PROGRAMACIÓN ORIENTADA A OBJETOS:

TEMA 8. MANEJO DE EXCEPCIONES

Presenta: Mtro. David Martínez Torres
Universidad Tecnológica de la Mixteca
Instituto de Computación
Oficina No. 37
dtorres@gs.utm.mx

Contenido

1. Concepto de manejo de excepciones.
2. Implementación de excepciones.
3. Jerarquía de excepciones.

Introducción

Las **excepciones** son un mecanismo especial para gestionar errores y permiten separar el tratamiento de errores del código normal de un programa [5].

Estos errores pueden ser generados por la **lógica** del programa, como un índice de un arreglo fuera de su rango, una división por cero, etc.

En algunos casos las excepciones no se podrán gestionar, por ejemplo los de la propia JVM, y en otros casos sí.

1. Concepto de manejo de excepciones.

- En Java, cuando un evento excepcional ocurre se dice que se **lanza una excepción**.

- El código responsable de hacer alguna acción cuando se arroja una excepción es llamado **manejador de excepciones** y lo que hace es **cachar la excepción lanzada**.

1. Concepto de manejo de excepciones.

El siguiente ejemplo muestra un caso común de excepción en tiempo de ejecución.

```
public class Prueba1 {

    public static void main(String[] args) {
        int[] enteros = { 1, 2, 3, 4, 5 };
        int i;
        // error se está accediendo a una posición fuera del arreglo.
        for (i = 0; i <= 5; i++)
            System.out.println("valor de enteros[" + i + "]: " + enteros[i]);
    }

}
```

Se intenta acceder a una posición fuera del arreglo.

Console

```
<terminated> Prueba1 (23) [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (04/09/2019
valor de enteros[0]: 1
valor de enteros[1]: 2
valor de enteros[2]: 3
valor de enteros[3]: 4
valor de enteros[4]: 5
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
        at cursoPOO.presentacion.Prueba1.main(Prueba1.java:10)
```

1. Concepto de manejo de excepciones.

Otra excepción común es intentar abrir un archivo inexistente. Aunque para esto requiere agregar throws o gestionar la excepción.

Prueba2.java

```
1  package cursoPOO.presentacion;
2
3  import java.io.FileReader;
4
5  public class Prueba2 {
6      public static void main(String[] args) {
7
8          FileReader archivo = new FileReader("este archivo no existe");
9
10     }
11
12 }
13
```

Unhandled exception type FileNotFoundException

2 quick fixes available:

- Add throws declaration
- Surround with try/catch

Press 'F2' for focus

1. Concepto de manejo de excepciones.

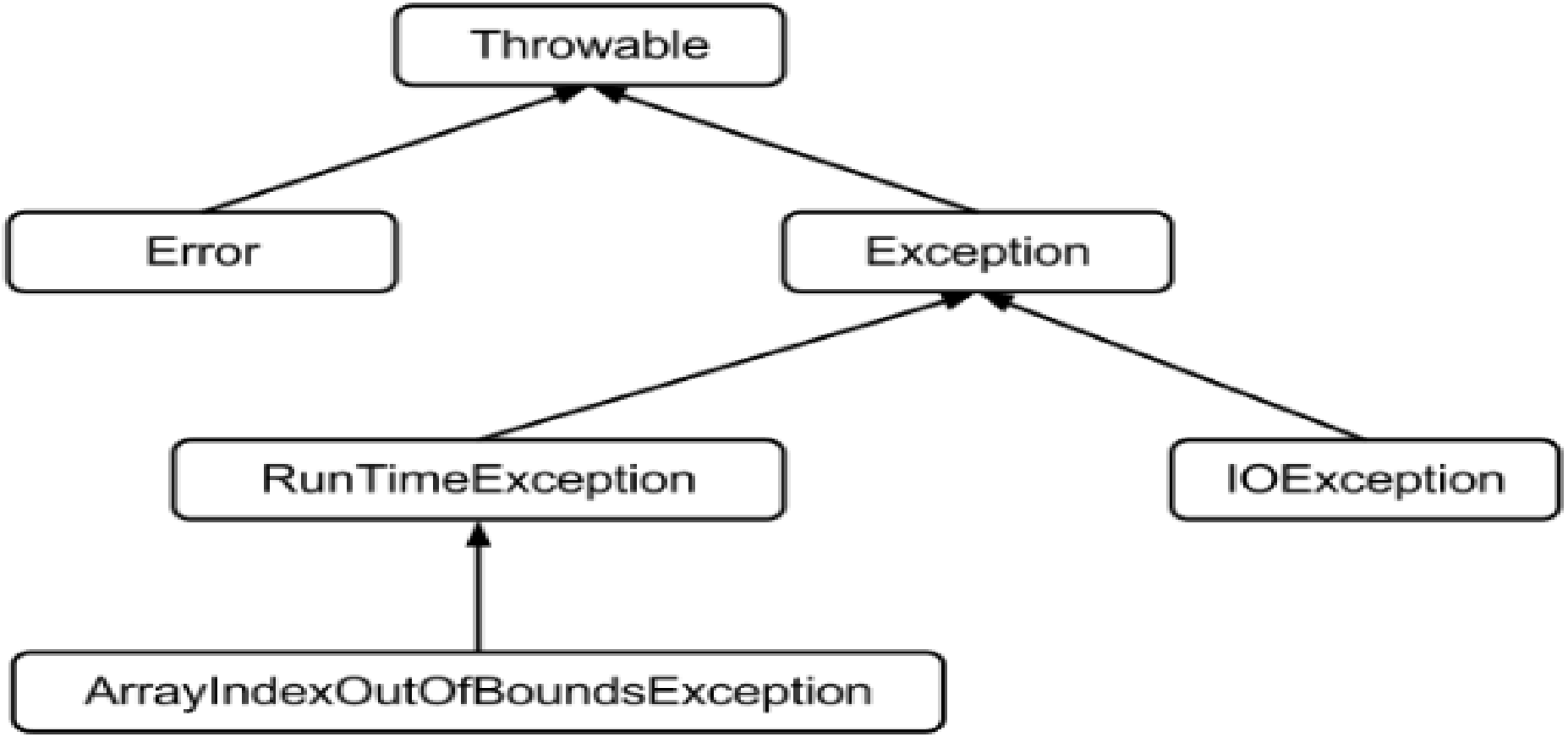
Las excepciones, en java durante la ejecución de programas, son instancias de clases concretas

Java informa que algo ha ido mal, porque recibimos un objeto de una clase que representa un determinado error.

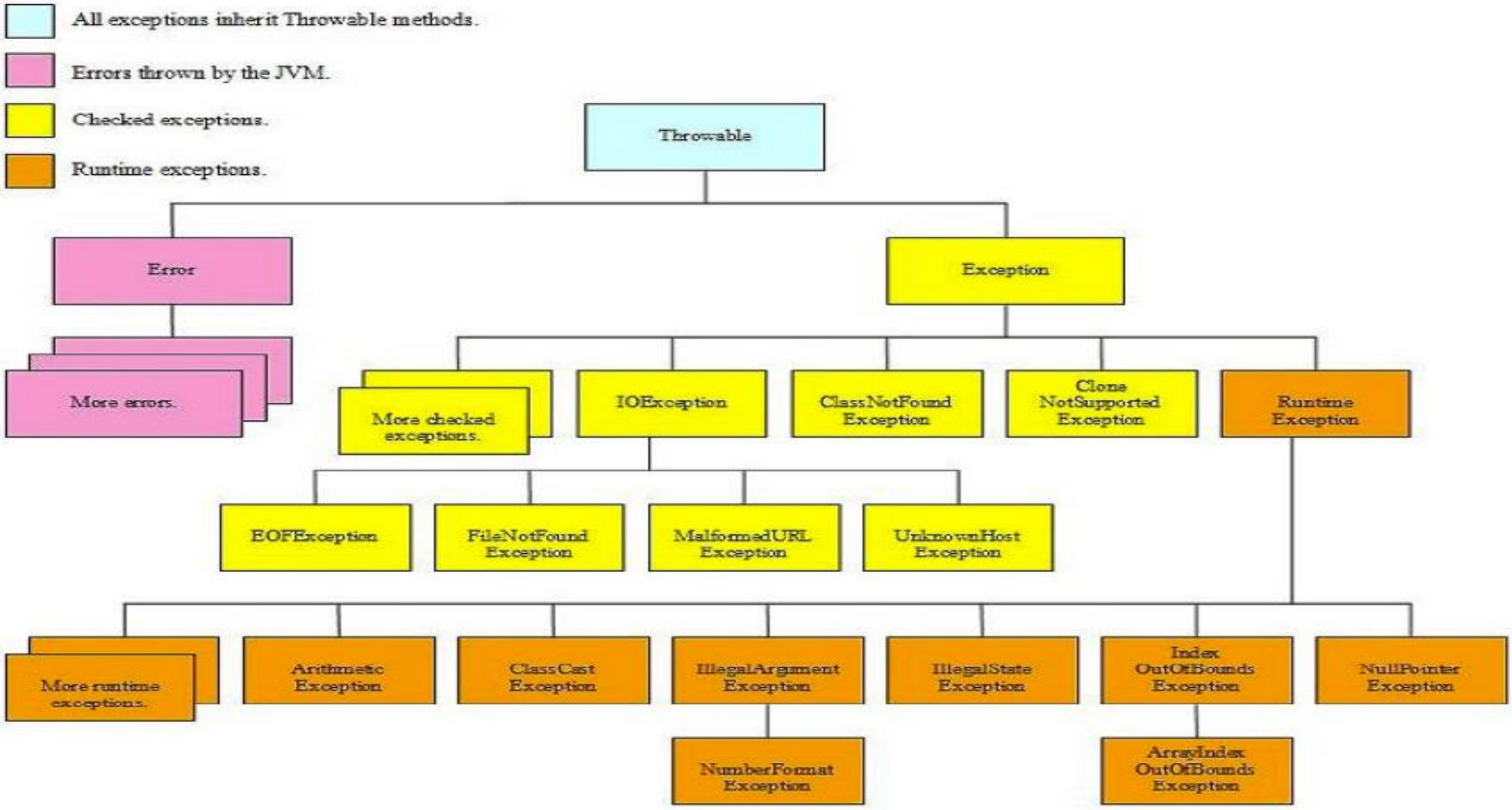
Para los ejemplos anteriores recibimos una instancia de la clase **ArrayIndexOutOfBoundsExcepcion**, y **FileNotFoundException** respectivamente.

Tipos de excepciones

- Existe una jerarquía de clases que representan errores en Java.



Exception Hierarchy [2]



Tipos de excepciones

Las excepciones se clasifican en:

1. **Excepciones marcadas**.
   Aquellas cuya captura es obligatoria.

2. **Excepciones no marcadas.**
   Las excepciones en tiempo de ejecución (RuntimeException y sus subclases). No es obligatorio capturarlas.

2. Implementación de excepciones

- El formato para escribir un bloque en el que se gestionan excepciones es:

```
try {
        instrucciones
} catch {
        instrucciones
} finally {
        instrucciones
}
```

El bloque **try** es utilizado para definir el bloque de código en el cual una excepción pueda ocurrir.

El o los bloques **catch** son utilizados para definir un bloque de código que maneje la excepción.

El bloque **finally** es opcional.

2. Implementación de excepciones

Ejemplo de la gestión de una excepción en el uso de un arreglo estático

```
public class Prueba2 {

    public static void main(String[] args) {
        int[] enteros = { 1, 2, 3, 4, 5 };
        int i;
        try{
            // error se está accediendo a una posición fuera del arreglo.
            for (i = 0; i <= 5; i++)
                System.out.println("valor de enteros[" + i + "]: " + enteros[i]);
        }catch (IndexOutOfBoundsException e) {
            System.out.println("Error: intenta acceder a un elemento fuera del arreglo");
            //System.out.println("Error: "+e.getMessage());
            //e.printStackTrace();
        }
    }
}
```

2. Implementación de excepciones: Realice una corrida

```
public class PruebaExcepciones {

    public static void main(String[] args) {
        // Primer try
        try {
            System.out.println("Entrando al primer try");
            double cociente=1000/0; //se lanza la excepción
            System.out.println("Después de la división"+cociente);
        }catch(ArithmeticException e) {
            System.out.println("División por cero");
        }finally {
            System.out.println("Entrado al primer finally");
        }
```

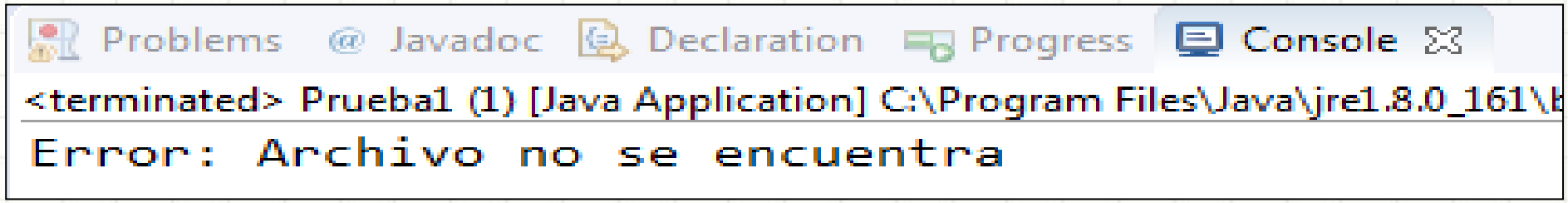
2. Implementación de excepciones: Realice una corrida

```
        //Segundo try
        try {
            System.out.println("Entrando al segundo try");
            Object objeto=null;
            objeto.toString();//se lanza la excepción
            System.out.println("Imprimiendo objeto");
        }catch (ArithmeticException e) {
            System.out.println("División por cero");
        }catch (NullPointerException e) {
            System.out.println("objeto null");
        }catch (Exception e) {
            System.out.println("ocurrió una excepción");
        }finally {
            System.out.println("Entrando al segundo finally");
        }
    }

}
```

2. Implementación de excepciones

- Gestionando la excepción al intentar abrir un archivo inexistente

```
public class Prueba1 {
    public static void main(String[] args) {

        try {
            FileReader archivo = new FileReader("este archivo no existe");
        } catch (FileNotFoundException e) {
            System.out.println("Error: Archivo no se encuentra");
            //System.out.println("Error: "+e.getMessage());
            //e.printStackTrace();
        }
    }
}
```



2. Implementación de excepciones

- Una buena práctica es cerrar los archivos después de trabajar con ellos.

```
 7  public class Prueba2 {
 8
 9      public static void main(String[] args) {
10          FileReader archivo = null;
11          try{
12              archivo = new FileReader("nombre del archivo");
13          }catch (FileNotFoundException e) {
14              e.printStackTrace();
15          }finally{
16              if (archivo != null)
17                  archivo.close();
18
19          }
20      }
21  }
22
23
```

Unhandled exception type IOException

2 quick fixes available:

Add throws declaration

Surround with try/catch

Press 'F2' for focus

El método puede producir un error de tipo IOException, y requerir otro bloque try-catch

2. Implementación de excepciones

```
public class Prueba2 {

    public static void main(String[] args) {
        FileReader archivo = null;
        try {
            archivo = new FileReader("nombre del archivo");
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } finally {
            try {
                System.out.println("finally siempre se ejecuta");
                if (archivo != null)
                    archivo.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

- Se puede escribir de la siguiente manera.

2. Implementación de excepciones

En caso de **no querer** gestionar la **excepción** en el lugar donde se produce, se usa la palabra reservada **throws**, indicando la excepción que no se gestiona.

```
public class Prueba4 {

    public static void main(String[] args) throws IOException {
        FileReader archivo = null;
        try {
            try {
                archivo = new FileReader("prueba1.txt");
            } finally {
                System.out.println("Siempre se ejecuta el finally");
                archivo.close();
            }
        } catch (FileNotFoundException e) {
            System.out.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }

}
```

Si no existe el archivo tendremos como salida lo siguiente.

```
Console
<terminated> Prueba4 [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (04/09
Siempre se ejecuta el finally
Exception in thread "main" java.lang.NullPointerException
        at cursoPOO.presentacion.Prueba4.main(Prueba4.java:16)
```

2. Implementación de excepciones

Desde Java 7, se pueden colocar varias opciones dentro del bloque catch, y si existe **relación padre-hija** entre ellas, basta con **indicar el padre.**

Revisar que se quitó throws IOException, para gestionarla

```
public class Prueba {

    public static void main(String[] args) {
        FileReader archivo = null;

        try{
            try{
                archivo = new FileReader("prueba1.txt");
            }finally{
                if(archivo!=null)
                    archivo.close();
            }
        }catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

Recomendable verificar que archivo !=null antes de cerrar.

Se quitó FileNotFoundException.

2. Implementación de excepciones

Todo método tiene como salida una alternativa, el **normal** si no genera ninguna excepción durante la ejecución, **o devolver una excepción** en el punto donde se produzca, si es que no se gestiona dentro del método.

```
int metodo() throws IOException{

…

}
```

Devolverá un entero o una excepción.

2. Implementación de excepciones

Pasos para crear sus propias excepciones

1. Definir una clase que representa la excepción
2. Lanzarla en las situaciones de error.
3. Gestionarla en la clase cliente como cualquier otra excepción.

2. Implementación de excepciones

Clase que define la excepción

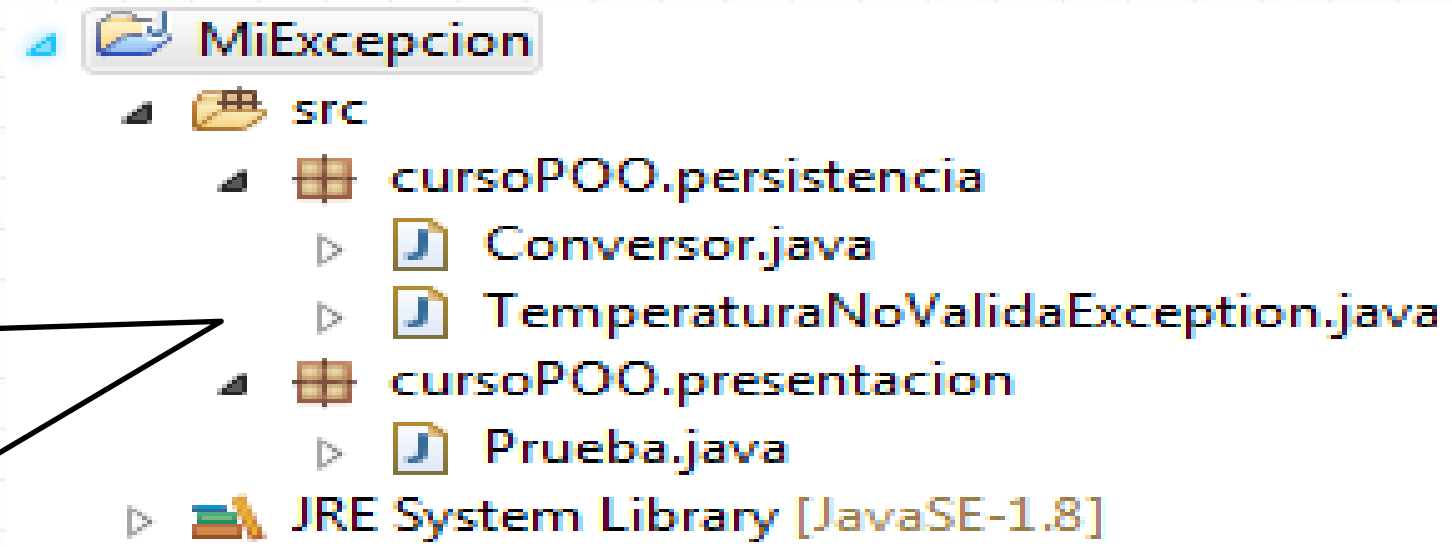
```
Public class MiExcepcion extends Exception{
      //constructor
      public MiExcepcion(){
          super("Texto de la excepción");
      }
}
```

El texto de la excepción se puede recuperar con el método **getMessage**() definido en la clase **Throwable**.

2. Implementación de excepciones

Creación de una clase excepción que realiza conversiones de temperaturas de grados Celsius a grados Fahrenheit.

Se **define** la clase excepción de nombre **TemperaturaNoValidaExcepcion**; **se lanza** la excepción en la clase **Conversor** y por último se **gestiona** la excepción en la clase **Prueba**.



```java
package cursoPOO.persistencia;

/**
 * La clase TemperaturaNoValidaException define la excepción.
 * En ella se coloca el mensaje que se desea mostrar en
 * caso de lanzarse la excepción.
 *
 * @author David Martínez Torres
 **/
public class TemperaturaNoValidaException extends Exception {

    /**
     *
     */
    private static final long serialVersionUID = 1L;

    public TemperaturaNoValidaException(){
        super("La temperatura no puede ser inferior a -57º Celsius");
    }

    public TemperaturaNoValidaException(double temperaturaActual){
        super("La temperatura no puede ser inferior a -57º Celsius " +
                "la actual es " + temperaturaActual);
    }

}
```

Paso1. Clase que define la excepción

```java
package cursoPOO.persistencia;

/**
 * La clase Conversor convierte de ºCelsius a ºFahrenheit
 * Formula: ºF=ºC*9/5)+32
 * En esta clase se lanza la excepción si se cumple la
 * condición.
 *
 * @author David Martínez Torres
 */
public class Conversor {
    public double celsiusAFharenheit(float celsius)
            throws TemperaturaNoValidaException{
        if(celsius<-57)
            throw new TemperaturaNoValidaException(celsius);
        return (celsius*9.0/5.0)+32;
    }

}
```

Paso2. Clase que lanza la excepción

```java
package cursoPOO.presentacion;

import cursoPOO.persistencia.Conversor;

/**
 * Clase Prueba que gestiona la excepción con el try-catch
 * @author David Martínez Torres
 *
 */
public class Prueba {

    public static void main(String[] args) {
        Conversor conversor = new Conversor();
        double resultado;
        try{
            resultado=conversor.celsiusAFharenheit(-300);
            System.out.println("Resultado= " + resultado + "ºF");
        }catch (TemperaturaNoValidaException e) {
            e.printStackTrace();
        }

    }

}
```

Paso3. Clase que gestiona la excepción

Markers | Properties | Servers | Console | Problems | Progress | Search | JUnit | Maven

<terminated> Prueba (11) [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (24/01/2019 09:21:15)

```
cursoPOO.persistencia.TemperaturaNoValidaException: La temperatura no puede ser i
        at cursoPOO.persistencia.Conversor.celsiusAFharenheit(Conversor.java:15)
        at cursoPOO.presentacion.Prueba.main(Prueba.java:17)
```

2. Implementación de excepciones

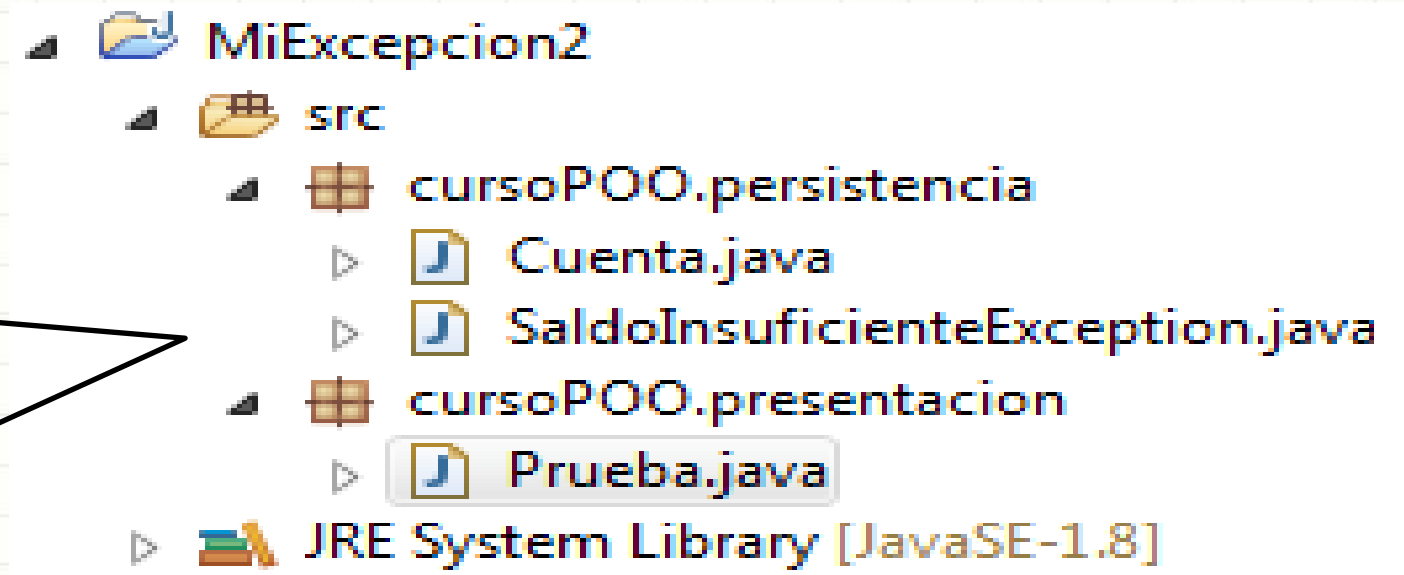
Es bueno saber **crear** sus propias excepciones, sin embargo, es mejor **buscar** alguna excepción ya predefinida y pasarle un mensaje descriptivo de lo que ha ido mal. En este caso se puede utilizar la clase **IllegalArgumentException** y pasar el mensaje.

```
package cursoPOO.persistencia;

public class ConversorUsaExcepcionJava {
    public double celsiusAFharenheit(float celsius)
            throws IllegalArgumentException{
        if(celsius<-57)
            throw new IllegalArgumentException("La "
                    + "temperatura no puede ser inferior"
                    + " a -57º Celsius " +
                    "la actual es " + celsius);
        return (celsius*9.0/5.0)+32;
    }
}
```

Ejemplo2 de la creación de una clase Excepción

Creación de una clase excepción que valida la realización de retiros de dinero.

Se **define** la clase excepción de nombre **SaldoInsuficienteException**; **se lanza** la excepción en la clase **Cuenta** y se **gestiona** la excepción en la clase Prueba.



Ejemplo2 de la creación de una clase Excepción

Paso1. Clase que define la excepción

```
public class SaldoInsuficienteException extends Exception {

    /**
     *
     */
    private static final long serialVersionUID = 4704750190795465735L;

    public SaldoInsuficienteException(){
        super("Saldo insuficiente");
    }

}
```

Ejemplo2 de la creación de una clase Excepción

```
public class Cuenta {
    private double saldo;

    public Cuenta(){
        saldo=0;
    }

    public void depositar(double monto){
        System.out.println("Depositando: "+monto);
        saldo+=monto;
    }

    public void retirar(double monto) throws SaldoInsuficienteException{
        System.out.println("Retirando: "+monto);
        if(saldo<monto)
            throw new SaldoInsuficienteException();
        else
            saldo -=monto;
    }

    public double getSaldo() {
        return saldo;
    }


}
```

Paso2. Clase Cuenta, lanza la excepción

Ejemplo2 de la creación de una clase Excepción

```
public class Prueba {

    public static void main(String[] args) {
        Cuenta cuenta = new Cuenta();
        try{
            cuenta.depositar(1000);
            cuenta.retirar(500);
            System.out.println("Consulta saldo: "+cuenta.getSaldo());
            cuenta.retirar(500);
            System.out.println("Consulta saldo: "+cuenta.getSaldo());
            cuenta.retirar(500);
            System.out.println("Consulta saldo: "+cuenta.getSaldo());
        }catch (SaldoInsuficienteExcepction e) {
            System.out.println("Error: "+e.getMessage());
        }
    }
}
```

Paso3. Clase que gestiona la excepción

Salida. Puede probar con e.printStackTrace();

Problems | Javadoc | Declaration | Progress | Console

<terminated> Prueba (11) [Java Application] C:\Program Files\Java\jre1.8.0_1

```
Depositando: 1000.0
Retirando: 500.0
Consulta saldo: 500.0
Retirando: 500.0
Consulta saldo: 0.0
Retirando: 500.0
Error: Saldo insuficiente
```

Referencias

1. Joyanes, Aguilar Luis. Programación Orientada a Objetos. 2ª edición. España, McGraw-Hill.
2. Angélica Nakayama C., Jorge A. Solano Gálvez. Guía práctica de estudio 10: Excepciones y errores. UNAM.
3. Rivera, López Rafael. Programación Orientada a Objetos con Java. Instituto Tecnológico de Veracruz.
4. Kathy Sierra & Bert Bates. Head First Java. O'Reilly Media. 2005
5. Kölling y Barnes. Programación orientada a objetos con Java usando Bluej. Madrid, España: Pearson Universidad. 2013