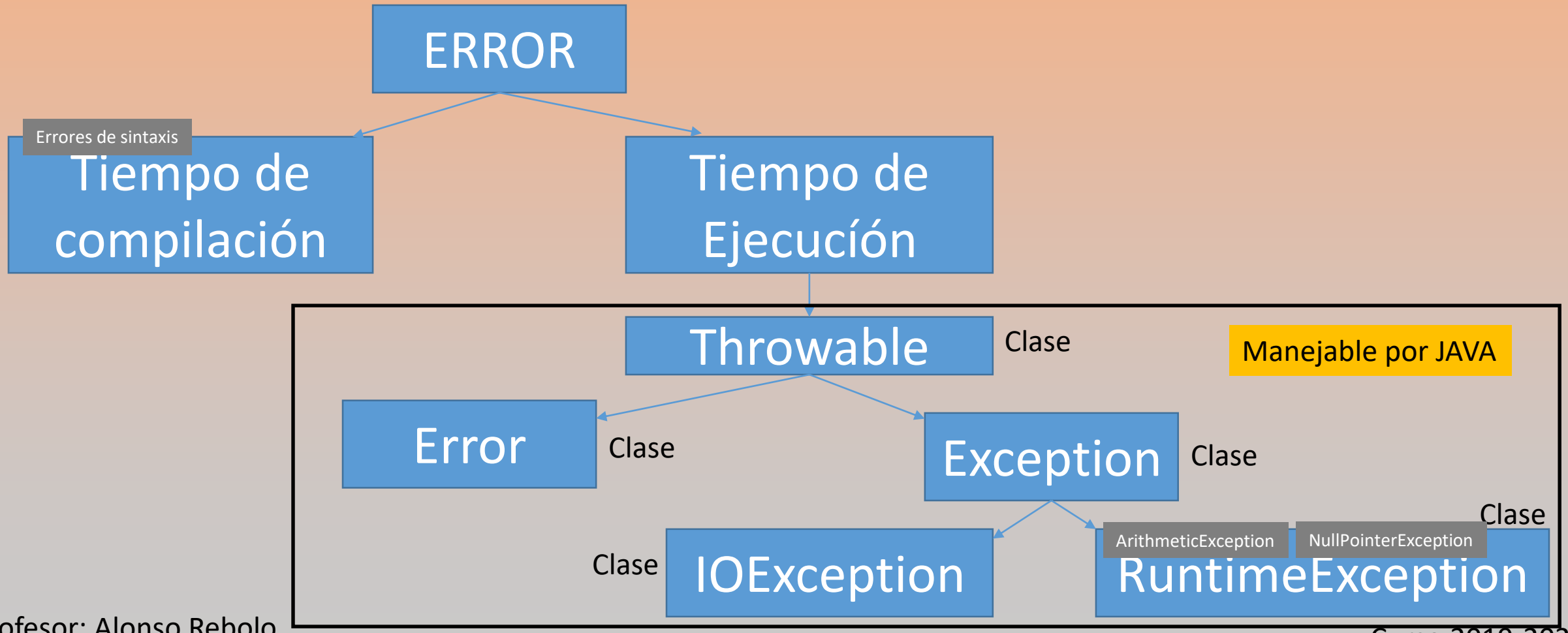
TRATAMIENTO DE ERRORES EN JAVA

Profesor: Alonso Rebolo

Curso 2019-2020

¿Qué son los errores en Java?

Las **excepciones** en **Java** no son más que clases especializadas que, como su nombre indica, representan excepciones, errores o fallos que se producen en las instrucciones de nuestros programas pero, al contrario que los errores de sintaxis, estas fallas se producen en **tiempo de ejecución.**



¿Qué pasa cuando se produce una excepción?

Cuando en Java se produce una excepción se instancia un objeto de una determina clase (dependiendo del tipo de error que se haya producido). El objeto creado mantendrá la información sobre el error producido y nos proporcionará los métodos necesarios para obtener dicha información.

Errores en tiempos de compilación

Son errores debido a fallos en la sintaxis. Una vez detectados, el programador debe modificar el código para solventar el error.

Estos errores no se pueden controlar mediante sentencias de código.

Throwable

Es la superclase de la que derivan todas las clases que tratan los errores y las excepciones

Error

Estos errores no se pueden controlar por código de programación por lo que El programador no puede hacer nada para evitarlos.

Ejemplos:

- **Fallo del Disco Duro**
- **Fallo en la RAM**
- **Fallo del Sistema Operativo**
- **Fallo en la Máquina Virtual Java (JVM)**

Exception

Estos errores si se pueden manejar con la intervención del programador

IOException: Se les denomina excepciones comprobadas. Estos errores no son producidos por fallo del programador y deben ser controlados por medio de código try-catch. (Ejemplo: Un fichero o una imagen que no está en el lugar indicado)

RuntimeException: Se les denomina excepciones No comprobadas. Estos errores son ocasionados por fallo en el código. (Ejemplo: intentar acceder a un índice de una array que no existe).

Se pueden controlar por código try-catch pero lo más correcto es modificar el código para solventar el error.

Sintaxis General de try-catch-finally

```
try
{
    código donde se puede dar el error

}
catch (tipo de excepción e)
{
    código que se ejecuta cuando sucede el error

}
finally
{
        Código que se ejecuta siempre, se haya producido o no el error
}
```

Sintaxis try-catch a partir de Java 7

```
try(Stream<String> flujo = Files.lines(camino, Charset.forName("UTF-8")))
{
        flujo.forEach((String s) -> System.out.println(s));
}
catch (IOException e)
{
        e.printStackTrace();
}
```

A este tipo de sintaxis se le llama:

Try-with-resources

The try-with-resources statement is a try statement that declares one or more resources.
A resource is an object that must be closed after the program is finished with it.
The try-with-resources statement ensures that each resource is closed at the end
of the statement. Any object that implements java.lang.AutoCloseable,
which includes all objects which implement java.io.Closeable, can be used as a resource.

try-with-resources

```
static String readFirstLineFromFile(String path) throws IOException
{
    try (BufferedReader br = new BufferedReader(new FileReader(path)))
    {
      return br.readLine();
    }
 }
```

In this example, the resource declared in the try-with-resources statement is a BufferedReader. The declaration statement appears within parentheses immediately after the try keyword. The class BufferedReader, in Java SE 7 and later, implements the interface java.lang.AutoCloseable. Because the BufferedReader instance is declared in a try-with-resource statement, it will be closed regardless of whether the try statement completes normally or abruptly (as a result of the method BufferedReader.readLine throwing an IOException).

Prior to Java SE 7, you can use a finally block to ensure that a resource is closed regardless of whether the try statement completes normally or abruptly.

Anidamientos de try

Se pueden anidar los try y cada uno irá con su(s) catch(s)

```
try
{
    for(int i=0;i<array.length;i++)
    {
        try
        {

        }
        catch (ArtithmeticException e)
        {
          e.printStackTrace():
        }
    }
}
catch(ArrayIndexOutOfBoundsException e)
{

}
```

Múltiples catch

Por cada try puede haber uno o más catch. Se pondrán de más específico a más general.

```
try
{

}
catch(ArrayIndexOutOfBoundsException e)
{

}
catch(...)
{

}
catch(Throwable t)
{

}
```

Cláusula throws

Si en el cuerpo de un método se lanza una excepción (de un tipo derivado de la clase IOException), en la cabecera del método hay que añadir una cláusula throws que incluye una lista de los tipos de excepciones que se pueden producir al invocar el método.

Ejemplo:
public String leerFichero (String nombreFichero) throws IOException

Las excepciones de tipo RuntimeException no son necesarias declararlas en la cláusula throws.

Al implementar un método, hay que decidir si las excepciones se propagarán hacia arriba (throws) o se capturarán en el propio método (catch)

Captura del error hacia arriba: throws

Un método que propaga una excepción:

```
public void f() throws IOException
{
        // Fragmento de código que puede
        // lanzar una excepción de tipo IOException
}
```

Un método puede lanzar una excepción al crear, explícitamente, un objeto Throwable y lo lance con throw, o bien porque llame a un método que genere la excepción y no la capture.

Captura del error en el método

Un método equivalente que no propaga la excepción:

```
public void f()
{
        // Fragmento de código libre de excepciones
        try
        {
            // Fragmento de código que puede
            // lanzar una excepción de tipo IOException // (p.ej. Acceso a un fichero)
         }
        catch (IOException error)
        {
            // Tratamiento de la excepción
        }
        finally
        {
            // Liberar recursos (siempre se hace)
        }
 }
```

Cláusula throw

Se utiliza en Java para lanzar objetos de tipo Throwable.

throw new Exception(“Mensaje de error…”);

Cuando se lanza una excepción se sale inmediatamente del bloque de código actual.

Si el bloque tiene asociada una cláusula catch adecuada para el tipo de la excepción generada, se ejecuta el cuerpo de la cláusula catch.

Si no, se sale inmediatamente del bloque (o método) dentro del cual está el bloque en el que se produjo la excepción y se busca una cláusula catch apropiada.

El proceso continúa hasta llegar al método main de la aplicación. Si ahí tampoco existe una cláusula catch adecuada, la máquina virtual Java finaliza su ejecución con un mensaje de error.

Ejemplos cláusula throw

```
Try
{
        origen = database.find(IDorigen);
        if (origen == null)
            throw new Exception("No existe ” + IDorigen);
            origen.setBalance (origen.getBalance() – cantidad);
            database.store(origen); destino = database.find(IDdestino);
            if (destino == null)
                throw new Exception("No existe ” + IDdestino);

            ...
}
```

Creación de una excepción propia

Hay que crear una clase que extienda a Exception o a algunas de sus subclases.

```
public DivideByZeroException extends ArithmeticException
{
	public DivideByZeroException(String Message)
	{
		super(message);
	}
 }
```

Lanzar el error:

```
public double dividir(int num, int den) throws DivideByZeroException
{
	 if (den==0) throw new DivideByZeroException(“Error!”);
	return ((double) num/(double)den);
}
```

Ejemplo de proceso de excepciones

```
public class ProgramaPrincipal
{
      public static void main(String[] args)
     {
        String str1="120“, str2="3“, String respuesta;
        int numerador, denominador, cociente;
        try
        {
           numerador=Integer.parseInt(str1);
           denominador=Integer.parseInt(str2);
           rango(numerador, denominador);
           cociente=numerador/denominador;
           respuesta=String.valueOf(cociente);
        }
        catch(NumberFormatException ex)
        {
           respuesta="Se han introducido caracteres no numéricos";
        }
        catch(ArithmeticException ex)
        {
           respuesta="División entre cero";
        }
        catch(ExcepcionIntervalo ex)
        {
           respuesta=ex.getMessage();
        }
        System.out.println(respuesta);
}
```

Nos creamos nuestra excepción personificada

```
public class ExcepcionIntervalo extends Exception
{
     public ExcepcionIntervalo(String msg)
     {
        super(msg);
     }
 }
```

Nos creamos un método

```
static void rango(int num, int den) throws ExcepcionIntervalo
{
     if((num>100)||(den<-5))
     {
           throw new ExcepcionIntervalo("Números fuera de rango");
     }
}
```

Ciclo de vida de una excepción

1.- Se coloca la llamada a la función susceptible de producir una excepción en un bloque try...catch

2.- En dicha función se especifica que puede producirse una Exception mediante la cláusula throws y el nombre de la Exception

3.- Se lanza mediante throw el objeto, creado previamente o en el mismo momento del throw mediante new

4.- Se captura en el correspondiente bloque catch

5.- En este bloque se notifica al usuario esta eventualidad imprimiendo el mensaje asociado a dicha excepción, o realizando una tarea específica.

Función que puede conllevar más de un tipo de Exception

```
public class ProgramaExcepcion
{
    public static void main(String[] args)
    {
        String str1="20“, str2="2“, respuesta;
        int numerador, denominador, cociente;
        try
        {
            cociente=calcular(str1, str2);
            respuesta=String.valueOf(cociente);
        }
        catch(NumberFormatException ex)
        {
            respuesta="Se han introducido caracteres no numéricos";
        }
        catch(ArithmeticException ex)
        {
            respuesta="División entre cero";
        }
        catch(ExcepcionIntervalo ex)
        {
            respuesta=ex.getMessage();
        }
        System.out.println(respuesta);
    }

    static int calcular(String str1, String str2) throws ExcepcionIntervalo, NumberFormatException, ArithmeticException
    {
        int num=Integer.parseInt(str1);
        int den=Integer.parseInt(str2);
        if((num>100)||(den<-5))
        {
            throw new ExcepcionIntervalo("Números fuera del intervalo");
        }
        return (num/den);
    }
}
```