



# PROGRAMACIÓN

## UNIDAD 7: Control y manejo de excepciones

# 1. ¿Qué es una excepción?



Una excepción es un problema que surge durante la ejecución de un programa. Una excepción puede ocurrir por muchas razones diferentes, por ejemplo:

- ▶ Un usuario ha introducido datos no válidos.
- ▶ Un archivo que necesita ser abierto no se puede encontrar .
- ▶ Una conexión de red se ha perdido en el medio de las comunicaciones o la JVM se ha quedado sin memoria.
- ▶ Etc...

Algunas de estas excepciones son causadas por error del usuario, otras por error del programador, y otras por los recursos físicos que han fallado de alguna manera.

## 2. Captura y tratamiento de excepciones

Vamos a ver tres de las palabras reservadas para la captura y tratamiento de excepciones:

- ▶ Try
- ▶ Catch
- ▶ Finally



## 2. Captura y tratamiento de excepciones



### BLOQUE TRY

Todo el código que vaya dentro de esta sentencia será el código sobre el que se intentará capturar el error si se produce y una vez capturado hacer algo con él. Lo ideal es que no ocurra un error, pero en caso de que ocurra un bloque try nos permite estar preparados para capturarlo y tratarlo. Así un ejemplo sería:

```
try {  
    System.out.println("bloque de código donde pudiera saltar un error es este");  
}
```

## 2. Captura y tratamiento de excepciones



### BLOQUE CATCH

En este bloque definimos el conjunto de instrucciones necesarias o de tratamiento del problema capturado con el bloque try anterior. Es decir, cuando se produce un error o excepción en el código que se encuentra dentro de un bloque try, pasamos directamente a ejecutar el conjunto de sentencias que tengamos en el bloque catch. Esto no es exactamente así pero ya explicaremos más adelante todo el funcionamiento. De momento para una mejor comprensión vamos a considerar que esto es así.

```
catch (Exception e) {  
    System.out.println("bloque de código donde se trata el problema");  
}
```

Fíjate que después de catch hemos puesto unos paréntesis donde pone “Exception e”. Esto significa que cuando se produce un error Java genera un objeto de tipo Exception con la información sobre el error y este objeto se envía al bloque catch.

## 2. Captura y tratamiento de excepciones



### BLOQUE FINALLY

Y para finalizar tenemos el bloque finally que es un bloque donde podremos definir un conjunto de instrucciones necesarias tanto si se produce error o excepción como si no y que por tanto se ejecuta siempre.

```
finally {  
    System.out.println("bloque de código ejecutado siempre");  
}
```

## 2. Captura y tratamiento de excepciones

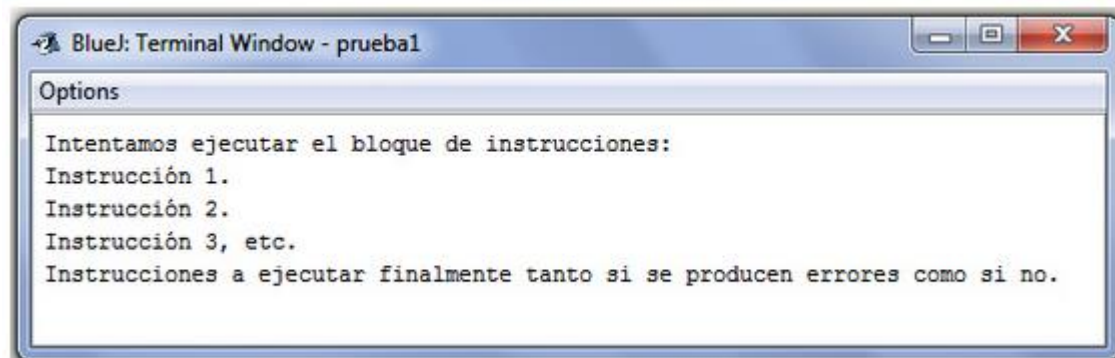


### EJEMPLO SIN ERROR

A continuación vamos a ver cómo se comporta un programa con tratamiento de errores pero donde no se produce ningún error.

```
/* Ejemplo Gestión de Excepciones Java aprenderaprogramar.com */
public class Programa {
    public static void main (String [] args) {
        try{
            System.out.println("Intentamos ejecutar el bloque de instrucciones:");
            System.out.println("Instrucción 1.");          System.out.println("Instrucción 2.");
            System.out.println("Instrucción 3, etc.");
        }
        catch (Exception e) { System.out.println("Instrucciones a ejecutar cuando se produce un error"); }
        finally{ System.out.println("Instrucciones a ejecutar finalmente tanto si se producen errores como si no."); }
    }
}
```

La salida obtenida tras ejecutar el programa anterior es:

A screenshot of a terminal window titled "BlueJ: Terminal Window - prueba1". The window displays the output of the Java program, which consists of five lines of text: "Intentamos ejecutar el bloque de instrucciones:", "Instrucción 1.", "Instrucción 2.", "Instrucción 3, etc.", and "Instrucciones a ejecutar finalmente tanto si se producen errores como si no.".

```
BlueJ: Terminal Window - prueba1
Options
Intentamos ejecutar el bloque de instrucciones:
Instrucción 1.
Instrucción 2.
Instrucción 3, etc.
Instrucciones a ejecutar finalmente tanto si se producen errores como si no.
```

## 2. Captura y tratamiento de excepciones



### EJEMPLO CON ERROR

A continuación vamos a ver cómo se comporta un programa con tratamiento de errores cuando se produce un error y cómo afecta al control de flujo del programa.

```
/* Ejemplo Gestión de Excepciones Java aprenderaprogramar.com */
public class Programa {
    public static void main (String [] args) {
        try {
            System.out.println("Intentamos ejecutar el bloque de instrucciones:");
            System.out.println("Instrucción 1.");
            int n = Integer.parseInt("M"); //error forzado en tiempo de ejecución.
            System.out.println("Instrucción 2.");
            System.out.println("Instrucción 3, etc.");
        }
        catch (Exception e) {
            System.out.println("Instrucciones a ejecutar cuando se produce un error");
        }
        finally {
            System.out.println("Instrucciones a ejecutar finalmente tanto si se producen errores como si no.");
        }
    }
}
```



## 2. Captura y tratamiento de excepciones

Se produce un error porque el método `parseInt` de la clase `Integer` espera que dentro de las comillas llegue un número y no una letra. Por ejemplo `int n = Integer.parseInt("65");` sirve para transformar el `String` `65` en un `int` de valor `65`. Al no encontrar un valor válido se produce un error de tipo `java.lang.NumberFormatException`.

La salida obtenida en este caso donde se produce error es:

```
Blue: Terminal Window - prueba1
Options
Intentamos ejecutar el bloque de instrucciones:
Instrucción 1.
Instrucciones a ejecutar cuando se produce un error
Instrucciones a ejecutar finalmente tanto si se producen errores como si no.
```

Si ahora escribimos dentro del bloque `catch` lo siguiente:

```
System.out.println("Se ha producido un error " + e );
```

Se nos mostrará información referente al tipo de excepción que se ha producido, almacenada en `e`.



## 2. Captura y tratamiento de excepciones

### RESUMEN

Ejecutamos las instrucciones del bloque try que no dan errores, pero cuando en una instrucción se produce un error o excepción inesperada se deja de ejecutar el código del bloque try, y pasamos a ejecutar el código del bloque catch. Hay un salto o cambio en el flujo del programa.

Finalmente se ejecutan, en todo caso, las instrucciones del bloque finally.

El bloque finally no es obligatorio, es decir, puede existir un bloque try catch y no existir bloque finally.



### 3. Lanzar y capturar excepciones

Puede suceder que queramos que nuestro código lance una excepción en una situación determinada.

Para lanzar una excepción en tiempo de ejecución vamos a utilizar la palabra clave **throw** junto a una instancia de la excepción que queremos lanzar.

Veámoslo con un ejemplo.

Vamos a crear una clase “Persona” con el método “setEdad” el cual solamente puede recibir por parámetro un número positivo, de lo contrario se va a lanzar una **excepción**.



### 3. Lanzar y capturar excepciones



```
1 public class Persona {  
2  
3     private int edad;  
4  
5     public int getEdad() {  
6         return this.edad;  
7     }  
8  
9     public void setEdad(int edad) {  
10        if (edad <= 0)  
11            throw new RuntimeException("La edad debe ser positiva");  
12        this.edad = edad;  
13    }  
14 }  
15 }
```

Si ejecutamos el siguiente código, donde creamos una persona y le establecemos la edad en -10, nuestro programa va a terminar su ejecución con la **excepción RuntimeException**:

```
1 public static void main(String[] args) {  
2     Persona persona = new Persona();  
3     persona.setEdad(-10);  
4 }
```

# 3. Lanzar y capturar excepciones



Para que nuestro programa no termine su ejecución con un error vamos a controlar la **excepción** utilizando un **try catch**:

```
1 public static void main(String[] args) {  
2     try {  
3         Persona persona = new Persona();  
4         persona.setEdad(-10);  
5     } catch (RuntimeException e) {  
6         System.out.println(e.getMessage());  
7     }  
8 }
```

Dentro del bloque **try** vamos a incluir el código que puede lanzar la **excepción**, mientras que dentro del **catch** vamos a manejar la posible **excepcion** lanzada dentro del **try**, en nuestro caso simplemente vamos a mostrar el mensaje (“La edad debe ser positiva”) de error que nos proporciona la **excepción**.

### 3. Lanzar y capturar excepciones



Si en lugar de lanzar una excepción en **tiempo de ejecución** lo queremos hacer en **tiempo de compilación** debemos utilizar una instancia de la clase **Exception**, pero con la diferencia de que en el método “setEdad” debemos indicar que ese método puede lanzar esa excepción, eso se puede hacer con la palabra clave **throws** como se muestra en el siguiente ejemplo:

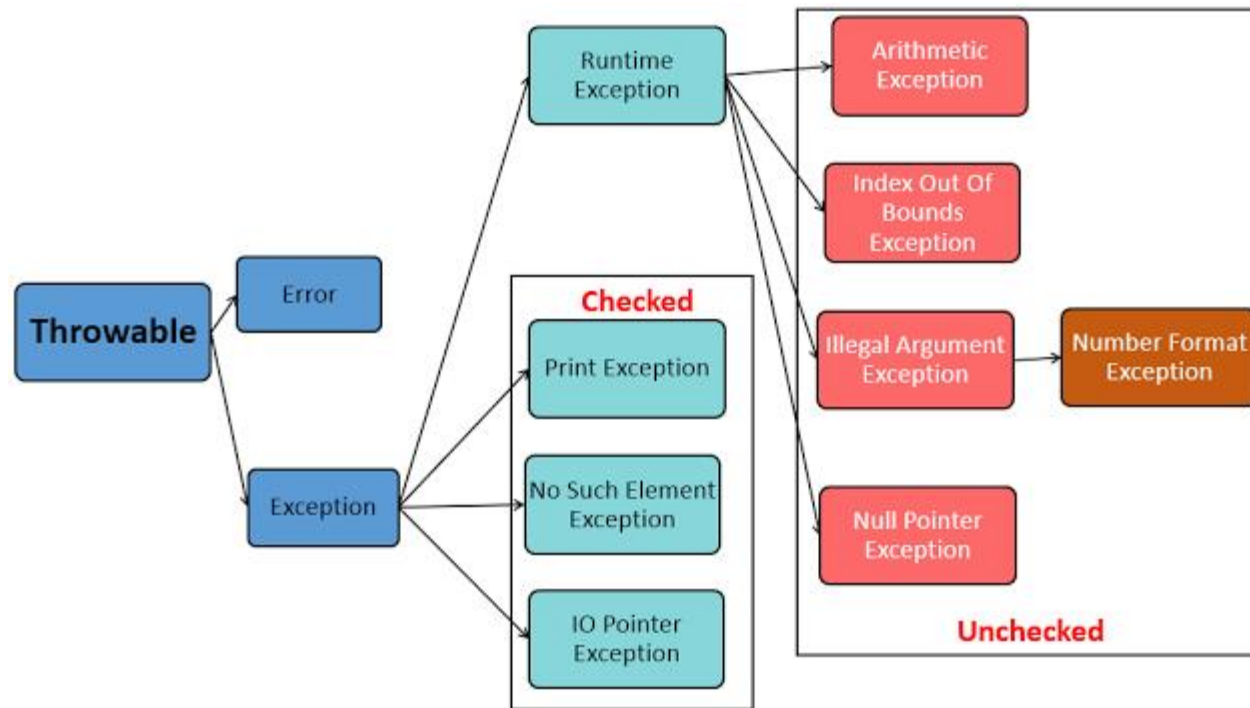
```
1 public void setEdad(int edad) throws Exception {  
2     if (edad <= 0)  
3         throw new Exception("La edad debe ser positiva.");  
4     this.edad = edad;  
5 }
```

Ahora si creamos una persona y le establecemos una edad, estamos obligados a manejar la **excepción**, de lo contrario no vamos a poder ejecutar nuestro programa:

```
1 public static void main(String[] args) {  
2     try {  
3         Persona persona = new Persona();  
4         persona.setEdad(-10);  
5     } catch (Exception e) {  
6         System.out.println(e.getMessage());  
7     }  
8 }
```

# 3. Lanzar y capturar excepciones

Excepciones Verificadas y No Verificadas



# 3. Lanzar y capturar excepciones

## Excepciones Verificadas y No Verificadas



Vamos a establecer una clasificación sencilla para las excepciones en JAVA y es que las excepciones pueden ser:

- **Checked** (Verificadas), todas aquellas clases que hereden de la clase Exception, exceptuando las clases que sean hija de RuntimeException. Se deben declarar en el método mediante la palabra **throws** y obligan al método que lo llama a realizar el manejo de dicha excepción.
- **Unchecked** (No verificadas) las clases que hereden de la clase RuntimeException. No es necesario declararlas en el método y no obliga al método que lo llama a hacer un tratamiento de dicha excepción



## 4. Creando nuestras propias excepciones



Aprovechando dos de las características más importantes de Java, la herencia y el polimorfismo, podemos crear nuestras propias excepciones de forma muy simple:

```
1 class ExcepcionPropia extends Exception{  
2 }
```

Para crear nuestra propia excepción debemos extender la clase de la clase Exception de Java

## 4. Creando nuestras propias excepciones



Sigamos todo el proceso mediante un ejemplo:

La siguiente clase **FondosInsuficientesExcepcion** es una excepción definida por el usuario que extiende de la clase `Exception`. Esta excepción comprueba que los fondos sean suficientes dentro de una cuenta.

```
1  import java.io.*;
2
3  public class FondosInsuficientesExcepcion extends Exception
4  {
5      private double cantidad;
6      public FondosInsuficientesExcepcion(double cantidad)
7      {
8          this.cantidad = cantidad;
9      }
10     public double getCantidad()
11     {
12         return cantidad;
13     }
14 }
```

## 4. Creando nuestras propias excepciones



Para demostrar el uso de nuestra excepción propia, vamos a crear la siguiente clase **CuentaCorriente** que contiene un método *retirar()*, dicha clase lanza una **FondosInsuficientesExcepcion** de no conseguir la cantidad necesaria.

```
1  import java.io.*;
2
3  public class CuentaCorriente
4  {
5      private double balance;
6      private int numero;
7      public CuentaCorriente(int numero)
8      {
9          this.numero = numero;
10     }
11     public void deposito(double cantidad)
12     {
13         balance += cantidad;
14     }
```

## 4. Creando nuestras propias excepciones

```
15 public void retiro(double cantidad) throws
16         FondosInsuficientesExcepcion
17 {
18     if(cantidad <= balance)
19     {
20         balance -= cantidad;
21     }
22     else
23     {
24         double resta = cantidad - balance;
25         throw new FondosInsuficientesExcepcion(resta);
26     }
27 }
28 public double getBalance()
29 {
30     return balance;
31 }
32 public int getNumero()
33 {
34     return numero;
35 }
36 }
```

Observemos que en la clase anterior lanzamos una excepción si la cantidad que queremos retirar es mayor a la cantidad que tenemos disponible en el balance. Para lanzar dicha excepción propia usamos la palabra clave **throw**.



## 4. Creando nuestras propias excepciones

El siguiente programa **Banco** demuestra la invocación de un *deposito()* y un *retirar()* de **CuentaCorriente**:



```
1  public class Banco
2  {
3      public static void main(String [] args)
4      {
5          CuentaCorriente c = new CuentaCorriente(973645);
6          System.out.println("Deposito $500");
7          c.deposito(500.00);
8          try
9          {
10             System.out.println("\nRetiro $100");
11             c.retiro(100.00);
12             System.out.println("\nRetiro $600");
13             c.retiro(600.00);
14         } catch (FondosInsuficientesExcepcion e)
15         {
16             System.out.println("No puede retirar, fondos insuficientes");
17             e.printStackTrace();
18         }
19     }
20 }
```