
Clase 17: Excepciones

— Programación & Laboratorio III —

Agenda

- **Concepto de excepción**
- Tipos de excepciones
- Excepciones checked o comprobadas
- Excepciones unchecked o no comprobadas
- Lanzamiento de excepciones
- Transferencia de control
- Cláusula throws
- Creando nuestras propias excepciones
- Malas prácticas
- Buenas prácticas



Concepto de excepción

- Una excepción en Java es un error o situación “excepcional” que se produce durante la ejecución de un programa.
- Ejemplos:
 - Leer un archivo que no existe
 - Acceder al valor N de una colección que tiene menos de N elementos.
 - Enviar o recibir información por red mientras se produce una pérdida de conectividad.
- Todas las excepciones en Java se representan a través de objetos que heredan de `java.lang.Throwable`.

Concepto de excepción (2)

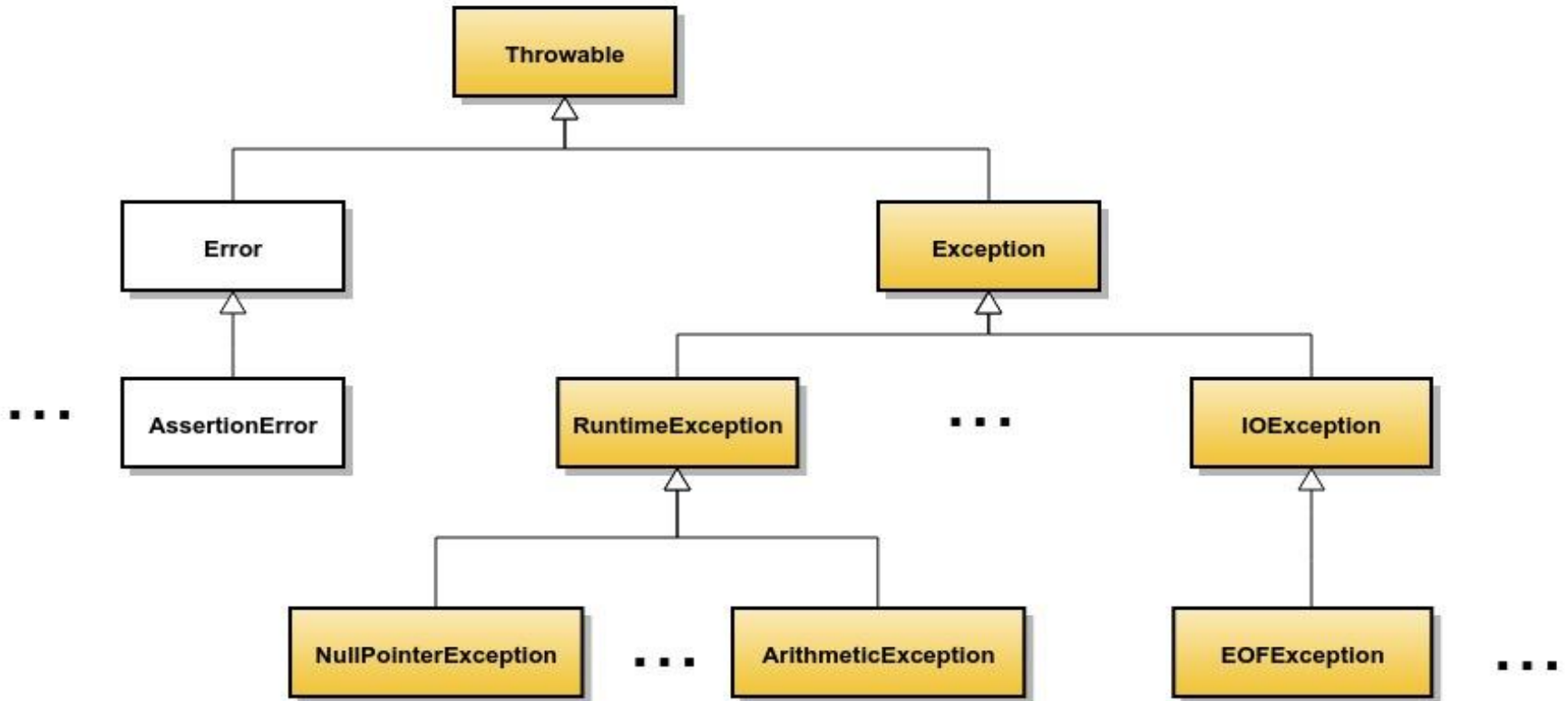
- Las excepciones proporcionan una forma clara de comprobar posibles errores sin oscurecer el código.
- Convierten las condiciones de error que un método puede señalar en una parte explícita del contrato del método.
- La lista de excepciones pueden ser vistas por el programador, comprobada por el compilador y preservada por las clases extendidas que redefinen el método.

Agenda

- ~~Concepto de excepción~~
- **Tipos de excepciones**
- Excepciones checked o comprobadas
- Excepciones unchecked o no comprobadas
- Lanzamiento de excepciones
- Transferencia de control
- Cláusula throws
- Creando nuestras propias excepciones
- Malas prácticas
- Buenas prácticas



Tipos de excepciones



Tipos de excepciones (2)

- Las excepciones también son objetos.
- Todos los tipos de excepción deben extender de la clase `Throwable` o de alguna de sus subclases.
- De `Throwable` nacen dos ramas: `Error` y `Exception`.
 - 1) `Error` representa errores de una magnitud tal que una aplicación nunca debería intentar realizar nada con ellos. Por ejemplo: errores de la JVM, desbordamiento de buffer.
 - 2) `Exception` representa aquellos errores que normalmente sí solemos gestionar.

Tipos de excepciones (3)

- Por convenio, los nuevos tipos de excepción extienden a `Exception`.
- De `Exception` nacen múltiples ramas: `ClassNotFoundException`, `IOException`, `ParseException`, `SQLException` → **Excepciones checked.**
- `RuntimeException` es la única **excepción unchecked.**

Agenda

- ~~— Concepto de excepción~~
- ~~— Tipos de excepciones~~
- **Excepciones checked o comprobadas**
- Excepciones unchecked o no comprobadas
- Lanzamiento de excepciones
- Transferencia de control
- Cláusula throws
- Creando nuestras propias excepciones
- Malas prácticas
- Buenas prácticas



Excepciones checked (comprobadas)

- Una excepción de tipo checked representa un error del cual técnicamente podemos recuperarnos.
- Son todas las situaciones que son totalmente ajenas al propio código, por ejemplo: fallo de una operación de lectura/escritura.
- Este tipo de excepciones deben ser **capturadas** o **relanzadas**.

Agenda


- ~~— Concepto de excepción~~
- ~~— Tipos de excepciones~~
- ~~— Excepciones checked o comprobadas~~
- **Excepciones unchecked o no comprobadas**
- Lanzamiento de excepciones
- Transferencia de control
- Cláusula throws
- Creando nuestras propias excepciones
- Malas prácticas
- Buenas prácticas



Excepciones unchecked (no comprobadas)

- Representan errores de programación. Por ejemplo: intentar leer de un array de N elementos un elemento que se encuentra en una posición mayor que N.

```
int[] numerosPrimos = {1, 3, 5, 7, 9, 11, 13, 17, 19, 23};  
System.out.println(numerosPrimos[10]);
```

 `ArrayIndexOutOfBoundsException`

Agenda

- ~~Concepto de excepción~~
- ~~Tipos de excepciones~~
- ~~Excepciones checked o comprobadas~~
- ~~Excepciones unchecked o no comprobadas~~
- **Lanzamiento de excepciones**
- Transferencia de control
- Cláusula throws
- Creando nuestras propias excepciones
- Malas prácticas
- Buenas prácticas



Lanzamiento de excepciones

- Las excepciones se lanzan utilizando la palabra reservada throw:

```
throw AException
```

- AException debe ser un objeto Throwable.
- Si se lanza la excepción no se regresa al flujo normal del programa.
- Las excepciones son objetos, por lo tanto se debe crear una instancia antes de lanzarse.

Lanzamiento de excepciones (2)

- Ejemplo:

```
public void buscarNumeroPrimo(int n) {  
    for (int i=0; i<10; i++) {  
        if (numerosPrimos[i] == n) {  
            System.out.println("El número " + n + " existe.");  
            break;  
        }  
    }  
    throw new NumeroNoExisteException(n);  
}
```

Agenda

- ~~Concepto de excepción~~
- ~~Tipos de excepciones~~
- ~~Excepciones checked o comprobadas~~
- ~~Excepciones unchecked o no comprobadas~~
- ~~Lanzamiento de excepciones~~
- **Transferencia de control**
- Cláusula throws
- Creando nuestras propias excepciones
- Malas prácticas
- Buenas prácticas



Transferencia de control

- Una vez que se produce una excepción, las acciones que hubiera detrás del punto donde se produjo la excepción no tienen lugar.
- Las acciones que sí se ejecutarán serán las contenidas dentro de los bloques `catch` y `finally`.

Transferencia de control (2)

- Ejemplo:

```
public static void main(String[] args) {  
    FileWriter fichero = null;  
    try {  
        fichero = new FileWriter(PATH_ARCHIVO);  
        fichero.write("Escribiendo...")  
    } catch (IOException e) {  
        e.printStackTrace();  
    } finally {  
        if (fichero != null) {  
            fichero.close();  
        }  
    }  
}
```

Transferencia de control (3)

- Las sentencias dentro de la cláusula `try` se ejecutan hasta que se lance una excepción o hasta que finalice con éxito.
- Si se lanza una excepción, se examinan sucesivamente las sentencias de la cláusula `catch`.
- La cláusula `finally` de una sentencia proporciona un mecanismo para ejecutar una sección de código, se lance o no una excepción.
- Generalmente la cláusula `finally` se utiliza para limpiar el estado interno o para liberar recursos, como archivos abiertos o cerrar conexiones a bases de datos.

Agenda

- ~~Concepto de excepción~~
- ~~Tipos de excepciones~~
- ~~Excepciones checked o comprobadas~~
- ~~Excepciones unchecked o no comprobadas~~
- ~~Lanzamiento de excepciones~~
- ~~Transferencia de control~~
- **Cláusula throws**
- Creando nuestras propias excepciones
- Malas prácticas
- Buenas prácticas



Cláusula throws

- Se utiliza para declarar las excepciones checked que puede lanzar un método.
- Se pueden declarar varias, separadas por comas.
- RuntimeException y Error son las únicas excepciones que no hace falta incluir en las cláusulas throws.

Cláusula throws (2)

- Si se invoca a un método que tiene una excepción checked en su cláusula throws, existen tres opciones:
 - 1) Capturar la excepción y gestionarla.

```
public void abrirArchivo(String path) throws IOException {  
    ...  
}  
  
public void leerArchivo(String path) {  
    try {  
        abrirArchivo(path);  
    } catch(IOException e) {  
        System.out.println("Se produjo un error al abrir el archivo").  
    }  
}
```

Cláusula throws (3)

- 2) Capturar la excepción y transformarla en una de nuestras excepciones.

```
public void abrirArchivo(String path) throws IOException {  
    ...  
}  
  
public void leerArchivo(String path) {  
    try {  
        abrirArchivo(path);  
    } catch(IOException e) {  
        throw new ArchivoNoExisteException(e);  
    }  
}
```

Cláusula throws (4)

3) Declarar la excepción en la cláusula throws y hacer que otro método la gestione.

```
public void abrirArchivo(String path) throws IOException {  
    ...  
}
```

```
public void leerArchivo(String path) throws IOException {  
    abrirArchivo(path);  
    ...  
}
```


Cláusula throws (5)

- No se permite que los métodos redefinidos declaren más excepciones checked en la cláusula throws que las que declara el método.
- Se pueden lanzar subtipos de las excepciones declaradas ya que podrán ser capturadas en el bloque catch correspondiente a su supertipo.
- Si una declaración de un método se hereda en forma múltiple, la cláusula throws de ese método

Agenda

- ~~Concepto de excepción~~
- ~~Tipos de excepciones~~
- ~~Excepciones checked o comprobadas~~
- ~~Excepciones unchecked o no comprobadas~~
- ~~Lanzamiento de excepciones~~
- ~~Transferencia de control~~
- ~~Cláusula throws~~
- **Creando nuestras propias excepciones**
- Malas prácticas
- Buenas prácticas



Creando nuestras excepciones

```
public class SaldoInsuficienteException extends Exception {  
    ...  
}  
  
public static void main(String[] args) {  
    Cliente cliente = new Cliente();  
    try {  
        cliente.pagar();  
    } catch(SaldoInsuficienteException e) {  
        e.printStackTrace();  
    }  
}
```

Agenda

- ~~Concepto de excepción~~
- ~~Tipos de excepciones~~
- ~~Excepciones checked o comprobadas~~
- ~~Excepciones unchecked o no comprobadas~~
- ~~Lanzamiento de excepciones~~
- ~~Transferencia de control~~
- ~~Cláusula throws~~
- ~~Creando nuestras propias excepciones~~
- **Malas prácticas**
- Buenas prácticas



Malas prácticas

1

```
FileWriter fichero = null;  
try {  
    fichero = new FileWriter("path");  
    fichero.write("Writing...");  
} catch (IOException e) {  
  
}
```

2

```
int[] numerosPrimos = {1, 3, 5, 7, 9, 11, 13, 17, 19, 23};  
int undecimoPrimo = numerosPrimos[10];  
try {  
    int i = 0;  
    while(true) {  
        System.out.println(numerosPrimos[i++]);  
    }  
} catch (ArrayIndexOutOfBoundsException e) {}
```

Malas prácticas

3

```
public void pagar() throws Exception{  
    //Do something  
}
```

4

```
public void pagar() {  
    try {  
  
    } catch (Exception e) {  
        throw new RuntimeException();  
    }  
}
```

Agenda

- ~~Concepto de excepción~~
- ~~Tipos de excepciones~~
- ~~Excepciones checked o comprobadas~~
- ~~Excepciones unchecked o no comprobadas~~
- ~~Lanzamiento de excepciones~~
- ~~Transferencia de control~~
- ~~Cláusula throws~~
- ~~Creando nuestras propias excepciones~~
- ~~Malas prácticas~~
- **Buenas prácticas**



Buenas prácticas

- Utilizar excepciones que ya existen.
- Lanzar excepciones de acuerdo al nivel de abstracción en el que nos encontramos.
- Documentar con Javadoc todas las excepciones que se lanzan en los métodos .

Bibliografía oficial

- <https://docs.oracle.com/javase/tutorial/essential/exceptions/definition.html>
- <https://docs.oracle.com/javase/tutorial/essential/exceptions/runtime.html>
- <https://docs.oracle.com/javase/tutorial/essential/exceptions/handling.html>
- <https://docs.oracle.com/javase/tutorial/essential/exceptions/throwing.html>
- <https://docs.oracle.com/javase/tutorial/essential/exceptions/advantages.html>