crea también un segundo método disp2() que muestre un mensaje por pantalla indicando el nombre y la edad pasados al constructor hijo.

```
package capitulo7.inheritance;
public class ChildClass extends ParentClass{
  ChildClass(){
     System.out.println("Constructor of Child");
  }
  public ChildClass(int age){
     super(“Child”, age);
  }
  void disp(){
```

```
      System.out.println("Child Method");
      //Calling the disp() method of parent class
      super.disp();
   }
   void disp2(){
      System.out.println(super.name + ” is “ + this.age + “ years old”);
   }
}
```

10. Se crea un método main dentro de la clase ChildClass

```
package capitulo7.inheritance;
public class ChildClass extends ParentClass{
   ChildClass(){
      System.out.println("Constructor of Child");
   }
   public ChildClass(int age){
      super(“Child”, age);
   }
   void disp(){
      System.out.println("Child Method");
      //Calling the disp() method of parent class
      super.disp();
   }
   void disp2(){
      System.out.println(super.name + ” is “ + this.age + “ years old”);
   }
   public static void main(String args[]){
   }
}
```

11. Se crea un objeto ChildClass. Se crea también un segundo objeto ChildClass con parámetro *age*=5.

```
package capitulo7.inheritance;
public class ChildClass extends ParentClass{
   ChildClass(){
      System.out.println("Constructor of Child");
   }
   public ChildClass(int age){
      super(“Child”, age);
   }
   void disp(){
      System.out.println("Child Method");
```

```
    //Calling the disp() method of parent class
    super.disp();
  }
  void disp2(){
    System.out.println(super.name + ” is “ + this.age “ years old”);
  }
  public static void main(String args[]){
    ChildClass child = new ChildClass();
    ChildClass child2 = new ChildClass(5);
  }
}
```

12. Se realiza una llamada a los métodos disp() y disp2() del objeto child y child2, respectivamente.

```
package capitulo7.inheritance;
public class ChildClass extends ParentClass{
  ChildClass(){
    System.out.println("Constructor of Child");
  }
  public ChildClass(int age){
    super(“Child”, age);
  }
  void disp(){
    System.out.println("Child Method");
    //Calling the disp() method of parent class
    super.disp();
  }
  void disp2(){
    System.out.println(super.name + ” is “ + this.age + “ years old”);
  }
  public static void main(String args[]){
    ChildClass child = new ChildClass();
    ChildClass child2 = new ChildClass(5);
    child.disp();
    child2.disp2();
  }
}
```

13. Output

```
Constructor of Parent

Constructor of Child

Child Method
```

```
Parent Method
Child is 5 years old
```

5.2. Tutorial 2 - Creando clases abstractas - Clases abstractas

Este tutorial pretende tener un primer contacto con una clase abstracta que tiene tanto métodos no-abstractos como métodos abstractos implementados por las clases que extienden a la clase abstracta.

1. Se crea un paquete llamado capitulo7.abstractclass

```
package capitulo7.abstractclass;
```

2. Dentro se crea una clase abstracta AbstractDemo

```
package capitulo7.abstractclass;

abstract class AbstractDemo{

}
```

3. Se crea un metodo que imprima por pantalla un mensaje

```
package capitulo7.abstractclass;

abstract class AbstractDemo{

  public void disp(){

    System.out.println("Concrete method of parent class");

  }

}
```

4. Se crea un método abstracto vacío

```
package capitulo7.abstractclass;

abstract class AbstractDemo{

  public void disp(){

    System.out.println("Concrete method of parent class");

  }

  public abstract void disp2();

}
```

5. Se crea otro método abstracto vacío que reciba dos parametros

```
package capitulo7.abstractclass;

abstract class AbstractDemo{

  public void disp(){

    System.out.println("Concrete method of parent class");

  }

  public abstract void disp2();

  public abstract int sumOfTwo(int n1, int n2);

}
```

6. Se crea una nueva clase Demo que herede de la clase abstracta AbstractDemo

```
package capitulo7.abstractclass;

public class Demo extends AbstractDemo{

}
```

7. Se implementa el método abstracto creado en la clase AbstractDemo y se muestra por pantalla un mensaje

```
package  capitulo7.abstractclass;

public class Demo extends AbstractDemo{

  public void  disp2(){

    System.out.println("overriding abstract method");

  }

}
```

8. Se implementa el método abstracto que recibe dos parametros de la clase AbstractDemo y se suman

```
package  capitulo7.abstractclass;

public class Demo extends AbstractDemo{

  public void  disp2(){

    System.out.println("overriding abstract method");

  }
```

```
  public int sumOfTwo(int num1, int num2){
    return num1+num2;
  }
}
```

9. Se crea un método main

```
package  capitulo7.abstractclass;
public class Demo extends AbstractDemo{
  public void  disp2(){
    System.out.println("overriding abstract method");
  }
  public int sumOfTwo(int num1, int num2){
    return num1+num2;
  }
  public static void main(String args[]){
  }
}
```

10. Se crea un objeto Demo. No se puede crear un objeto de una clase abstract

```
package capitulo7.abstractclass;
public class Demo extends AbstractDemo{
  public void disp2(){
    System.out.println("overriding abstract method");
  }
  public int sumOfTwo(int num1, int num2){
    return num1+num2;
  }
  public static void main(String args[]){
    //AbstractDemo obj = new AbstractDemo();
    Demo obj = new Demo();
```

```
  }
}
```

11. Se realiza una llamada al método de la clase heredada

```
package  capitulo7.abstractclass;

public class Demo extends AbstractDemo{

  public void  disp2(){

    System.out.println("overriding abstract method");

  }

  public int sumOfTwo(int num1, int num2){

    return num1+num2;

  }

  public static void main(String args[]){

    //AbstractDemo obj = new AbstractDemo();

    Demo obj = new Demo();

    obj.disp();

  }

}
```

12. Se realiza una llamada al método de la clase Demo

```
package  capitulo7.abstractclass;

public class Demo extends AbstractDemo{

  public void  disp2(){

    System.out.println("overriding abstract method");

  }

  public int sumOfTwo(int num1, int num2){

    return num1+num2;

  }

  public static void main(String args[]){

    //AbstractDemo obj = new AbstractDemo();
```

```
    Demo obj = new Demo();
    obj.disp();
    obj.disp2();
  }
}
```

13. Se imprime por pantalla lo que devuelve el método de suma de la clase Demo

```
package  capitulo7.abstractclass;
public class Demo extends AbstractDemo{
  public void  disp2(){
    System.out.println("overriding abstract method");
  }
  public int sumOfTwo(int num1, int num2){
    return num1+num2;
  }
  public static void main(String args[]){
    //AbstractDemo obj = new AbstractDemo();
    Demo obj = new Demo();
    obj.disp();
    obj.disp2();
    System.out.println(obj.sumOfTwo(3, 7));
  }
}
```

14. Output

```
Concrete method of parent class
overriding abstract method
10
```

5.3. Tutorial 3 - Implementando interfaces - Interfaces

Este tutorial pretende hacer uso de una serie de interfaces que se implementan entre ellas para poder ver qué es exactamente una interfaz en Java y cómo se puede hacer uso de ella.

1. Se crea un paquete llamado capitulo7.interfaceclass

```
package capitulo7.interfaceclass;
```

2. Se crea una interfaz Inf1 con un método vacío

```
package capitulo7.interfaceclass;

interface Inf1{

  public void method1();

}
```

3. Se crea una interfaz Inf2 que herede de Inf1

```
package capitulo7.interfaceclass;

interface Inf1{

  public void method1();

}

interface Inf2 extends Inf1 {

}
```

4. Dentro se crea un método default que muestre por pantalla un mensaje

```
package capitulo7.interfaceclass;

interface Inf1{

  public void method1();

}

interface Inf2 extends Inf1 {

  default void  newMethod(){

    System.out.println("Newly added default method");

  }

}
```

5. También se crea un método estático que muestre por pantalla otro mensaje

```
package capitulo7.interfaceclass;

interface Inf1{

  public void method1();

}

interface Inf2 extends Inf1 {

  default void newMethod(){

    System.out.println("Newly added default method");

  }

  static void anotherNewMethod(){

    System.out.println("Newly added static method");

  }

}
```

6. Por último se crea un método vacío

```
package capitulo7.interfaceclass;

interface Inf1{

  public void method1();

}

interface Inf2 extends Inf1 {

  default void newMethod(){

    System.out.println("Newly added default method");

  }

  static void anotherNewMethod(){

    System.out.println("Newly added static method");

  }

  public void method2();

}
```

7. Se crea una interfaz Inf3 vacía

```
package capitulo7.interfaceclass;
```

```
interface Inf1{
  public void method1();
}
interface Inf2 extends Inf1 {
  default void newMethod(){
    System.out.println("Newly added default method");
  }
  static void anotherNewMethod(){
    System.out.println("Newly added static method");
  }
  public void method2();
}
interface Inf3{}
```

8. Se crea una clase Demo que implemente las interfaces Inf2 y Inf3

```
package capitulo7.interfaceclass;
interface Inf1{
  public void method1();
}
interface Inf2 extends Inf1 {
  default void newMethod(){
    System.out.println("Newly added default method");
  }
  static void anotherNewMethod(){
    System.out.println("Newly added static method");
  }
  public void method2();
}
interface Inf3{}
public class Demo implements Inf2,Inf3{
```

```
}
```

9. Se deben implementar los métodos no implementados de la interfaz Inf2 y Inf1 ya que la primera hereda de la segunda

```
package capitulo7.interfaceclass;

interface Inf1{

  public void method1();

}

interface Inf2 extends Inf1 {

  default void newMethod(){

    System.out.println("Newly added default method");

  }

  static void anotherNewMethod(){

    System.out.println("Newly added static method");

  }

  public void method2();

}

interface Inf3{}

public class Demo implements Inf2,Inf3{

  public void method1(){

    System.out.println("method1");

  }

  public void method2(){

    System.out.println("method2");

  }

}
```

10. Se crea un método main

```
package capitulo7.interfaceclass;

interface Inf1{

  public void method1();
```

```
}
interface Inf2 extends Inf1 {
  default void newMethod(){
    System.out.println("Newly added default method");
  }
  static void anotherNewMethod(){
    System.out.println("Newly added static method");
  }
  public void method2();
}
interface Inf3{}
public class Demo implements Inf2,Inf3{
  public void method1(){
    System.out.println("method1");
  }
  public void method2(){
    System.out.println("method2");
  }

  public static void main(String args[]){
  }
}
```

11. Se crea un objeto Inf2 mediante el constructor de la clase Demo

```
package capitulo7.interfaceclass;
interface Inf1{
  public void method1();
}
interface Inf2 extends Inf1 {
  default void newMethod(){
```

```
    System.out.println("Newly added default method");
  }
  static void anotherNewMethod(){
    System.out.println("Newly added static method");
  }
  public void method2();
}
interface Inf3{}
public class Demo implements Inf2,Inf3{
  public void method1(){
    System.out.println("method1");
  }
  public void method2(){
    System.out.println("method2");
  }

  public static void main(String args[]){
    Inf2 obj = new Demo();
  }
}
```

12. Se llama al método default de la interfaz Inf2

```
package capitulo7.interfaceclass;
interface Inf1{
  public void method1();
}
interface Inf2 extends Inf1 {
  default void newMethod(){
    System.out.println("Newly added default method");
```

```
  }
  static void anotherNewMethod(){
    System.out.println("Newly added static method");
  }
  public void method2();
}
interface Inf3{}
public class Demo implements Inf2,Inf3{
  public void method1(){
    System.out.println("method1");
  }
  public void method2(){
    System.out.println("method2");
  }

  public static void main(String args[]){
    Inf2 obj = new Demo();
    obj.newMethod();
  }
}
```

13. Se llama al método estático de la interfaz Inf2

```
package capitulo7.interfaceclass;
interface Inf1{
  public void method1();
}
interface Inf2 extends Inf1 {
  default void newMethod(){
    System.out.println("Newly added default method");
```

```
  }
  static void anotherNewMethod(){
    System.out.println("Newly added static method");
  }
  public void method2();
}
interface Inf3{}
public class Demo implements Inf2,Inf3{
  public void method1(){
    System.out.println("method1");
  }
  public void method2(){
    System.out.println("method2");
  }

  public static void main(String args[]){
    Inf2 obj = new Demo();
    obj.newMethod();
    Inf2.anotherNewMethod();
  }
}
```

14. Por último se realiza una llamada a los métodos de implementados

```
package capitulo7.interfaceclass;
interface Inf1{
  public void method1();
}
interface Inf2 extends Inf1 {
  default void newMethod(){
    System.out.println("Newly added default method");
```

```
  }
  static void anotherNewMethod(){
    System.out.println("Newly added static method");
  }
  public void method2();
}
interface Inf3{}
public class Demo implements Inf2,Inf3{
  public void method1(){
    System.out.println("method1");
  }
  public void method2(){
    System.out.println("method2");
  }

  public static void main(String args[]){
    Inf2 obj = new Demo();
    obj.newMethod();
    Inf2.anotherNewMethod();
    obj.method1();
    obj.method2();
  }
}
```

15. Output

```
Newly added default method
Newly added static method
method1
method2
```

5.4. Tutorial 4 - Entendiendo el polimorfismo - Polimorfismo

El siguiente tutorial muestra una serie de ejemplos relacionados con el polimorfismo en Java, para ello, se crean una serie de clases que se extienden unas a las otras y se sobreescribe uno de los métodos de una de las clases. Se pretende observar cómo, depende del objeto que se cree y teniendo en cuenta la herencia de estos objetos, se accede a los métodos de unas clases u otras.

1. Se crea un paquete llamado capitulo7.polymorphism

```
package capitulo7.polymorphism;
```

2. Se crea una clase Animal

```
package capitulo7.polymorphism;
public class Animal{
}
```

3. Se crea un método *getSound()* que devuelva el string "*Unknown*"

```
package capitulo7.polymorphism;
public class Animal{
  public String getSound(){
    return “Unknown”;
  }
}
```

4. Se crea otro método *displayInformation()* que muestre por pantalla un mensaje con el output del método anterior.

```
package capitulo7.polymorphism;
public class Animal{
  public String getSound(){
    return “Unknown”;
  }
   public void displayInformation() {
            System.out.println(“Animal is making a sound: “ + getSound()
   }
}
```

5. Se crea una nueva clase *Cat* que herede de *Animal*

```
package capitulo7.polymorphism;
public class Cat extends Animal{
}
```

6. Se sobreescribe el método *getSound()* de la clase Animal para que devuelva otro string.

```
package capitulo7.polymorphism;
public class Cat extends Animal{
  @Override
  public String getSound(){
    return "Meow";
  }
}
```

7. Se crea un método main

```
package

capitulo7.polymorphism;
public class Cat extends Animal{
  @Override
  public String getSound(){
    return "Meow";
  }
  public static void main(String args[]){
  }
}
```

8. Se crea un objeto Animal con el constructor de la clase Cat

```
package capitulo7.polymorphism;
public class Cat extends Animal{
  @Override
  public String getSound(){
    return "Meow";
  }
  public static void main(String args[]){
    Animal cat = new Cat();
  }
}
```

9. Se realiza una llamada al método *displayInformation()*

```
package capitulo7.polymorphism;
public class Cat extends Animal{
  @Override
  public String getSound(){
    return "Meow";
  }
```

```
  public static void main(String args[]){
    Animal cat = new Cat();
    cat.displayInformation();
  }
}
```

10. Output

```
Animal is making a sound: Meow
```

11. Se crea ahora otra nueva clase Horse que herede de Animal

```
package capitulo7.polymorphism;
public class Horse extends Animal{
}
```

12. Se sobrescribe el método *getSound()* de la clase Animal para que devuelva otro string distinto.

```
package capitulo7.polymorphism;
public class Horse extends Animal{
  @Override
  public String getSound(){
    return "Neigh";
  }
}
```

13. Se crea un método main

```
package capitulo7.polymorphism;
public class Horse extends Animal{
  @Override
  public String getSound(){
    return "Neigh";
  }
  public static void main(String args[]){
  }
}
```

14. Se crea un objeto Animal con el constructor de la clase Horse

```
package
```

```
capitulo7.polymorphism;
public class Horse extends Animal{
  @Override
  public String getSound(){
    return "Neigh";
  }
  public static void main(String args[]){
    Animal animal = new Horse();
  }
}
```

15. Se crea un objeto Horse haciendo casting al objeto Animal creado anteriormente

```
package capitulo7.polymorphism;
public class Horse extends Animal{
  @Override
  public String getSound(){
    return "Neigh";
  }
  public static void main(String args[]){
    Animal animal = new Horse();
    Horse horse = (Horse) animal;
  }
}
```

16. Se realiza una llamada al método *displayInformation()*

```
package capitulo7.polymorphism;
public class Horse extends Animal{
  @Override
  public String getSound(){
    return "Neigh";
  }
  public static void main(String args[]){
    Animal animal = new Horse();
    Horse horse = (Horse) animal;
    horse.displayInformation();
  }
}
```

17. Output

```
Animal is making a sound: Neigh
```

18. Se crea una nueva clase *SoundAnimals*

```
package capitulo7.polymorphism;
public class SoundAnimals {
}
```

19. Se crea un método *sound* que reciba como parámetro un objeto *Animal* y muestre un mensaje por pantalla con "Sound Animals: " y el sonido del animal recibido como parámetro.

```
package

capitulo7.polymorphism;
public

class SoundAnimals {
   public static void sound(Animal animal){
            System.out.println("Sound Animal: " + animal.getSound());
   }
}
```

20. Se crea un método main

```
package

capitulo7.polymorphism;
public

class SoundAnimals {
   public static void sound(Animal animal){
            System.out.println("Sound Animal: " + animal.getSound());
   }
   public static void main(String[] args){
}
}
```

21. Se llama a las distintas clases con el método sound(Animal animal)

```
package capitulo7.polymorphism;
public class SoundAnimals {
   public static void sound(Animal animal){
            System.out.println("Sound Animal: " + animal.getSound());
   }
   public static void main(String[] args){
            sound(new Cat());
            sound(new Horse());
            sound(new Animal());
```

```
}
}
```

22. Output

```
Sound Animal: Meow
Sound Animal: Neigh
Sound Animal: Unknown
```

5.5. Tutorial 5 - Escribiendo Lambdas Simples

El siguiente tutorial muestra el contexto en el que se puede hacer uso de una expresión lambda, así como un ejemplo de una expresión lambda sencilla.

1. Se crea un paquete llamado capitulo6.lambdas

```
package capitulo6.lambdas
```

2. Se crea una clase llamada Lambdas dentro del paquete capitulo6.lambdas

```
package capitulo6.lambdas;
public class Lambdas {

}
```

3. Se importa la interfaz Predicate de java.util.

```
package capitulo6.lambdas;
import java.util.function.Predicate;


public class Lambdas {

}
```

4. Se crea el método main

```
package capitulo6.lambdas;
import java.util.function.Predicate


public class Lambdas {
    public static void main(String args[]) {

    }

}
```

5. Se crea la expresión lambda que implemente el método test de Predicate

```
package capitulo6.lambdas;
```

```
public class Lambdas {

    public static void main(String args[]) {

        Predicate<Integer> isPar = num -> (num%2 == 0);


        System.out.println("Result: " + isPar.test(4));
        System.out.println("Result: " + isPar.test(5));

     }

}
```

6. Output

```
Result: true
Result: false
```

El siguiente tutorial muestra el uso de una interfaz y una expresión lambda, así como un ejemplo de cómo utilizarlas.

1. Se crea una interfaz llamada StringModify dentro del paquete capitulo6.lambdas

```
package capitulo6.lambdas;
    public interface StringModify {

}
```

2. Se crea un método en la interfaz de nombre sModify y cuyos parámetros un String a y un String b

```
package capitulo6.lambdas;

public interface StringModify {
    public String sModify(String a, String b);

}
```

3. Se crea una clase llamada Lambdas2 dentro del paquete capitulo6.lambdas

```
package capitulo6.lambdas;
    public class Lambdas2 {

}
```

4. Se crea el método main

```
package capitulo6.lambdas;

public class Lambdas2 {
    public static void main(String args[]) {

    }
```

```
}
```

5. Se crea la expresión lambda que implemente el método sModify de StringModify

```
package capitulo6.lambdas;

public class Lambdas2 {

   public static void main(String args[]) {

      StringModify s = (str1, str2) -> (str1 + str2).toUpperCase();

      System.out.println("Result: " + s.sModify("Hello", "World"));

   }

}
```

6. Output

```
Result: HELLO WORLD
```

6. Tutorial Capítulo 8: Excepciones

6.1. Tutorial 1 - Entendiendo las excepciones

En el siguiente tutorial se pretende lanzar una excepción en tiempo de ejecución, mediante la palabra reservada **throw**.

1. Se crea un paquete llamado capitulo8.throwingruntimeexception.

```
package capitulo8.throwingruntimeexception;
```

2. Se crea una clase llamada 'Person' y una variable de tipo int llamada 'age'.

```
package capitulo8.throwingruntimeexception;

public class Person {

    private int age;

}
```

3. Se crean los métodos 'getAge' y 'setAge' , este último solamente puede recibir por parámetro un número positivo, de lo contrario se va a lanzar una excepción mediante la palabra reservada throw.

```
package capitulo8.throwingruntimeexception;
public class Person {
    private int age;
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        if (age <= 0) {
            throw new RuntimeException("Age should be positive.");
        }
        this.age = age;
    }
}
```

4. Se crea el método main, donde se crea una persona y se establece la edad en -10.

```
package capitulo8.throwingruntimeexception;
public class Person {
    private int age;
    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        if (age <= 0) {
            throw new RuntimeException("Age should be positive.");
        }
        this.age = age;
    }
    public static void main(String[] args) {
        final Person person = new Person();
        person.setAge(-10);
    }

}
```

5. Output.

```
Exception in thread "main" java.lang.RuntimeException: Age should be positive.

   at capitulo8.throwingruntimeexception.Person.setAge(Person.java:30)

   at capitulo8.throwingruntimeexception.Person.main(Person.java:40)
```

6.2. Tutorial 2 - Entendiendo las excepciones

En el siguiente tutorial se pretende mostrar como lanzar una excepción en tiempo de ejecución, mediante la palabra reservada **throws**.

1. Se crea un paquete llamado capitulo8.throwinganexception.

```
package capitulo8.throwinganexception;
```

2. Se crea una clase llamada 'Person' y una variable de tipo int llamada 'age'.

```
package capitulo8.throwinganexception;

public class Person {

    private int age;

}
```

3. Se crea los métodos 'getAge' y 'setAge', en este último se debe indicar que ese método puede lanzar una excepción, eso se hace con la palabra clave throws.

```
package capitulo8.throwinganexception;

public class Person {

    private int age;

    public int getAge() {

        return age;

    }

    public void setAge(int age) throws Exception {

    if (age <= 0) {

            throw new Exception("Age should be positive.");

        }

       this.age = age;

    }

}
```

4. Ahora se crea una persona y se establece una edad, pero en este caso es obligatorio manejar la excepción, de lo contrario no se va a poder ejecutar el programa. Para ello hay que introducir un try-catch. Dentro del bloque try se va a incluir el código que puede lanzar la excepción, mientras que dentro del catch se va a manejar la posible excepcion lanzada dentro del try, en este caso simplemente se va a mostrar el mensaje (Age should be positive) de error que nos proporciona la excepción.

```
package capitulo8.throwinganexception;
public class Person {
```

```
    private int age;
    public int getAge() {
        return age;
    }
    public void setAge(int age) throws Exception {
        if (age <= 0) {
            throw new Exception("Age should be positive.");
        }
        this.age = age;
    }
    public static void main(String[] args) {
        try {
            final Person person = new Person();
            person.setAge(-10);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

5. Output.

```
Age should be positive.
```

6.3. Tutorial 3 - Entendiendo las excepciones

En el siguiente tutorial se pretende mostrar la estructura básica de un bloque try-catch, así como establecer un primer contacto con las excepciones en Java y la palabra reservada **throw**.

1. Se crea un paquete llamado capitulo8.throwinganexception.

```
package capitulo8.throwinganexception;
```

2. Se crea una clase llamada TryCatchThrowExceptionExercise dentro del paquete capitulo8.throwinganexception.

```
package capitulo8.throwinganexception;
public class TryCatchThrowExceptionExercise {
}
```

3. Se crea el método 'fall()' el cual lanza una excepción de tipo 'RuntimeException'.

```
package  capitulo8.throwinganexception;
public class TryCatchThrowExceptionExercise {
  private static void fall() {

     throw new RuntimeException();
```

```
  }
}
```

4. Se crea el método main.

```
package capitulo8.throwinganexception;
public class TryCatchThrowExceptionExercise {
  private static void fall() {

     throw new RuntimeException();

  }
  public static void main(String args[]) {
  }
}
```

5. Se crea un try-catch.

```
package  capitulo8.throwinganexception;
public class TryCatchThrowExceptionExercise {
  private static void fall() {

     throw new RuntimeException();

  }
  public static void main(String args[]) {

    try {
    } catch () {
    }
  }
}
```

6. Dentro del try, se llama al método 'fall()' creado anteriormente.

```
package  capitulo8.throwinganexception;
public class TryCatchThrowExceptionExercise {
  private static void fall() {

     throw new RuntimeException();

  }
  public static void main(String args[]) {
    try {
      fall();
    } catch () {
    }
  }
}
```

7. Al catch, se le pasa como parámetro una excepción de tipo 'RuntimeException' y dentro del catch se crea un mensaje indicando el tipo de error.

```
package  capitulo8.throwinganexception;
public class TryCatchThrowExceptionExercise {
  private static void fall() {

     throw new RuntimeException();

  }
  public static void main(String args[]) {
    try {
      fall();
    } catch (RuntimeException e) {
      System.out.println("RuntimeException");
    }
  }
}
```

8. Fuera del try-catch, se crea un mensaje indicando que el try-catch ha finalizado.

```
package  capitulo8.throwinganexception;
public class TryCatchThrowExceptionExercise {
  private static void fall() {

     throw new RuntimeException();

  }
  public static void main(String args[]) {
    try {
      fall();
    } catch (RuntimeException e) {
      System.out.println("RuntimeException");
    }
    System.out.println("Finished Try-catch.");
  }
}
```

9. Output.

```
RuntimeException.

Finished Try-catch.
```

6.4. Tutorial 4 - Usando try-catch-finally

En el siguiente tutorial se pretende seguir ver como se utilizan las sentencias try y catch incluyendo también la clausula **finally**.

1. Se crea un paquete llamado capitulo8.trycatchfinally.

```
package capitulo8.trycatchfinally;
```

2. Se crea una clase llamada 'TryCatchFinallyExercise'.

```
package capitulo8.trycatchfinally;

public class TryCatchFinallyExercise {

}
```

3. Se crea el método main con un array 'a' inicializado a 2.

```
package capitulo8.trycatchfinally;

public class TryCatchFinallyExercise {

    public static void main(String args[]) {

        int a[] = new int[2];

    }

}
```

4. Se crea un try-catch-finally. Dentro del try se imprimime por pantalla el elemento 3 del array 'a'.

```
package capitulo8.trycatchfinally;

public class TryCatchFinallyExercise {

    public static void main(String args[]) {

        int a[] = new int[2];

        try {

            System.out.println("Access element three:" + a[3]);

        }

    }

}
```

5. Al catch, se le pasa como parámetro una excepción de tipo 'ArrayIndexOutOfBoundsException' y dentro del catch se crea un mensaje indicando el tipo de error.

```
package capitulo8.trycatchfinally;

public class TryCatchFinallyExercise {

    public static void main(String args[]) {

        int a[] = new int[2];

        try {
```

```
            System.out.println("Access element three:" + a[3]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Exception thrown:" + e);
        }
    }
}
```

6. Dentro del finally se le asigna al array en la posición 0 un 6 y se imprimime por pantalla. Además, se imprime un mensaje que dice: 'La sentencia finally se ha ejecutado'.

```
package capitulo8.trycatchfinally;

public class TryCatchFinallyExercise {

    public static void main(String args[]) {
        int a[] = new int[2];
        try {
            System.out.println("Access element three:" + a[3]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Exception thrown:" + e);
        } finally {
            a[0] = 6;
            System.out.println("First element value: " + a[0]);
            System.out.println("The finally statement is executed.");
        }
    }
}
```

7. Output.

```
Exception thrown:java.lang.ArrayIndexOutOfBoundsException: 3
The finally statement is executed.
First element value: 6
```

6.5. Tutorial 5 - Usando try-catch-finally

En el siguiente tutorial se pretende seguir ver como se utilizan las sentencias try y catch incluyendo también la clausula **finally**.

1. Se crea un paquete llamado capitulo8.trycatchfinally.

```
package capitulo8.trycatchfinally;
```

2. Se crea una clase llamada TryCatchFinallyExercise2 dentro del paquete capitulo8.throwinganexception.

```
package capitulo8.trycatchfinally;

public class TryCatchFinallyExercise2 {

}
```

3. Se crea el método 'fall()' el cual lanza una excepción de tipo 'RuntimeException'.

```
package capitulo8.trycatchfinally;
public class TryCatchFinallyExercise2 {
  private static void fall() {

     throw new RuntimeException();

  }
}
```

4. Se crea el método main.

```
package capitulo8.trycatchfinally;
public class TryCatchFinallyExercise2 {
  private static void fall() {

     throw new RuntimeException();

  }
  public static void main(String args[]) {
  }
}
```

5. Se crea un try-catch-finally.

```
package capitulo8.trycatchfinally;
public class TryCatchFinallyExercise2 {
  private static void fall() {

     throw new RuntimeException();

  }
  public static void main(String args[]) {
    try {
    } catch () {
    } finally {
    }
```

```
  }
}
```

6. Dentro del try, se llama al método 'fall()' creado anteriormente.

```
package capitulo8.trycatchfinally;
public class TryCatchFinallyExercise2 {
  private static void fall() {

     throw new RuntimeException();

  }
  public static void main(String args[]) {
    try {
      fall();
    } catch () {
    } finally {
    }
  }
}
```

7. Al catch, se le pasa como parámetro una excepción de tipo 'RuntimeException' y dentro del catch se crea un mensaje indicando el tipo de error.

```
package capitulo8.trycatchfinally;
public class TryCatchFinallyExercise2 {
  private static void fall() {

     throw new RuntimeException();

  }
  public static void main(String args[]) {
    try {
      fall();
    } catch (RuntimeException e) {
      System.out.println("RuntimeException.");
    } finally {
    }
  }
}
```

8. Dentro del finally se crea un mensaje indicando que aun no ha terminado el método.

```
package capitulo8.trycatchfinally;
public class TryCatchFinallyExercise2 {
  private static void fall() {

     throw new RuntimeException();

  }
  public static void main(String args[]) {
```

```
    try {
      fall();
    } catch (Exception e) {
      System.out.println("RuntimeException.");
    } finally {
      System.out.println("I'm finishing.");
    }
  }
}
```

9. Fuera del try-catch-finally, se crea un mensaje, indicando que el try-catch ha finalizado.

```
package capitulo8.trycatchfinally;
public class TryCatchFinallyExercise2 {
  private static void fall() {

     throw new RuntimeException();

  }
  public static void main(String args[]) {
    try {
      fall();
    } catch (Exception e) {
      System.out.println("RuntimeException");
    } finally {
      System.out.println("I'm finishing.");
    }
      System.out.println("Finished Try-catch.");
  }
}
```

10. Output.

```
RuntimeException.

I'm finishing.

Finished Try-catch.
```

6.6. Tutorial 6 - Multiples try-catch

El siguiente tutorial pretende mostrar cómo es posible tener varios bloques catch para un mismo try, teniendo en cuenta para ello las excepciones que pueden darse si se realiza una operación de división entre 0.

1. Se crea un paquete llamado capitulo8.multipletrycatch.

```
package capitulo8.multipletrycatch;
```

2. Se crea una clase llamada MultipleTryCatch dentro del paquete capitulo8.multipletrycatch.

```
package capitulo8.multipletrycatch;
public class MultipleTryCatch {
}
```

3. Se crea el método main.

```
package capitulo8.multipletrycatch;
public class MultipleTryCatch {
  public static void main(String args[]){
  }
}
```

4. Se crea un try-catch.

```
package capitulo8.multipletrycatch;
public class MultipleTryCatch {
  public static void main(String args[]){

    try{
    } catch(){

    }

  }
}
```

5. Dentro del try, se crea un array de tamaño 7, y luego se guarda en la posición 10 el resultado de una división entre cero, y se muestra un mensaje.

```
package capitulo8.multipletrycatch;
public class MultipleTryCatch {
  public static void main(String args[]){

    try {

       int arr[] = new int[7];

       arr[10] = 10 / 0;

       System.out.println("Last Statement of try block.");

    } catch(){

    }

  }
}
```

6. Se crean tres catch para soportar diversas excepciones que se pueden dar. En el primer catch, se le pasa como parámetro la excepción 'ArithmeticException' que indica si se ha cometido alguna excepción aritmética, y dentro del catch se muestra un mensaje.

```
package capitulo8.multipletrycatch;
public class MultipleTryCatch {
  public static void main(String args[]){
    try{
      int arr[] = new int[7];

      arr[10] = 10 / 0;

      System.out.println("Last Statement of try block.");
    } catch(ArithmeticException e){
      System.out.println("You should not divide a number by zero.");
    } catch(){

    } catch(){
    }
  }
}
```

7. En el segundo catch, se le pasa como parámetro una excepción 'ArrayIndexOutOfBoundsException' que indica si se ha accedido a una posición del array que no existe, y dentro del catch se muestra un mensaje.

```
package capitulo8.multipletrycatch;
public class MultipleTryCatch {
  public static void main(String args[]){
   try{
       int arr[] = new int[7];

       arr[10] = 10 / 0;

       System.out.println("Last Statement of try block.");

    } catch(ArithmeticException e){
       System.out.println("You should not divide a number by zero.");
    } catch(ArrayIndexOutOfBoundsException  e){
       System.out.println("Accessing array elements outside of the limit.");
    } catch(){
    }
  }
}
```

8. En el tercer catch, se le pasa como parámetro una excepción genérica, y dentro del catch se muestra un mensaje.

```
package capitulo8.multipletrycatch;
public class MultipleTryCatch {
```

```
  public static void main(String args[]){
   try{
       int arr[] = new int[7];

       arr[10] = 10 / 0;

       System.out.println("Last Statement of try block.");

    } catch(ArithmeticException e){
       System.out.println("You should not divide a number by zero.");
    } catch(ArrayIndexOutOfBoundsException e){
       System.out.println("Accessing array elements outside of the limit.");
    } catch(Exception e){

       System.out.println("Some Other Exception.");
    }
  }
}
```

9. Al final se muestra un mensaje de finalizado del try-catch.

```
package capitulo8.multipletrycatch;
public class MultipleTryCatch {
  public static void main(String args[]){
   try{
       int arr[] = new int[7];

       arr[10] = 10 / 0;

       System.out.println("Last Statement of try block.");

    } catch(ArithmeticException e){
       System.out.println("You should not divide a number by zero.");
    } catch(ArrayIndexOutOfBoundsException  e){
       System.out.println("Accessing array elements outside of the limit.");
    } catch(Exception e){

       System.out.println("Some Other Exception.");
    }

    System.out.println("Out of the try-catch block.");
  }
}
```

10. Output.

```
You should not divide a number by zero.

Out of the try-catch block.
```

6.7. Tutorial 7 - Multiples try-catch

El siguiente tutorial pretende mostrar cómo es posible tener varios bloques catch para un mismo try, teniendo en cuenta para ello las excepciones que pueden darse si se realiza la búsqueda de un fichero que no existe.

1. Se crea un paquete llamado capitulo8.multipletrycatch.

```
package capitulo8.multipletrycatch;
```

2. Se crea una clase llamada MultipleTryCatch2 dentro del paquete capitulo8.multipletrycatch.

```
package capitulo8.multipletrycatch;
public class MultipleTryCatch2 {
}
```

3. Se crea el método main.

```
package capitulo8.multipletrycatch;
public class MultipleTryCatch2 {
  public static void main(String args[]){
  }
}
```

4. Se crea un try-catch.

```
package capitulo8.multipletrycatch;
public class MultipleTryCatch2 {
  public static void main(String args[]){

    try{
    } catch(){

    }

  }
}
```

5. Dentro del try, se accede a un archivo llamado (author.txt) y mientras los valores del archivo sean distintos de -1, imprimirá por pantalla el valor.

```
package capitulo8.multipletrycatch;
public class MultipleTryCatch2 {
  public static void main(String args[]){

    try {

        FileInputStream file = new FileInputStream("author.txt");

        int k;

        while ((k = file.read()) != -1) {

            System.out.print((char) k);

        }
```

```
        file.close();
    } catch(){
    }
  }
}
```

6. Se crean dos catch para soportar diversas excepciones que se pueden dar. En el primer catch, se le pasa como parámetro la excepción 'FileNotFoundException' que indica que no se ha encontrado el archivo, y dentro del catch se muestra un mensaje.

```
package capitulo8.multipletrycatch;
public class MultipleTryCatch2 {
  public static void main(String args[]){
    try{
      FileInputStream file = new FileInputStream("author.txt");
        int k;
        while ((k = file.read()) != -1) {
            System.out.print((char) k);
        }
        file.close();
  } catch(FileNotFoundException e){
       System.out.println("Source file does not exist.\n" + e);
  }
}
```

7. En el segundo catch, se le pasa como parámetro una excepción 'IOException', y dentro del catch se muestra un mensaje.

```
package capitulo8.multipletrycatch;
public class MultipleTryCatch2 {
  public static void main(String args[]){
   try{
       FileInputStream file = new FileInputStream("author.txt");
        int k;
        while ((k = file.read()) != -1) {
            System.out.print((char) k);
        }
        file.close();
```

```
    } catch(FileNotFoundException e){
       System.out.println("Source file does not exist.\n" + e);
    } catch(IOException e){
       System.out.println("Some I/O problem.\n" + e);
    }
  }
}
```

8. Output.

```
Source file does not exist.

java.io.FileNotFoundException: author.txt (El sistema no puede encontrar el archivo e
specificado)
```

6.8. Tutorial 8 - Tipos de excepciones: ArithmeticException

Este tutorial pretende mostrar la excepción Java ArithmeticException, la cual extiende a RuntimeException (siendo por lo tanto una excepción opcional). Esta excepción es lanzada cuando tiene lugar una división entre 0, que es el ejemplo que muestra el código.

1. Se crea un paquete llamado capitulo8.typesofexception.

```
package capitulo8.typesofexception;
```

2. Se crea una clase llamada ArithmeticExceptionExercise dentro del paquete capitulo8.typesofexception.

```
package capitulo8.typesofexception;
class ArithmeticExceptionExercise {
}
```

3. Se crea el método main.

```
package capitulo8.typesofexception;
class ArithmeticExceptionExercise {
  public static void main(String args[]) {
  }
}
```

4. Se crean dos variables: num 1 (entero) y num2 (entero).

```
package capitulo8.typesofexception;
class ArithmeticExceptionExercise {
  public static void main(String args[]) {
    int num1;
    int num2;
```

```
  }
}
```

5. Se crea un try catch.

```
package capitulo8.typesofexception;
class ArithmeticExceptionExercise {
  public static void main(String args[]) {
    int num1;
    int num2;


    try {
    } catch () {
    } catch () {

    }
  }
}
```

6. Dentro del try se inicializan las variables num1 y num2 y se crean dos mensajes indicando que ha finalizado.

```
package capitulo8.typesofexception;
class ArithmeticExceptionExercise {

  public static void main(String args[]) {
    int num1;
    int num2;
    try {
      num1 = 0;
      num2 = 62 / num1;
      System.out.println(num2);
      System.out.println("Try finished.");
    } catch () {
    } catch () {

    }
  }
}
```

7. En el primer catch se le pasa como parámetro una excepción 'ArithmeticException' para la división entre cero y dentro se crea un mensaje indicando el error.

```
package capitulo8.typesofexception;
class ArithmeticExceptionExercise {
  public static void main(String args[]) {
    int num1;
    int num2;
    try {
```

```
      num1 = 0;
      num2 = 62 / num1;
      System.out.println(num2);
      System.out.println("Ha terminado el try");
    } catch (ArithmeticException e) {
      System.out.println("It should not be divided by zero.");
    } catch () {
    }
  }
}
```

8. En el segundo catch se le pasa como parámetro una excepción genérica y dentro se crea un mensaje indicando el error.

```
package capitulo8.typesofexception;
class ArithmeticExceptionExercise {
  public static void main(String args[]) {
    int num1;
    int num2;
    try {
      num1 = 0;
      num2 = 62 / num1;
      System.out.println(num2);
      System.out.println("Ha terminado el try");
    } catch (ArithmeticException e) {
      System.out.println("No se debería dividir entre cero");
    } catch (Exception e) {
      System.out.println("An exception has ocurred.");
    }
  }
}
```

9. Por último, se crea un mensaje indicando que ha finalizado el try-catch.

```
package capitulo8.typesofexception;
class ArithmeticExceptionExercise {
  public static void main(String args[]) {
    int num1;
    int num2;
    try {
      num1 = 0;
      num2 = 62 / num1;
      System.out.println(num2);
      System.out.println("Ha terminado el try");
    } catch (ArithmeticException e) {
      System.out.println("No se debería dividir entre cero");
```

```
    } catch (Exception e) {
      System.out.println("Ha ocurrido una excepción");
    }
    System.out.println("Try-catch has finished.");
 }
}
```

10. Output.

```
It should not be divided by zero.

Try-catch has finished.
```

6.9. Tutorial 9 - Tipos de excepciones: ArrayIndexOutOfBoundsException

Este tutorial pretende mostrar la excepción Java ArrayIndexOutOfBoundsException, la cual extiende a RuntimeException (siendo por lo tanto una excepción opcional). Esta excepción es lanzada cuando se exceden las dimensiones de un array.

1. Se crea un paquete llamado capitulo8.typesofexception.

```
package capitulo8.typesofexception;
```

2. Se crea una clase llamada ArrayIndexOutOfBoundsExceptionExercise dentro del paquete capitulo8.typesofexception.

```
package capitulo8.typesofexception;
class ArrayIndexOutOfBoundsExceptionExercise {
}
```

3. Se crea el método main.

```
package capitulo8.typesofexception;

public class ArrayIndexOutOfBoundsExceptionExercise {

  public static void main(String args[]) {

  }

}
```

4. Se crea un try catch.

```
package capitulo8.typesofexception;

public class ArrayIndexOutOfBoundsExceptionExercise {

  public static void main(String args[]) {

    try{

    } catch(){

    }
```

```
  }
}
```

5. Dentro del try se crea un array de tamaño 10 y se añade a la posición 11 del array el valor 9.

```
package capitulo8.typesofexception;

public class ArrayIndexOutOfBoundsExceptionExercise {

  public static void main(String args[]) {

    try{

      int a[]=new int[10];

      a[11] = 9;

    } catch(){

    }

  }
}
```

6. En el catch se le pasa como parámetro una excepción ArrayIndexOutOfBoundsException, que es lanzada si se añade algún elemento fuera del rango establecido en un array. También se creará dentro del catch un mensaje indicando el tipo de error.

```
package capitulo8.typesofexception;

public class ArrayIndexOutOfBoundsExceptionExercise {

  public static void main(String args[]) {

    try{

      int a[]=new int[10];

      a[11] = 9;

    } catch(ArrayIndexOutOfBoundsException e){

       System.out.println ("It should not be divided by zero.

Try-catch has finished.");

    }

  }
}
```

7. Output.

```
Accessing array elements outside of the limit.
```

6.10. Tutorial 10 - Tipos de excepciones: NullPointerException

Este tutorial pretende mostrar la excepción Java NullPointerException, la cual extiende a RuntimeException (siendo por lo tanto una excepción opcional). Esta excepción es lanzada cuando se hace referencia a ningún objeto (dirección de memoria null).

1. Se crea un paquete llamado capitulo8.typesofexception.

```
package capitulo8.typesofexception;
```

2. Se crea una clase llamada NullPointerExceptionExercise dentro del paquete capitulo8.typesofexception.

```
package capitulo8.typesofexception;

public class NullPointerExceptionExercise {

}
```

3. Se crea el método main.

```
package capitulo8.typesofexception;
public class NullPointerExceptionExercise {
  public static void main(String args[]) {
  }
}
```

4. Se crea un try catch.

```
package capitulo8.typesofexception;

public class NullPointerExceptionExercise {

  public static void main(String args[]) {

    try{

    } catch(){

    }

  }

}
```

5. Dentro del try se crea una variable de tipo String con valor nulo, y se muestra la longitud de la variable.

```
package capitulo8.typesofexception;

public class NullPointerExceptionExercise {

  public static void main(String args[]) {

    try{
```

```
      String str=null;
      System.out.println (str.length());
    } catch(){
    }
  }
}
```

6. En el catch se le pasa como parámetro una excepción concreta indicando que se accede a un valor nulo, cuando espera acceder a un objeto y dentro del catch se crea un mensaje indicando el tipo de error.

```
package capitulo8.typesofexception;
public class NullPointerExceptionExercise {
  public static void main(String args[]) {
    try{
      String str=null;
      System.out.println (str.length());
    } catch(NullPointerException e){
      System.out.println("The object is null.");
    }
  }
}
```

7. Output.

```
The object is null.
```

6.11. Tutorial 11 - Tipos de excepciones: NumberFormatException

Este tutorial pretende mostrar la excepción Java NumberFormatException, la cual extiende a RuntimeException (siendo por lo tanto una excepción opcional). Esta excepción es lanzada cuando no es posible transformar un objeto String a un objeto de algún tipo de datos numérico.

1. Se crea un paquete llamado capitulo8.typesofexception.

```
package capitulo8.typesofexception;
```

2. Se crea una clase llamada NumberFormatExceptionExercise dentro del paquete capitulo8.typesofexception.

```
package capitulo8.typesofexception;

public class NumberFormatExceptionExercise {

}
```

3. Se crea el método main.

```
package capitulo8.typesofexception;

public class NumberFormatExceptionExercise {

  public static void main(String args[]) {

  }

}
```

4. Se crea un try catch.

```
package capitulo8.typesofexception;

public class NumberFormatExceptionExercise {

  public static void main(String args[]) {

    try{

    } catch(){

    }

  }

}
```

5. Dentro del try se crea una variable con valor "XYZ" (texto) y se cambia el tipo de la variable (a entero). Después, se crea un mensaje donde se muestra el valor de la variable.

```
package capitulo8.typesofexception;

public class NumberFormatExceptionExercise {

  public static void main(String args[]) {

    try{

      int num=Integer.parseInt ("XYZ") ;

      System.out.println(num);

    } catch(){

    }

  }

}
```

6. En el catch se le pasa como parámetro una excepción concreta que señaliza si el formato del número es incorrecto y dentro del catch se crea un mensaje indicando el tipo de error.

```
package capitulo8.typesofexception;

public class NumberFormatExceptionExercise {

  public static void main(String args[]) {

    try{

      int num=Integer.parseInt ("XYZ") ;

      System.out.println(num);

    }catch(NumberFormatException e){

      System.out.println("The XYZ value can not be converted to Integer type.");

    }

  }

}
```

7. Output.

```
The XYZ value can not be converted to Integer type.
```

6.12. Tutorial 12 - Tipos de excepciones: FileNotFoundException

Este tutorial pretende mostrar las excepciones Java FileNotFoundException y IOException, las cuales extienden a Exception (siendo por lo tanto una excepciones obligatorias que deben ser tratadas en el código). FileNotFoundException es una excepción lanzada cuando no se encuentra el fichero que se debe leer en la ruta especificada o hay otro problema de lectura del fichero, e IOException es lanzada cuando hay algún problema o interrupción con operaciones de entrada o salida.

1. Se crea un paquete llamado capitulo8.typesofexception.

```
package capitulo8.typesofexception;
```

2. Se crean los import necesarios para controlar las excepciones.

```
package capitulo8.typesofexception;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
```

3. Se crea una clase llamada FileNotFoundExceptionExercise dentro del paquete capitulo8.typesofexception.

```
package capitulo8.typesofexception;

import java.io.FileNotFoundException;
```

```
import java.io.FileReader;
import java.io.IOException;
public class FileNotFoundExceptionExercise {
}
```

4. Se crea el método main.

```
package capitulo8.typesofexception;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
public class FileNotFoundExceptionExercise {
  public static void main(String args[]) {
  }
}
```

5. Se crea un try catch.

```
package capitulo8.typesofexception;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
public class FileNotFoundExceptionExercise {
  public static void main(String args[]) {
    try {
    } catch () {
    } catch () {
    }
  }
}
```

6. Dentro del try se crea una variable tipo FileReader en el que guardaremos la ruta del fichero que queremos utilizar.

```
package capitulo8.typesofexception;
import java.io.FileNotFoundException;
```

```
import java.io.FileReader;
import java.io.IOException;
public class FileNotFoundExceptionExercise {
  public static void main(String args[]) {
    try {
      FileReader fr = new FileReader("C://file.txt");
      System.out.print(c);
    } catch () {
    } catch () {
    }
  }
}
```

7. Se crea un array tipo char de tamaño 50 y se lee el contenido del array.

```
package capitulo8.typesofexception;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
public class FileNotFoundExceptionExercise {
  public static void main(String args[]) {
    try {
      FileReader fr = new FileReader("C://file.txt");
      char [] a = new char[50];
      fr.read(a);
    } catch () {
    } catch () {
    }
  }
}
```

8. Luego se crea un bucle for que recorre el contenido del fichero y lo muestra por pantalla.

```
package capitulo8.typesofexception;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
public class FileNotFoundExceptionExercise {
  public static void main(String args[]) {
    try {
      FileReader fr = new FileReader("C://file.txt");
```

```
      char [] a = new char[50];
      fr.read(a);
      for(char c : a)
        System.out.print(c);
    } catch () {
    } catch () {
    }
  }
}
```

9. En primer catch, se le pasa como parámetro una excepción 'FileNotFoundException' que indica si existe el fichero y dentro del catch, se muestra un mensaje de error.

```
package capitulo8.typesofexception;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

public class FileNotFoundExceptionExercise {

  public static void main(String args[]) {

    try {

      FileReader fr = new FileReader("C://file.txt");

      char [] a = new char[50];

      fr.read(a);

      for(char c : a)

        System.out.print(c);

    } catch (FileNotFoundException e) {

      System.out.println("File not found.");

    } catch () {

    }

  }

}
```

10. En el segundo catch, se le pasa como parámetro una excepción 'IOException' que indica si hay problemas en la lectura y escritura del fichero, y dentro del catch, se muestra un mensaje de error.

```
package capitulo8.typesofexception;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
public class FileNotFoundExceptionExercise {
  public static void main(String args[]) {
    try {
      FileReader fr = new FileReader("C://file.txt");
      char [] a = new char[50];
      fr.read(a);
      for(char c : a)
        System.out.print(c);
    } catch (FileNotFoundException e) {
      System.out.println("No se encontró el archivo");
    } catch (IOException e) {
      System.out.println("File could not be read/write.");
    }
  }
}
```

11. Output.

```
File not found.
```

6.13. Tutorial 13 – Tipos de excepciones: StringIndexOutOfBoundsException

Este tutorial pretende mostrar la excepción Java StringIndexOutOfBoundsException, la cual extiende a Exception (siendo por lo tanto una excepción obligatoria que debe ser tratada en el código). StringIndexOutOfBoundsException es una excepción lanzada cuando se intenta acceder a una posición que no existe en un string.

1. Se crea un paquete llamado capitulo8.typesofexception.

```
package capitulo8.typesofexception;
```

2. Se crea una clase llamada StringIndexOutOfBoundExceptionExercisedentro del paquete capitulo8.typesofexception.

```
package capitulo8.typesofexception;

public class StringIndexOutOfBoundExceptionExercise{

}
```

3. Se crea el método main.

```
package capitulo8.typesofexception;
public class StringIndexOutOfBoundExceptionExercise{
  public static void main(String args[]) {
  }
}
```

4. Se crea un try catch.

```
package capitulo8.typesofexception;
public class StringIndexOutOfBoundExceptionExercise {
  public static void main(String args[]) {
    try {
    } catch () {
    }
  }
}
```

5. Dentro del try se crea un String 'beginnersbook' y se muestra la longitud de la palabra. Se coge la primera posición y luego la posición 40, se muestra por pantalla la posición 40.

```
package capitulo8.typesofexception;
public class StringIndexOutOfBoundExceptionExercise{
  public static void main(String args[]) {
    try {
      String str = "beginnersbook";
      System.out.println(str.length());
      char c = str.charAt(0);
      c = str.charAt(40);
      System.out.println(c);
    } catch () {
```

```
    }
  }
}
```

6. En el catch, se le pasa como parámetro una excepción 'StringIndexOutOfBoundsException' y dentro del catch, se muestra un mensaje de error.

```
package capitulo8.typesofexception;
public class StringIndexOutOfBoundExceptionExercise{
  public static void main(String args[]) {
    try {
      String str = "beginnersbook";
      System.out.println(str.length());
      char c = str.charAt(0);
      c = str.charAt(40);
      System.out.println(c);
    } catch (StringIndexOutOfBoundsException e) {
      System.out.println("The position you are trying to access does not exist,
                                                          outside of the limits.");
    }
  }
}
```

7. Output.

```
13
The position you are trying to access does not exist, outside of the limits.
```