

Tema 3. Manejo de Errores. Excepciones. Afirmaciones

Metodología y Desarrollo de Programas

- Introducción
- Estructura de excepciones
- Clases Exception en Java
- Bibliografía usada

- ¿Qué hacer si de repente se produce un error irreversible en el programa?
Dos posibles soluciones:
 - ❖ Tratarlo mediante una instrucción selectiva:
 - ✓ Mostrar un mensaje de error
 - ✓ Devolver al procedimiento que lo llama un valor especial
 - ✓ Interrumpir la ejecución
 - ❖ No tratarlo: No hacer nada y continuar como si nada hubiera pasado
- C++ y Java posee un mecanismo de gestión de errores incorporado que se denomina *manejo de excepciones*
- La utilización del manejo de excepciones permite gestionar y responder a los errores en tiempo de ejecución
 - ❖ Divisiones por cero
 - ❖ Reserva de memoria no concedida
 - ❖ Apertura de ficheros inexistentes
 - ❖ Lectura de ficheros incorrecta
 - ❖ etc.

➤ Construido a partir de cuatro palabras reservadas:

❖ **try**

- ✓ Bloque de código donde se quiere controlar una circunstancia anómala en la ejecución del programa
- ✓ Si se produce se lanza una excepción, se interrumpe el flujo de ejecución y el control se pasa al manejador de excepciones

❖ **catch**

- ✓ Manejador: Bloque de código donde se captura la excepción producida anteriormente y se procesa decidiendo que hacer
- ✓ Siempre que exista una instrucción try debe existir al menos un catch

❖ **throw**

- ✓ Cuando se produce una excepción, ésta puede ser lanzada por el sistema o por el usuario. En este segundo caso se utiliza la instrucción throw.

- **finally (java)**

- ✓ Instrucciones que se ejecutan siempre tanto si ha existido excepción como si no ha existido

Estructura de excepciones

➤ La forma general de try y catch es:

```
.....  
try {  
    //Bloque try  
}  
catch (type1 arg) {  
    //Bloque catch  
}  
catch (type2 arg) {  
    //Bloque catch  
}  
catch (type3 arg) {  
    //Bloque catch  
}  
....  
catch (typeN arg) {  
    //Bloque catch  
}  
.....
```

Ejemplo 1

```
int main(int argc, char *argv[])  
{  
    try {  
        cout << "Estoy en try"<< endl;  
        throw 10;  
        cout << "Esto no se ejecuta"<< endl;  
    }  
    catch (int i) {  
        cout << "Error: " << i << endl;  
    }  
    catch (double i) { //Este nunca se ejecutaría  
        cout << "Error: " << i << endl;  
    }  
    return 0;  
}  
Salida:  
Estoy en try  
Error: 10
```

Estructura de excepciones

- Ejemplo 2. Generación de una excepción en un bloque no try, siempre que la función si se encuentre dentro de un bloque try

```
void prueba(int valor) {  
    cout << "En prueba: " << valor << endl;  
    if (valor !=5 ) throw valor;  
}  
int main(int argc, char *argv[]) {  
    try {  
        prueba(5);  
        prueba(10);  
        prueba(15); //Nunca se ejecuta  
    }  
    catch (int i) {  
        switch (i){  
            case 10: cout <<"Error: 10" << endl;  
                    break;  
            case 15:cout <<"Error: 15" << endl;  
                    break;  
        }  
    }  
}
```

Salida:

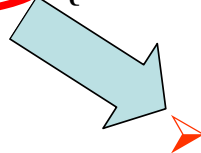
En prueba: 5
En prueba: 10
Error: 10

Estructura de excepciones

➤ Ejemplo 3

- ❖ Reutilización de código --> Fichero interfaz --> ¿Qué excepciones devuelve una función o una clase?
- ❖ Para restringir ó proporcionar información del tipo de excepciones que devuelve una función o clase, en su definición se tienen que indicar los tipos de excepciones que devuelve

```
void prueba(int valor) throw (int, char, double) {  
    if (valor == 0) throw valor;  
    if (valor == 1) throw 'a';  
    if (valor == 2) throw 23.23;  
}  
  
int main(int argc, char *argv[]) {  
    try {  
        prueba(1);  
    }  
    catch (int i) {  
        cout << "Numero: " << i;    }  
    catch (char c) {  
        cout << "Caracter: " << c; }  
    catch (double d) {  
        cout << "Real: " << d;  
    }  
}
```



➤ Solamente puede generar excepción de tipo **int, char o double**

➤ Cuidado:

- ❖ *Siempre* se debe indicar que tipo devuelve
 - ❖ Si no se indica los tipos no **genera** ninguna excepción
- ```
void prueba(int valor) throw () {
```

# Excepciones en Java

- Java incorpora un sistema para que el programador defina la captura y gestión de las excepciones. Además dispone de un GRAN número de excepciones ya definidas

```
try {
 String input = input.readLine();
 int i = Integer.parseInt(input);
 System.out.println("El número es "+i);
} catch (NumberFormatException nfe) {
 System.out.println("El formato del número es erroneo");
} catch (NullPointerException npe) {
 System.out.println("No se ha introducido ningún valor");
}

public static void main(String[] args) {
 try {
 int[] a = new int[5];
 for (int i = 0; i <= 5; i++)
 a[i] = i;
 System.out.println("Llega aquí");
 }
 catch (ArrayIndexOutOfBoundsException e) {
 System.out.println("Error de acceso fuera de limite de matriz");
 System.out.println("Índice erroneo:" + e.getMessage());
 }
 finally {
 System.out.println("finally");
 }
}
```



# Excepciones en Java

---

- Cuando en un programa se arroja una excepción y esta no es capturada, la excepción es capturada por la JVM, mostrando un mensaje parecido a este:

```
Exception in thread "main"
java.lang.NullPointerException
at MiClase.main(MiClase.java:17)
```

- Al producirse un error en un método se genera un objeto que representa el error (Excepción).
- Si el error se genera en un método m, la JVM busca un gestor adecuado dentro del propio método.
- Si el gestor existe, cederá el control a dicho gestor
- Si el gestor no existe, buscará el gestor en el método que haya invocado al método m, y así sucesivamente, hasta encontrar un gestor capaz de tratar la excepción producida

# Excepciones en Java

---

- Java dispone de la cláusula *finally* donde se incluye aquellas instrucciones que se ejecutan tanto si ha existido excepción como si no ha existido
- La cláusula **throws** indica al compilador las excepciones que un método puede lanzar

```
void metodoN() throws IOException, EOFException {
 ...
 // Código que puede lanzar las excepciones IOException y EOFException
 ...
} // Fin del metodo2
```

# Ejemplo

## ➤ Reenviando la excepción

➤ **public static void doio (InputStream in, OutputStream out)  
throws IOException // en caso de más de una excepción throws exp1, exp2**  
{  
    int c;  
    while (( c=in.read()) >=0 )  
    {  
        c= Character.toLowerCase( (char) c);  
        out.write( c );  
    }  
}

*Si la excepción no es capturada, se entiende reenviada*

➤ Alternativamente:

➤ **public static void doio (InputStream in, OutputStream out) throws Throwable {**  
    int c;  
    try { while (( c=in.read()) >=0 )  
    { c= Character.toLowerCase( (char) c);  
        out.write( c );  
    }  
    } catch ( Throwable t ) {  
        throw t;  
    }  
}

*En este caso el método envía una excepción - que aquí corresponde al mismo objeto capturado -por lo tanto debe declararse en la cláusula throws.*

➤ **!!! Si el método usa la cláusula throw debe indicarlo en su declaración con la cláusula throws.**

# Ventajas de usar excepciones: Separar código de casos de error

---

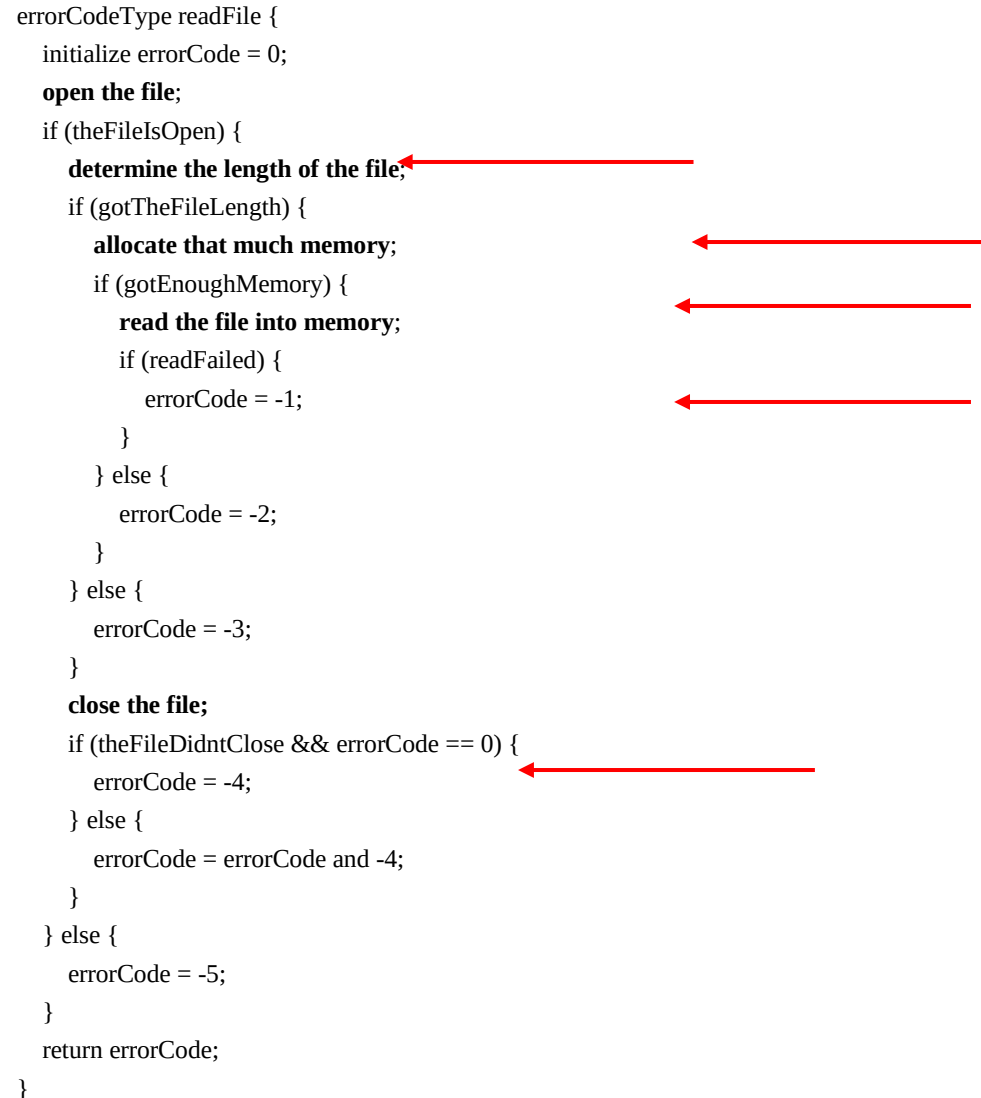
- Supongamos el siguiente ejemplo

```
readFile {
 open the file;
 determine its size;
 allocate that much memory;
 read the file into memory;
 close the file;
}
```

# Ventajas de usar excepciones: Separar código de casos de error

## Sin excepciones

```
errorCodeType readFile {
 initialize errorCode = 0;
 open the file;
 if (theFileIsOpen) {
 determine the length of the file;
 if (gotTheFileLength) {
 allocate that much memory;
 if (gotEnoughMemory) {
 read the file into memory;
 if (readFailed) {
 errorCode = -1;
 }
 } else {
 errorCode = -2;
 }
 } else {
 errorCode = -3;
 }
 close the file;
 if (theFileDidntClose && errorCode == 0) {
 errorCode = -4;
 } else {
 errorCode = errorCode and -4;
 }
 } else {
 errorCode = -5;
 }
 return errorCode;
}
```



The diagram illustrates the error handling logic in the provided code. Red arrows point to the following lines:

- determine the length of the file;**
- allocate that much memory;**
- read the file into memory;**
- errorCode = -1;
- errorCode = -4;

# Ventajas de usar excepciones: Separar código de casos de error

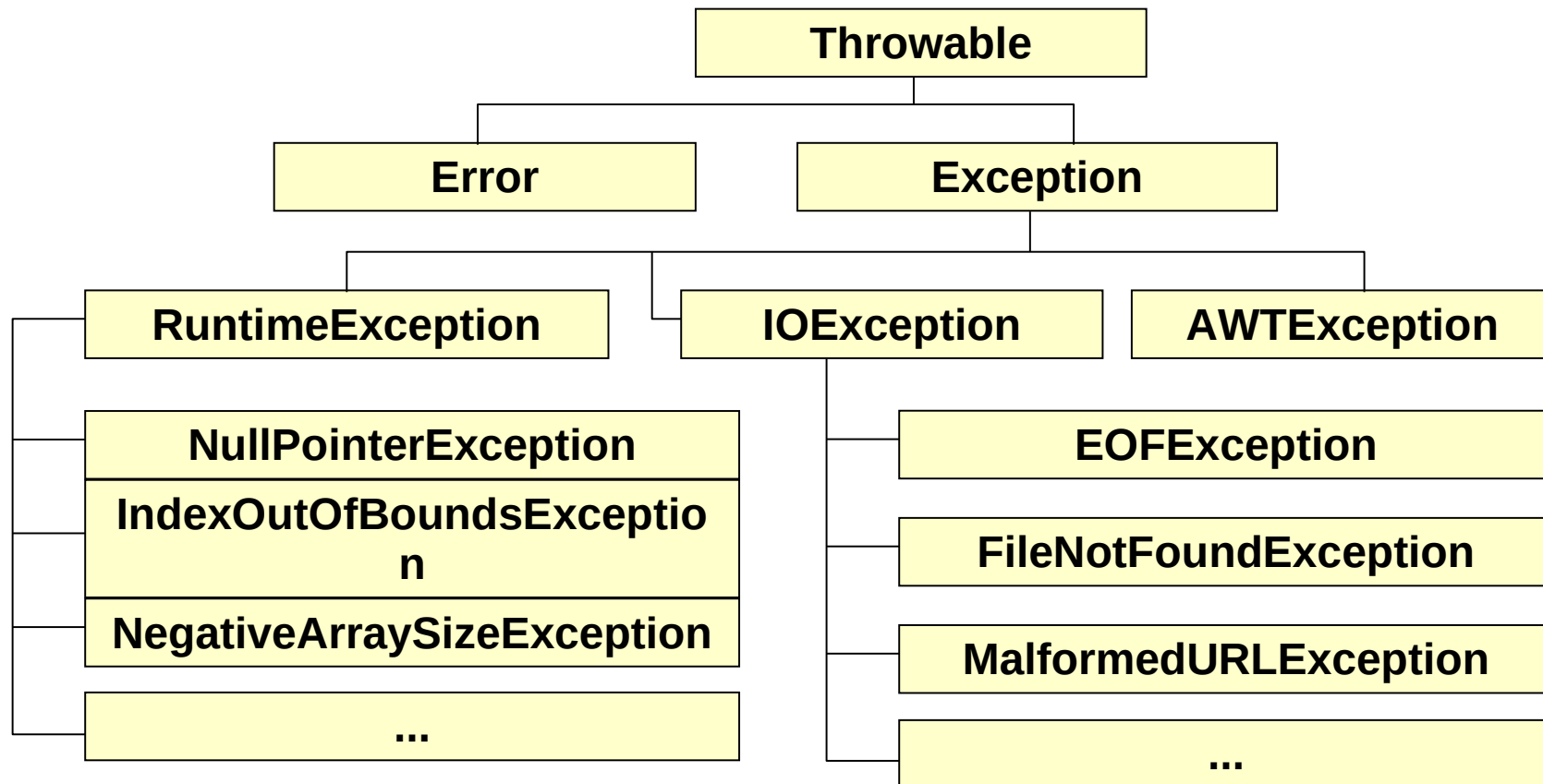
---

## Con excepciones

```
readFile {
 try {
 open the file;
 determine its size;
 allocate that much memory;
 read the file into memory;
 close the file;
 } catch (fileOpenFailed) {
 doSomething;
 } catch (sizeDeterminationFailed) {
 doSomething;
 } catch (memoryAllocationFailed) {
 doSomething;
 } catch (readFailed) {
 doSomething;
 } catch (fileCloseFailed) {
 doSomething;
 }
}
```

# Excepciones en Java

## ➤ Jerarquía de clases en Java



# Excepciones en Java

---

- La clase **java.lang.Throwable** es la superclase de las clases utilizadas para manejar excepciones
- Existen dos tipos de excepciones de la clase Throwable:
  - ❖ **java.lang.Error** representa los errores de compilación y del sistema. Estos errores son ajenos al programador y son **irrecuperables**.
  - ❖ **Java.lang.Exception** representa las excepciones generadas por la aplicación y que deben ser capturadas por el programador mediante (try/catch/finally).
- Dentro de Exception, dos tipos:
  - ❖ **Runtime Exceptions:** Son las excepciones que se producen en el sistema de ejecución de Java tales como referencias null, hacer división por cero, acceder a un elemento inexistente en un array
  - ❖ **NonRuntime Exceptions:** Son excepciones que se producen fuera del sistema de ejecución de Java. Son ejemplo de estas las excepciones que se producen por acceso a archivos (IOExceptions)



# Excepciones en Java

---

- Las clases derivadas de Exception pueden pertenecer a distintos paquetes
  - ❖ java.lang: Throwable, Exception, RuntimeException,...
  - ❖ java.io: FileNotFoundException,...
  - ❖ ....
- Toda clase heredada de Throwable tiene los siguientes métodos
  - ❖ String **getMessage()**: Recupera el mensaje asociado a la excepción
  - ❖ String **toString()** String que describe la excepción
  - ❖ void **printStackTrace()** muestra el método que lanzó la excepción

- Excepciones más comunes:
  - ❖ IOException Generalmente fallas de entrada o salida, tal como la imposibilidad de leer desde un archivo
  - ❖ NullPointerException: Referencia a un objeto NULL
  - ❖ NumberFormatException: Una conversión fallida entre Strings y números
  - ❖ OutOfMemoryException: Muy poca memoria para instanciar un objeto nuevo (new)
  - ❖ SecurityException: Un applet tratando de realizar una acción no permitida por la configuración de seguridad del browser
  - ❖ StackOverflowException: El sistema corriendo fuera de espacio en Stack (stack space)

# Excepciones en Java

---

- Programador se puede crear sus propias excepciones, para lo cual sólo tiene que heredar de la clase Exception
- Exception tiene 4 constructores, de los cuales:
  - ❖ Por defecto (no recibe parámetros)
  - ❖ Parametrizado, recibe una cadena con el mensaje de error

```
class MiExcepcion extends Exception {
 public MiExcepcion() {
 super();
 ...
 }
 public MiExcepcion(String s) {
 super(s);
 ...
 }
}
```

# Ejemplo

```
public class Excepciones {
 public static void main(String[] args) {
 try {
 metodo1();
 } catch (NullPointerException npe) {
 System.out.println("Se ha producido una" +
 "NullPointerException, capturada en el main");
 }
 }

 public static void metodo1() {
 try {
 metodo2();
 } catch (NullPointerException npe) {
 System.out.println("Se ha producido una" +
 "NullPointerException, capturada en el metodo 1");
 }
 }

 public static void metodo2() {
 metodo3();
 }

 public static void metodo3() {
 Object a = null;
 a.toString(); // Esto generará una NullPointerException
 }
}
```

# Ejemplo

---

```
public class TryCatchFinally {
 public static void main(String[] args) {
 try {
 System.out.println("Paso 1");
 int a = 10 / 0; // Lanza una ArithmeticException
 System.out.println("Paso 2");
 } catch (ArithmeticException ae) {
 System.out.println("Paso 3");
 } catch (Exception e) {
 System.out.println("Paso 4");
 }

 try {
 System.out.println("Paso 5");
 int a = 10 / 1; // NO lanza una excepción
 System.out.println("Paso 6");
 Object b = null;
 b.toString(); // Lanza una NullPointerException
 System.out.println("Paso 7");
 } catch (ArithmeticException ae) {
 System.out.println("Paso 8");
 } catch (Exception e) {
 System.out.println("Paso 9");
 }
 }
}
```

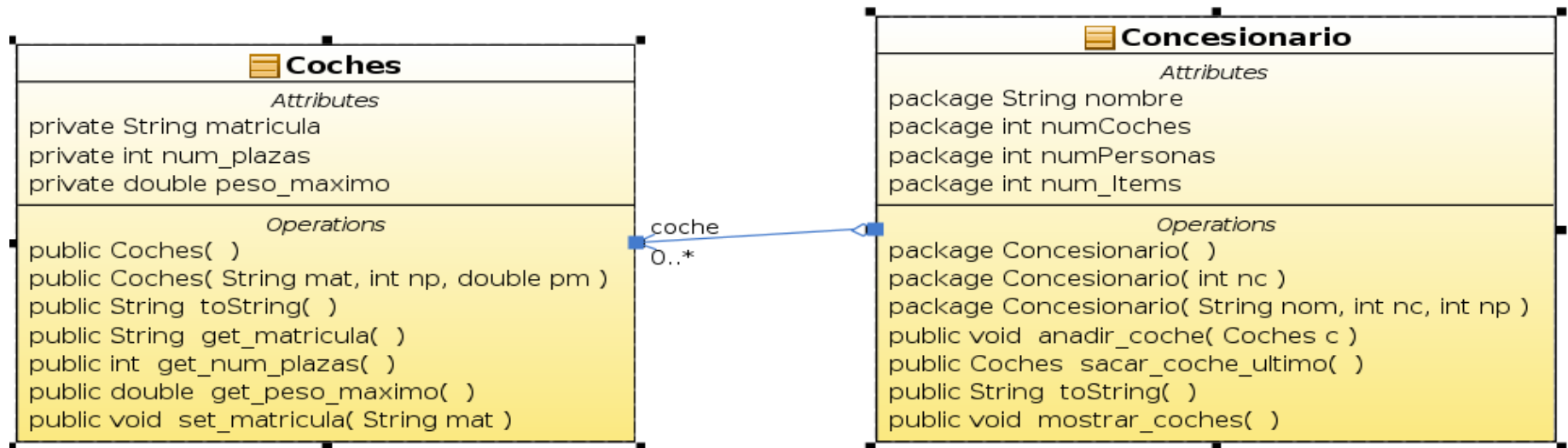
# Ejemplo

```
public class TryCatchFinally {
 public static void main(String[] args) {
 try {
 System.out.println("Paso 1");
 int a = 10 / 0; // Lanza una excepción
 System.out.println("Paso 2");
 } catch (ArithmeticException ae) {
 System.out.println("Paso 3");
 } finally {
 System.out.println("Paso 4");
 }

 try {
 System.out.println("Paso 5");
 int a = 10 / 1; // NO lanza una excepción
 System.out.println("Paso 6");
 } catch (ArithmeticException ae) {
 System.out.println("Paso 7");
 } finally {
 System.out.println("Paso 8");
 }
 }
}
```

# Ejemplo de excepciones

- Informatizar un concesionario de coches. Debe controlarse las distintas excepciones que se puedan producir.
- **Clase Coche**



# Ejemplo de excepciones (Sin excepciones)

```
public class Concesionario {
 String nombre;
 int numCoches;
 int numPersonas;
 Coches coche[];
 int num_Items;

 //Constructor sin parametros
 Concesionario() {
 nombre = "Sin nombre";
 numCoches = 10;
 numPersonas = 50;
 coche = new Coches[numCoches];
 num_Items = 0;
 }
 Concesionario(int nc) {
 nombre = "Sin nombre";
 numCoches = nc;
 numPersonas = 50;
 coche = new Coches[numCoches];
 num_Items = 0;
 }
 Concesionario(String nom, int nc, int np) {
 nombre = nom;
 numCoches = nc;
 numPersonas = np;
 coche = new Coches[nc];
 num_Items = 0;
 }
}
```

```
public boolean anadir_coche(Coches c) {
 if (num_Items >= numCoches) {
 System.out.println("entra en error");
 return false;
 } else {
 coche[num_Items] = c;
 num_Items++;
 return true;
 }
}
//metodo para sacar el ultimo coche que ha entrado

public Coches sacar_coche_ultimo() {
 if (num_Items == 0) {
 System.out.println("entra en error");
 return null;
 } else {
 Coches c = coche[num_Items-1];
 num_Items--;
 return c;
 }
}
//Devolver todos los atributos del concesionario
public String toString() {
 return "Nombre:" + nombre + " Numero maximo coches:" + numCoches + " Numero
maximo de personas:" + numPersonas;
}
public void mostrar_coches() {
 System.out.println("numero de items" + num_Items);
 for (int i = 0; i < num_Items; i++) {
 System.out.println("dentro de for " + i);
 System.out.println(coche[i].toString());
 }
}
```



# Ejemplo de excepciones. Solución sin excepciones.

## ➤ Programa Principal sin excepciones

```
public class Empresa {

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

 Concesionario co = new Concesionario("Merida", 2, 10);

 Coches c = new Coches("4444bbb", 5, 1000.0);
 Coches c1 = new Coches("5555CCC", 2, 1000.0);
 Coches c2 = new Coches();

 If (co.sacar_coche_ultimo()==null)
 System.out.println("No hay coches");
 If(! co.anadir_coche(c))
 System.out.println("No hay hueco");
 If(! co.anadir_coche(c1))
 System.out.println("No hay hueco");
 If(! co.anadir_coche(c2))
 System.out.println("No hay hueco");

 }
}
```

# Ejemplo de excepciones. Solución.

## ➤ Dos excepciones: Lleno y Vacio

```
package concesionario;

public class ConcesionarioLLeno extends Exception{

 private static final long serialVersionUID = 1L;

 public ConcesionarioLLeno(){
 System.out.println("dentro constructor");
 }

 public ConcesionarioLLeno(String msg)
 {
 super(msg);
 }

 public String getMessage()
 {
 return "Error en concesionario LLeno: "+super.getMessage();
 }

}
```

```
package concesionario;

public class ConcesionarioVacio extends Exception{

 private static final long serialVersionUID = 2L;

 public ConcesionarioVacio(){
 System.out.println("dentro constructor");
 }

 public ConcesionarioVacio(String msg) {
 super(msg);
 }

 public String getMessage() {
 return "Error en concesionario Vacio: "+super.getMessage();
 }

}
```

# Ejemplo de excepciones. Solución.

## ➤ Concesionario con excepciones

```
public void anadir_coche(Coches c) throws ConcesionarioLLeno {
 if (num_Items >= numCoches) {
 System.out.println("entra en error");
 throw new ConcesionarioLLeno();
 } else {
 coche[num_Items] = c;
 num_Items++;
 }

}
//metodo para sacar el ultimo coche que ha entrado

public Coches sacar_coche_ultimo() throws ConcesionarioVacio{
 if (num_Items == 0) {
 System.out.println("entra en error");
 throw new ConcesionarioVacio();
 } else {
 Coches c = coche[num_Items-1];
 num_Items--;
 return c;
 }
}
```

# Ejemplo de excepciones. Solución.

## ➤ Programa principal 1

```
public class Empresa {

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

 Concesionario co = new Concesionario("Merida", 2, 10);

 Coches c = new Coches("4444bbb", 5, 1000.0);
 Coches c1 = new Coches("5555CCC", 2, 1000.0);
 Coches c2 = new Coches();

 try {
 Coches temp = co.sacar_coche_ultimo();
 co.anadir_coche(c); //No se ejecuta
 co.anadir_coche(c1); //No se ejecuta
 co.anadir_coche(c2); //No se ejecuta
 } catch (ConcesionarioLLeno ex) {
 System.out.println(ex.getMessage());
 } catch (ConcesionarioVacio ex) {
 System.out.println(ex.getMessage());
 }
 }
}
```

# Ejemplo de excepciones. Solución.

## ➤ Programa principal 2

```
public class Empresa1 {

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

 Concesionario co = new Concesionario("Merida", 2, 10);

 Coches c = new Coches("4444bbb", 5, 1000.0);
 Coches c1 = new Coches("5555CCC", 2, 1000.0);
 Coches c2 = new Coches();
 try {
 Coches temp = co.sacar_coche_ultimo();
 } catch (ConcesionarioVacio ex) {
 System.out.println(ex.getMessage());
 }
 try {
 co.anadir_coche(c);
 co.anadir_coche(c1);
 co.anadir_coche(c2); //Produce excepcion
 co.anadir_coche(c2); //No se ejecuta
 } catch (ConcesionarioLLeno ex) {
 System.out.println(ex.getMessage());
 }
 }
}
```

# Ejemplo de excepciones. Solución.

## ➤ Programa principal 2

```
public class Empresa2 {

 public static void main(String[] args) throws IOException //public
 static void main(String[] args)
 {
 Concesionario co = new Concesionario("Merida", 2, 10);
 InputStreamReader reader = new InputStreamReader(System.in);
 BufferedReader Input = new BufferedReader(reader);
 int opcion=0;
 do {
 System.out.println(" 1. Sacar 2 Añadir 3. Salir");
 opcion= Integer.parseInt(Input.readLine());
 try {
 if (opcion == 1) {
 System.out.println(co.sacar_coche_ultimo().toString());
 } else if (opcion == 2) {
 Coches c = new Coches("4444bbb", 5, 1000.0);
 co.anadir_coche(c);
 }
 } catch (ConcesionarioLLeno ex) {
 System.out.println(ex.getMessage());
 } catch (ConcesionarioVacio ex) {
 System.out.println(ex.getMessage());
 }
 } while (opcion != 3);
 }
}
```

# Ejemplo de excepciones. Solución.

## ➤ Programa Principal 3

```
public class Empresa3 {

 public static void main(String[] args) //public static void main(String[] args)
 {
 //Nuestra empresa tiene 10 concesionario
 Concesionario[] conc = new Concesionario[10];

 for (int i = 0; i < 10; i++) {
 conc[i] = new Concesionario((int) (Math.random() * 6));
 }

 for (int i=0;i<10;i++) { //Por cada concesionario
 int cantidad_coches_meter=(int) (Math.random() * 6);

 for (int j=0;j<cantidad_coches_meter;j++) { //coches a insertar
 Coches aux= new Coches();
 try {
 conc[i].anadir_coche(aux);
 } catch (ConcesionarioLLeno ex) {
 System.out.println(ex.getMessage());
 }
 }
 }
 }
}
```

# Ejemplo de excepciones. Solución.

## ➤ Programa Principal 4

```
public class Empresa4 {

 public static void main(String[] args) //public static void main(String[]
args)
 {
 //Nuestra empresa tiene 10 concesionario
 Concesionario[] conc = new Concesionario[10];

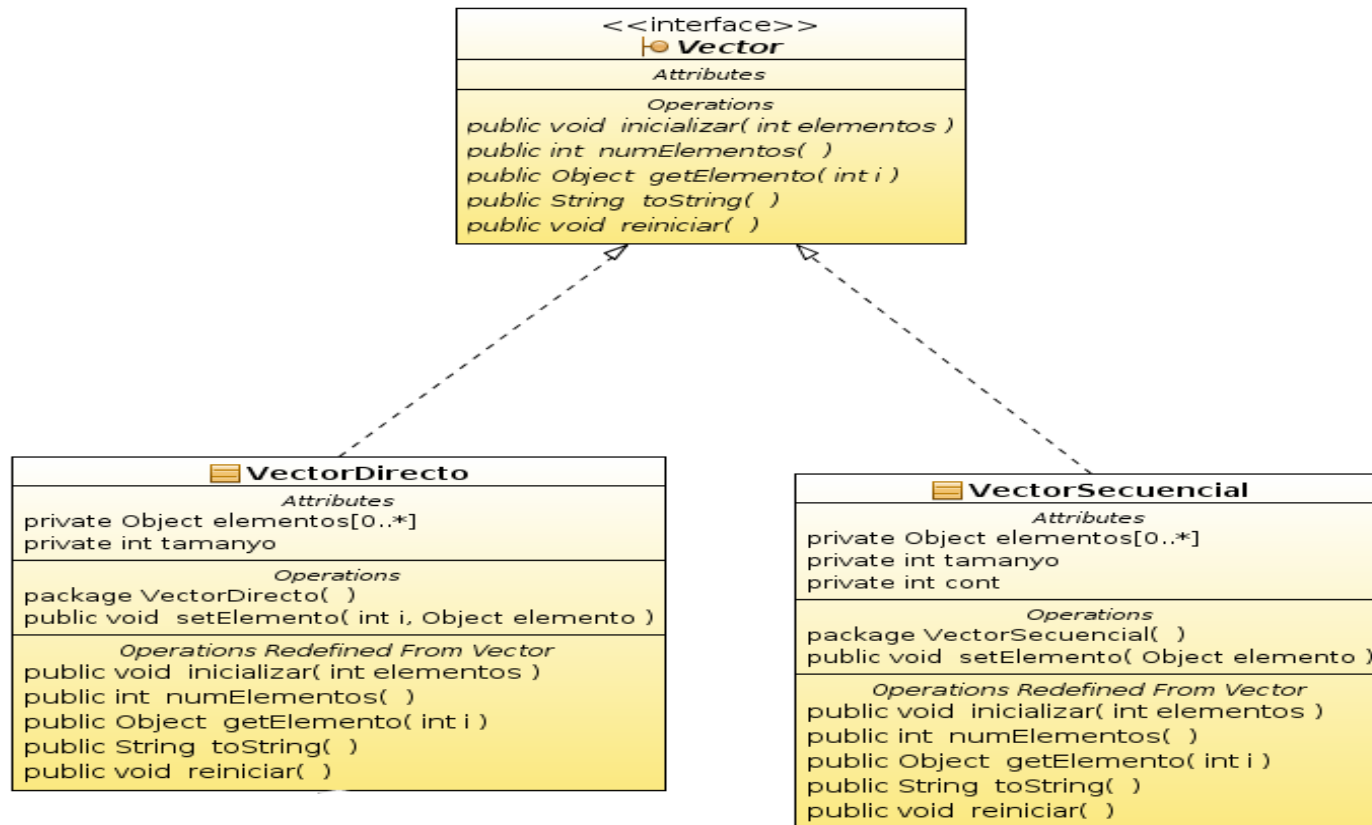
 for (int i = 0; i < 10; i++) {
 conc[i] = new Concesionario((int) (Math.random() * 6));
 }

 for (int i = 0; i < 10; i++) {
 int max = (int) (Math.random() * 6);
 try {
 for (int j = 0; j < max; j++) {
 Coches aux = new Coches();
 conc[i].anadir_coche(aux);
 }
 } catch (ConcesionarioLLeno ex) {
 System.out.println(ex.getMessage());
 }
 }
 }
}
```



# Ejemplo de excepciones. VectorDirecto

- Implementación de la clase VectorDirecto donde se pueden producir tres excepciones:
  - ❖ VectorLleno: Se intenta añadir un objeto nuevo cuando esta lleno
  - ❖ FueraRango: Recuperar o acceder a una posición que no existe
  - ❖ DatoNoValido: Recuperar una posición que no ha sido usada previamente



# Ejemplo de excepciones. VectorDirecto

```
/**
 * * Implementación del set de la clase Vectordirecto, el cual sirve para introducir un
 * elemento en una posición del vector indicada por el usuario
 * @param pos Posición del vector en la que se va a insertar el elemento. La posición va desde 1 hasta
n
 * @param elemento Tipo de elemento que queremos insertar(Electrodomestico,Mueble,etc)
 * @throws FueraRango Excepción que contro la si la posición seleccionada está dentro del intervalo
de posiciones del vector
 */
public void setElemento(int pos, Object elemento) throws FueraRango{

 if((pos>tamanyo)|| (pos==0)){//Si la posición en la que quiero introducir está fuera del
intervalo del vector(es mayor que el tamaño del vector)
 throw new FueraRango();//Llamamos a la excepción
 }else{
 if(vectordir[pos-1]==null){//Si no había ningún elemento en esa posición, introducimos
uno nuevo e incrementamos el contador
 vectordir[pos-1]=elemento;
 contador++;
 }else{//Si la posición ya estaba ocupada y queremos introducir un elemento, lo insertamos
pero no incrementamos el contador
 vectordir[pos-1]=elemento;
 }
 }
}
```

# Ejemplo de excepciones. VectorDirecto

```
/**
 * Implementación del método getElemento del interface Vector el cual devuelve el objeto hay en
 * la posición del vector directo indicada
 * @param pos Posición del vector directo en la que se encuentra el objeto que queremos mostrar.
 * La posición va desde la 1 hasta n
 * @return Devuelve el objeto que se encuentra en la posición del vector directo que hemos
 * pasado por parámetro
 * @throws DatoNoValido Excepción que controla si la posición seleccionada está vacía
 * @throws FueraRango Excepción que contro la si la posición seleccionada está dentro
 * del intervalo de posiciones del vector
 */
public Object getElemento(int pos) throws DatoNoValido, FueraRango{

 if(pos > tamaño || (pos == 0)){//Si la posición en la que quiero introducir está fuera del
 intervalo del vector(es mayor que tamaño del vector)
 throw new FueraRango();//Llamamos a la excepción
 }else{
 if(vectordir[pos-1] == null){//Si en la posición no hay nada
 throw new DatoNoValido();//Llamamos a la excepción
 }else{
 return vectordir[pos-1];
 }
 }
 }
}
```

# Bibliografía Recomendada

---

## ➤ Libros C++:

- ❖ C++ Guia de autoenseñanza. Herbert Schildt.
- ❖ Programación en C++ : algoritmos, estructuras de datos y objetos. Luis Joyanes Aguilar. [S004.43C++joy]
- ❖ Como programar en C++ . H. M. Deitel. [S004.43C++dei]
- ❖ Resolución de problemas con C++. Savitch. [S004.43C++sav]

## ➤ Manuales en Internet C++

- ❖ Aprenda C++ como si estuvieras en primero. Universidad de Navarra.

## ➤ Libros Java

- ❖ **Piensa en Java. 4ª Edición. Bruce Eckel. Pearson Prentice Hall.**
- ❖ Core Java 2. Autores Cay S. Horstmann Y Gary Cornell. Editorial Pearson Educación
- ❖ Java 2. Manual De Programación. Luis Joyanes Aguilar; Matilde Fernández Azuela. Editorial McGraw-Hill