

TEMA 2

CONCEPTOS DE JAVA PARA ESTRUCTURAS DE DATOS

Fallos de ejecución: clasificación, representación y tratamiento

Objetivos

Estudiar el tratamiento que se da en *Java* a los errores que se producen al programar:

- Representación como objetos de la clase *Throwable*, de la que heredan *Error* y *Exception*
- Se incidirá en la jerarquía *Exception* ya que en ella se incluyen las *excepciones de usuario*
- Creación y gestión de excepciones: lanzamiento mediante la cláusula *throw*, propagación mediante la cláusula *throws* y su captura mediante la instrucción *try-catch-finally*

Índice

1.- Errores y excepciones en la ejecución de un programa

- Situaciones inesperadas
- Jerarquía de errores y excepciones (*Throwable*)
- Definición de una excepción de usuario

2.- Tratamiento de excepciones

- Lanzamiento de excepciones: la cláusula *throw*
- Propagación de excepciones: la cláusula *throws*
- Captura y manejo de excepciones: la instrucción *try-catch-finally*

1.- ERRORES Y EXCEPCIONES

Situaciones inesperadas

- Cuando ocurre una situación inesperada durante la ejecución de un programa, el programa debe darse cuenta y debe tener la posibilidad de arreglar el problema

Ejemplo:

```
public static Circulo leerCirculo(Scanner teclado) {
    Color colores[] = {Color.black, Color.red, Color.blue};
    System.out.println("Introduzca el radio del círculo\n");
    double radio = teclado.nextDouble();
    System.out.println("Elige el color: 0-Negro 1-Rojo 2-Azul\n");
    Color color = colores[teclado.nextInt()];
    return new Circulo(color, radio);
}
```

¿Qué situaciones inesperadas pueden producirse al ejecutar este código?

1.- ERRORES Y EXCEPCIONES

Situaciones inesperadas

- Situaciones inesperadas posibles:
 - El objeto *teclado* puede apuntar a *null* o a una posición de memoria no válida
 - La entrada o salida estándar pueden no estar disponibles (o no tener permisos de lectura/escritura)
 - El radio que teclea el usuario puede no ser un número real válido
 - El índice del color elegido puede no ser un número entero válido
 - El índice del color elegido puede ser un número entero fuera del rango especificado
 - Puede no haber suficiente memoria para crear nuevas variables
 - Etc.

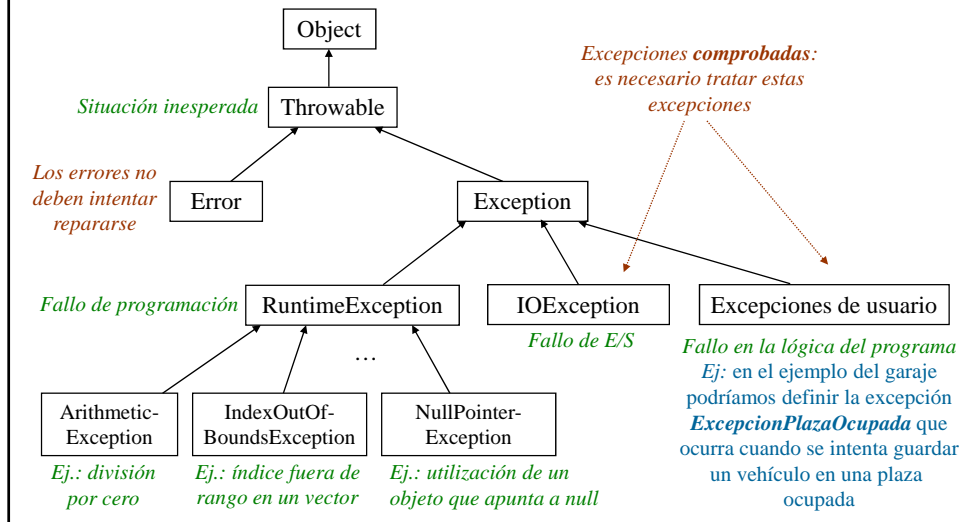
1.- ERRORES Y EXCEPCIONES

Situaciones inesperadas

- Las situaciones inesperadas se clasifican en:
 - Irrecuperables o ***errores***: como agotar la memoria del sistema, ya que se involucra al *hardware*
 - Recuperables o ***excepciones***: como el resto de situaciones inesperadas que hemos comentado:
 - Podemos, por ejemplo, pedir al usuario que vuelva a teclear el radio si el radio que introdujo no era válido
- Los errores/excepciones son clases
- Cuando ocurre un error o una excepción se crea un objeto de la clase correspondiente, que contiene información sobre el problema que ha ocurrido

1.- ERRORES Y EXCEPCIONES

Jerarquía de errores y excepciones



1.- ERRORES Y EXCEPCIONES

Definición de una excepción de usuario

```
public class ExcepcionPlazaOcupada extends Exception {
    /* Constructores */
    public ExcepcionPlazaOcupada(String mensaje) {
        super(mensaje);
    }
    public ExcepcionPlazaOcupada() {
        super();
    }
}
```

Normalmente, una excepción se construye pasándole una cadena (*mensaje*) con una descripción del problema que ha ocurrido. Esta cadena se puede recuperar con el método:

```
String getMessage();
```

, heredado de la clase *Throwable*.

2.- TRATAMIENTO DE EXCEPCIONES

Lanzamiento de una excepción

- Una excepción se lanza con la instrucción *throw*:

```
public void guardarVehiculo(Vehiculo v, int plaza)
    throws ExcepcionPlazaOcupada {
    if (plaza >= 0 && plaza < numPlazas) {
        if (plazas[plaza] != null)
            throw new ExcepcionPlazaOcupada("La plaza " + plaza +
                " está ocupada");
        plazas[plaza] = v;
    }
}
```

→ Propagamos la excepción

- Cuando se lanza una excepción (comprobada) en un método es necesario:
 - O bien propagar la excepción
 - O bien capturar la excepción

2.- TRATAMIENTO DE EXCEPCIONES

Propagación de una excepción

- Cuando se llama a un método que propaga una excepción tenemos también las dos mismas opciones: *propagar* la excepción o *capturarla*

```
public static void main (String args[])
    throws ExcepcionPlazaOcupada {
    Garaje garaje = new Garaje(10);
    garaje.guardarVehiculo(new Coche(100, 5), 1);
}
```

→ Propagamos la excepción

→ Este método puede lanzar la excepción ExcepcionPlazaOcupada. Como no capturamos la excepción, entonces tenemos que volver a propagarla

2.- TRATAMIENTO DE EXCEPCIONES

Captura de una excepción

- Para capturar una excepción se utiliza *try-catch-finally*:

```
public static void main (String args[]) {
    Garaje garaje = new Garaje(10);
    try {
        garaje.guardarVehiculo(new Coche(100, 5), 1);
        System.out.println("Esto solo se muestra si no se lanza " +
            "la excepción");
    } catch (ExcepcionPlazaOcupada e) {
        System.out.println("Error guardando el vehiculo: " +
            e.getMessage());
    } finally {
        System.out.println("Fin del programa");
    }
}
```

Dentro del try se pone el código que puede lanzar una excepción

Pueden haber varios bloques catch ya que se pueden producir distintas excepciones en el bloque try

El código del finally se ejecuta siempre, haya o no haya excepción

2.- TRATAMIENTO DE EXCEPCIONES

Captura de una excepción: ejemplo

- Lectura del *radio* de un círculo manejando las posibles excepciones

```
int radio;
boolean hayError = true;
do {
    try {
        System.out.println("Introduzca el radio del círculo");
        radio = teclado.nextDouble();
        if (radio < 0) throw new ExcepcionRadioNegativo("Radio: "+radio);
        hayError = false;
    } catch (InputMismatchException e1) {
        System.out.println("No has teclado un número. Vuelve a teclear");
    } catch (ExcepcionRadioNegativo e2) {
        System.out.println("No se admiten radios negativos (" +
            e2.getMessage() + "). Vuelve a teclear");
    }
} while (hayError);
```

Clase Scanner:
`public double nextDouble() throws InputMismatchException`

2.- TRATAMIENTO DE EXCEPCIONES

Ejercicios

- Ejercicio 1: Completa la clase **Primitiva** (que almacena 6 números del 1 al 49):

```
public class Primitiva {
    private Integer numeros[];
    private int numGuardados;
    public Primitiva() { ... }
    public void guardarNumero(Integer n)
        throws ExcepcionNumeroDuplicado, ExcepcionNumeroFueraRango,
               ExcepcionCombinacionCompleta { ... }
}
```

Ocurre si se guardan dos números iguales

Ocurre si $n \notin [1,49]$

Ocurre si queremos guardar un número cuando ya hemos guardado los 6

- Ejercicio 2: Escribe un programa que lea por teclado seis números y los guarde en un objeto de clase **Primitiva**. El programa debe tratar las posibles excepciones que puedan aparecer

2.- TRATAMIENTO DE EXCEPCIONES

Ejercicios

- Ejercicio 3: Dada la siguiente definición de clases:

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

public class BaseDeDatos {
    public BaseDeDatos(String nombre) { ... }
    private void abrir() throws ImposibleAbrirBD { ... }
    public void inicializar() throws ImposibleAbrirBD {
        abrir();
        ...
    }
}
```

(Continúa en la siguiente transparencia)

2.- TRATAMIENTO DE EXCEPCIONES

Ejercicios

```
public class TestBaseDeDatos {
    public static void main (String args[])
        throws ImposibleAbrirBD {
        BaseDeDatos bd = new BaseDeDatos("MiBD");
        bd.inicializar();
    }
}
```

Se pide modificar convenientemente el código de la clase *TestBaseDeDatos* para que haga tres intentos de inicializar la base de datos en caso de que se produzca la excepción *ImposibleAbrirBD*. Si al tercer intento no se logra inicializar la base de datos, deberá notificar al usuario que no ha sido posible abrir la base de datos sacando por pantalla un mensaje de error.