Tema 2: Errores y Excepciones: Modelo y Gestión

Germán Moltó
Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

Tema 2: Errores y Excepciones: Modelo y Gestión

Índice general:

Objetivos y Bibliografía

- Estudiar el tratamiento que se proporciona en Java a los errores producidos durante la programación.
- Analizar la jerarquía de Excepciones y Errores en Java y su representación como objetos de la clase Throwable.
- Aprender la sintaxis que proporciona Java para la creación y gestión de excepciones, así como su propagación a otros métodos.

- Bibliografía:
  - Capítulo 7 del libro de Wiener, R. Pinson, L.J. Fundamentals of OOP and Data Structures in Java (Cambridge University Press, 2000).

Modelización Java de la Jerarquía de Errores y Excepciones

- Estudio del tratamiento que da Java a los errores que se producen al programar. En particular, se tratarán los siguientes aspectos:
  - Errores como Objetos **Throwable** (del paquete java.lang).
  - Las subclases de Throwable: **Error**, **Exception** y Excepciones de Usuario.
  - Lanzamiento y Propagación de Excepciones: excepciones capturadas y no capturadas. Cláusula ***throw*** y ***throws.***
  - Captura de Excepciones: Instrucción **try-catch-finally.**
- Ejemplo:
  - Reformulación del tratamiento de errores realizado en lasFiguras

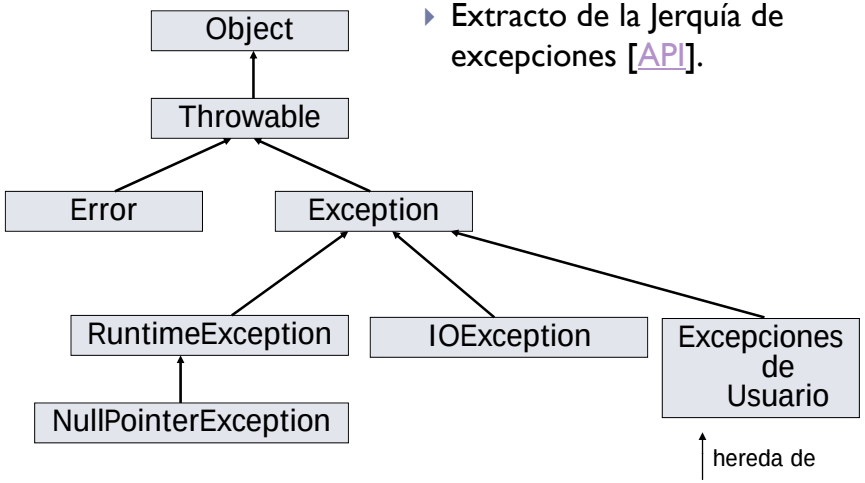
Situaciones Inesperadas en la Ejecución (I)

- Situaciones inesperadas durante la ejecución:
  - Un método intenta leer o escribir en un fichero no accesible.
  - En el proyecto lasFiguras, al leer de teclado un círculo se le asigna a su radio un carácter.
  - Se realiza una división por cero o se calcula la raíz cuadrada de un valor negativo, o se accede a una posición inexistente de un array o se accede a un objeto null, etc.
  - Se produce un fallo en la lógica de la aplicación, como en lasFiguras al insertar una nueva figura en un grupo que ya está lleno o intentar borrar una en un grupo donde no está.
  - Se agota la memoria del sistema o se accede a una zona no permitida de ésta.
- Si no realizamos una **gestión** de los situaciones inesperadas, el programa abortará **abruptamente**.

Situaciones Inesperadas en la Ejecución (II)

- Una buena aplicación debe **prever y gestionar** las situaciones inesperadas que pueden ocurrir en ejecución.
- Debe como mínimo **avisar** de la situación producida y, siempre que sea posible, tratar de **recuperarse** para que no afecte al normal desarrollo del programa.
- Tipos de Situaciones Inesperadas:
  - Errores Irrecuperables → Agotamiento de Memoria
  - Errores Recuperables → Resto de situaciones inesperadas.
    - Excepciones de Entrada / Salida
    - Excepción de Fallo de Programación
    - Excepción de Usuario

La Jerarquía Throwable



- Extracto de la Jerquía de excepciones [API].

Mecanismo de Excepciones: Generalidades

- Las excepciones en Java modelan las situaciones inesperadas.
  - Se corresponden con objetos de tipo **Exception** y se lanzan desde los métodos (lo suelen indicar en su cabecera).
- Clasificamos las excepciones en dos tipos:
  1. Excepciones propias de Java:
     - IOException: FileNotFoundException, UnknownHostException, SocketException, etc.
     - RuntimeException: ArithmeticException, ClassCastException, NullPointerException, IllegalArgumentException, etc.
     - …
  2. Excepciones de Usuario
     - Creadas por el programador para su aplicación.

Definición de Excepciones de Usuario

- Las Excepciones de Usuario, que no pertenecen al estándar de Java, deben ser incorporadas a la jerarquía de la clase **Exception**.
- Ejemplo:
  - Aplicación de gestión de un grupo de figuras donde se desea notificar el error al tratar de insertar en un grupo lleno.

```
public class GrupoLlenoException extends Exception {
    public GrupoLlenoException(String mensaje) {
       super(mensaje);
    }
    public GrupoLlenoException() {
      super();
   }
}
```


¿Cómo se crea una nueva excepción?

- Una excepción es otro objeto más, por lo que se utiliza de la misma manera:
  - Podemos especificar un mensaje adicional o no.
  - Podemos declarar una variable polimórfica cuyo tipo estático sea Exception.

```
...
GrupoLlenoException ex = new GrupoLlenoException();
...
```

```
...
Exception ex = new GrupoLlenoException("Completo!");
...
```


Métodos que Lanzan Excepciones

- Por lo general, los métodos deben indicar en su especificación las excepciones que **pueden** lanzar:
  - Utilizando la clausula throws (separadas por comas).

```
...
public void insertar(Figura f) throws GrupoLlenoException{
 ...
}
...
```

- La excepción podrá ser lanzada dentro del código:
  - De forma explícita (creando un objeto de tipo compatible con la excepción y lanzándolo), ó
  - Tras invocar a un método que la pueda lanzar.

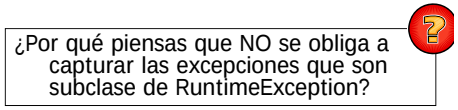
Ejemplo de Método que Lanza Excepción: parseInt de Integer

- El método parseInt de la clase Integer
  - Convierte una cadena que representa un número en un tipo primitivo int.
  - Si la cadena no representa un número (i.e."45" vs "kjd"), lanza una excepción.

```
public class Integer {
  ...
  public static int parseInt(String s) throws NumberFormatException
   {
      if (s == null) throw new NumberFormatException("null");
      ... //Más adelante también puede lanzarse la excepción.
   }
}
```

Excepciones Comprobadas (I)

- Excepción **Comprobada** (checked):
  - Cuando un método h declara en su definición que propaga una Excepción.
  - Al invocar al método h, se debe capturar la excepción o propagarla. Sino, error al compilar.
- Cualquier Excepción de Usuario es Comprobada.
- Las **RuntimeException** nunca son excepciones Comprobadas
  - Los métodos no están obligados a capturar o a propagar una RuntimeException.
  - Tampoco están obligados a declarar que las lanzan
    - Ej.: nextDouble() de Scanner.



Excepciones Comprobadas (II)

- Si un método especifica en su cabecera que lanza una excepción, salvo que sea de tipo RuntimeException, los métodos que lo invoquen están obligados a gestionarla.
- Si el método h sobrescribe al p, h no puede propagar más Excepciones Comprobadas que p.
- Las Excepciones Comprobadas que lanza un método deben ser de un tipo compatible al que declara propagar. Es decir:
  - si se declara propagar una excepción de tipo E,
  - se deberá lanzar una excepción de tipo E o de cualquier subclase de E.

Mecanismo de Gestión de Excepciones

- Invocar un método que lanza una excepción requiere una de estas gestiones:
  - **Capturar** la excepción (gestionándola), ó
  - **Propagar** la excepción (indicándolo en su definición), ó
  - **Capturar, gestionar y propagar** la excepción.
- La propagación de la excepción se hace **en orden inverso** a la secuencia de llamadas realizada.
  - Hasta que las gestiona algún método o la JVM aborta.

Captura de Excepciones

```
try {
   // código que puede lanzar las excepciones e1, e2, etc.
} catch (ExcepcionTipo1 e1) {
   // código de gestión de e1, de tipo ExcepcionTipo1 o subclase de ésta.
} catch (ExcepcionTipo2 e2) {
   // código de gestión de e2, de tipo ExcepcionTipo2 o subclase de ésta.
}finally {
      /* Código que se ejecuta siempre, independientemente de que se
         produzca la excepcion. Sirve para hacer lo indispensable. */
}
```

- Ya que todas las excepciones son subclases de Exception, siempre se puede realizar un try-catch general de Exception y capturar cualquier tipo excepción.

Ejemplo de Excepciones (I): Propagación

- Diseñar un módulo de grabación (clase ModuloGrabacion) que utilice la funcionalidad de GrabacionDVD.

```
public class GrabacionDVD {
   public void grabar() throws ExcepcionEnGrabacion { … }
}
```

```
public class ModuloGrabacion {
    GrabacionDVD grabDVD;
    ...
    public void crearDVD() throws ExcepcionEnGrabacion {
          ...
          grabDVD.grabar();
          ...
    }}
```


Ejemplo de Excepciones (II): Captura

```
public class GrabacionDVD {
   public void grabar() throws ExcepcionEnGrabacion { … }
}
```

```
public class ModuloGrabacion {
    GrabacionDVD grabDVD;
    ...
    public void crearDVD() {
    try{
          grabDVD.grabar();
    }catch(ExcepcionEnGrabacion ex) {
          System.out.println("Te he fastidiado un DVD");
    }
 }}
```

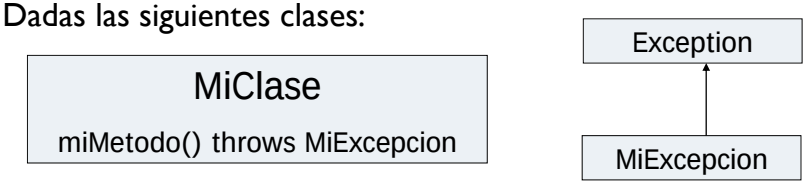

Ejemplo de Excepciones (III): Captura y Propagación

```
public class GrabacionDVD {
   public void grabar() throws ExcepcionEnGrabacion { … }
}
```

```
public class ModuloGrabacion {
    GrabacionDVD grabDVD;
    ...
    public void crearDVD() throws ExcepcionEnGrabacion{
    try{
          grabDVD.grabar();
    }catch(ExcepcionEnGrabacion ex) {
       System.out.println("Te he fastidiado un DVD");
       throw ex; //No es necesario throw new ExcepcionEnGrabacion("..");
    }}}
```


Ejercicio Excepciones

- Dadas las siguientes clases:



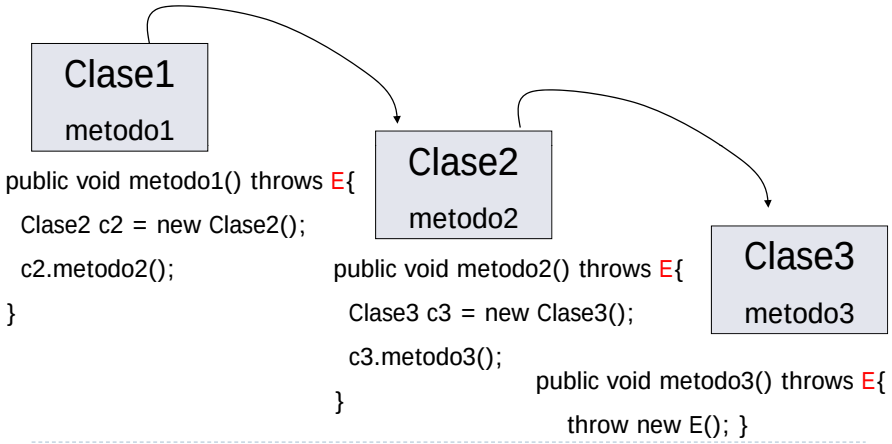
- ¿Compilaría el siguiente código?:

```
public static void main(String args[]){
  MiClase mc = new MiClase();
  mc.miMetodo();
}
```

- Si no compila, realizar las modificaciones oportunas.

Propagación de Excepciones en Java

- ¿Cuál será el orden de propagación de la excepción E lanzada en método 3?



Ejemplo: Gestión Grupo de Figuras (I)

- Adaptación de la sesión 2 de la práctica 1:

```
public interface GrupoDeFiguras{
   public void insertar(Figura f);
   public Figura eliminar(Figura fig);
   public Figura buscar(Figura fig);
   public Figura recuperar(int pos);
   public int talla();
   public double area();
   public String toString();
   public inicializar(Scanner teclado);
}
```


Ejemplo: Gestión Grupo de Figuras (II)

- Clase ArrayGrupoDeFiguras

```
public class ArrayGrupoDeFiguras implements GrupoDeFiguras{
   private Figura elArray[];
   private int talla;
   private static final int CAPACIDAD_DEL_ARRAY = 10;
   public ArrayGrupoDeFiguras(){
      elArray = new Figura[CAPACIDAD_DEL_ARRAY];
      talla = 0;
   }
  //Implementación de los métodos de la interfaz GrupoDeFiguras
}
```


Ejemplo: Grupo de Figuras (II)

```
public class TestGrupoDeFiguras
{
   public static void main(String args[]){
      Scanner teclado = new Scanner(System.in);
      GrupoDeFiguras gdf = new ArrayGrupoDeFiguras();
      gdf.inicializar(teclado);
      System.out.println("Grupo de Figuras actual: " + gdf.toString());
      gdf.insertar (new Circulo());
      Figura c = new Circulo();
      System.out.println("Buscar en el Grupo el circulo: " + c);
      int posicionC = gdf.buscar(c);
      if (posicionC != null ) System.out.println("Aparece en la posicion: " + posicionC);
      else System.out.println("El Circulo buscado NO pertenece al Grupo");
      System.out.println("Borrar del Grupo el Circulo: " + c);
      Figura borrada = gdf.eliminar(c);
   }
}
```

Ejemplo de Gestión de Excepciones en TestGrupoDeFiguras (I)

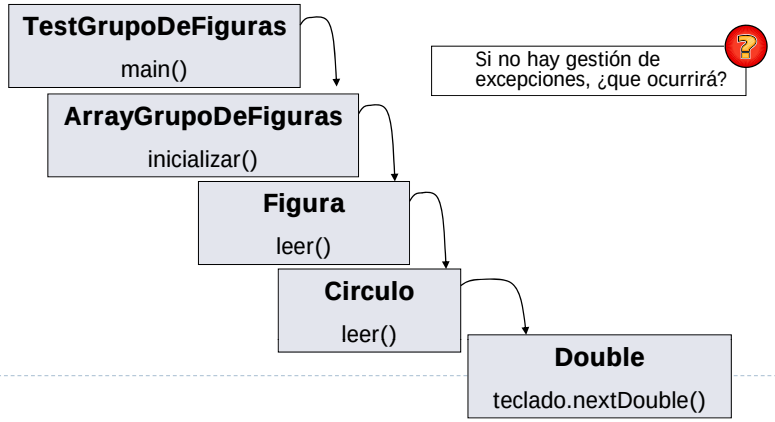
- Previsiones:
  1. Fallo al inicializar el grupo de figuras desde teclado
  2. Fallo al insertar una nueva figura en el grupo, si ya está lleno
  3. Fallo al tratar de borrar una figura inexistente
- Gestión en el caso de que se produzcan fallos de tipo
  1. Recuperarla: detectarla y dar la oportunidad de volver a introducir el dato hasta que sea correcto
  2. Avisar como mínimo, aunque lo mejor es recuperarla (aumentar la talla del grupo para realizar la inserción)
  3. Avisar como mínimo, aunque se podría recuperar (ofrecer la posibilidad de borrar otra)

Lectura de Teclado con la clase Scanner

- java.util.Scanner permite hacer lectura de teclado formateada.
  - Scannner teclado = new Scanner(System.in);
  - String s = teclado.nextLine(); String s = teclado.next();
  - int i = teclado.nextInt();
- Los métodos pueden lanzar la excepción InputMismatchException …
  - Si el tipo del dato leído no puede ser convertido al especificado por el usuario.
  - InputMismatchException es subclase de RuntimeException y, por lo tanto, el usuario NO está obligado a capturar la excepción (aunque puede ser recomendable).

Gestión de Excepciones en TestGrupoDeFiguras (II)

- Supongamos que al inicializar un GrupoDeFiguras desde teclado, el usuario introduce un carácter (en vez de un dígito) al especificar el radio del Círculo.



Notificación de Excepción al Insertar

- Para notificar que el grupo está lleno al insertar …
  - Modificar la especificación del método y su implementación:

```
public void insertar (Figura f) throws GrupoLlenoException{
  if ( this.talla == elArray.length)
       throw new GrupoLlenoException("Lleno!");
  …
}
```

- Si el método que invoca al que lanza la Excepción no la captura y gestiona, la máquina Java se encarga de relanzarla, siguiendo en sentido inverso la secuencia de métodos invocados hasta el que la lanzó:
  - insertar de ArrayGrupoDeFiguras la relanza al main de TestGrupoDeFiguras.

Captura de ExcepcionGrupoLleno

```
import excepciones.*;
public class TestGrupoDeFiguras  {
public static void main(String args[]) {
    ...
    try{
          g.insertar(new Circulo());
    }catch(GrupoLlenoException e){
          System.out.println(e);
          e.printStackTrace();
    }
...
 }
```

- En caso de producirse la excepción, se obtendría el mensaje "Lleno!".

Gestión Alternativa de Grupo Lleno (I)

- Gestión silenciosa de la situación inesperada:
  - Duplicar el tamaño del vector para insertar satisfactoriamente.

```
public void insertar(Figura f) {
   if ( this.talla == elArray.length)  duplicarArray();
   this.elArray[talla] = f; this.talla++;
}
private void duplicarArray() {
   Figura nuevo[] = new Figura[elArray.length * 2];
   for (int i = 0; i < elArray.length; i++) nuevo[i] = elArray[i];
   this.elArray = nuevo;
}
```

Si insertar realiza esta gestión ¿Qué ocurre si se deja el código del main de TestGrupoDeFiguras como en la transparencia anterior?

Gestión Alternativa de Grupo Lleno (II)

- Es posible utilizar un esquema mixto:
  - Gestión automática del tamaño del vector y, además,
  - Notificación al usuario mediante una excepción.

```
public void insertar(Figura f) throws GrupoLlenoException {
   try{
         this.elArray[talla] = f;
   } catch(ArrayIndexOutOfBoundsException e){
         duplicarArray();
         this.elArray[talla] = f;
         throw  new  GrupoLlenoException("Se ha aumentado el tamaño del
                                          Grupo para insertar \n"+f);
   }
    this.talla++;
 }
```


Fallos de I/O en leer de Figura (I)

```
public static Figura leer(Scanner teclado) {
      Figura res = null;
      System.out.println("*** LEYENDO DESDE TECLADO ***\n");
      System.out.println("Pulse: \n 1 para leer Círculo \n 2 para leer Rectángulo \n 3
   para leer Cuadrado. \n 0 para salir.");
      int opcion = leerIntValido(teclado);
      if ( opcion == 1 ) res = Circulo.leer(teclado);
      else if ( opcion == 2 ) res = Rectangulo.leer(teclado);
      else if (opcion == 3) res = Cuadrado.leer(teclado);
      return res;
}
```

- Modificamos la lectura del *int* para que, en caso de error, se pregunte de nuevo al usuario los datos de entrada.
- Errores controlados:
  - *Inserción de carácter en lugar de entero.*
  - *Inserción de entero fuera del intervalo [0,3].*

Fallos de I/O en leerIntValido (II)

```
private static int leerIntValido(Scanner teclado) {
   boolean hayError=true;
   int res = 0;
    do {
       try {
              res = teclado.nextInt();  hayError = false;
              if ( res > 3 || res < 0 ) hayError = true;
         } catch (InputMismatchException eUnchecked){  teclado.next(); }
        if (hayError){
              System.out.println("Introduce una opción válida");
              System.out.println("1 para leer Círculo, 2 para leer Rectángulo, 3 para
                                  leer Cuadrado y  0 para SALIR\n");
           }
     } while (hayError);
   return res;
}
```