Programación Orientada a Objetos

Tema 5:
Tratamiento de errores

Contenidos

- Tema 5: Tratamiento de errores: excepciones

1. INTRODUCCIÓN
2. MANEJO DE EXCEPCIONES EN JAVA
3. TIPOS DE EXCEPCIONES
   - Excepciones predefinidas
   - Excepciones definidas por el usuario
4. PROPAGACIÓN DE EXCEPCIONES

INTRODUCCIÓN

- Cuando sucede un **evento anormal en la ejecución de un programa** y lo detiene decimos que se ha producido una **excepción**. Cualquier programa bien escrito debe ser capaz de tratar las posibles excepciones que pueden ocurrir en su ejecución de manera inteligente y de recuperarse, si es posible, de ellos.

- Con los **mecanismos de recuperación ante excepciones construimos programas robustos** y con capacidad de recuperación ante errores más o menos previsibles que se pueden producir en el momento en que se ejecuta el programa.

- En Java **una excepción es un objeto** que avisa que ha ocurrido alguna condición inusual. Existen muchos objetos de excepción predefinidos, y también podremos crear los nuestros propios.

- Cuando se capturan excepciones en Java las sentencias que pueden causar un error se deben insertar en un **bloque formado por *try* y *catch***, a continuación, en *catch* debemos tratar esos posibles errores. También existe la posibilidad de lanzar una excepción cuando se produzca una determinada situación en la ejecución del programa, para hacerlo utilizaremos la instrucción ***throw***.

MANEJO DE EXCEPCIONES

- **Ejemplo de programa que no trata excepciones:**

```
public class EjExcepcion {
  public static void main (String args[]) {
        String cadenas[] = {
         "Cadena 1",
         "Cadena 2",
         "Cadena 3",
         "Cadena 4"
         };

    for (int i=0; i<=4; i++) System.out.println(cadenas[i]);
  }
}
```

- Como resultado de la ejecución del programa obtendremos:

```
> java EjExcepcion
Cadena 1
Cadena 2
Cadena 3
Cadena 4
java.lang.ArrayIndexOutOfBoundsException
    at EjExcepcion.main(EjExcepcion.java, Compiled Code)
Exception in thread "main" Process Exit...
```

MANEJO DE EXCEPCIONES

- **Ejemplo de programa que trata excepciones:**
- Vamos a reescribir el programa anterior para tratar las excepciones:

```
public class EjExcepcionBien {
  public static void main (String args[]) {
    String cadenas[] = {"Cadena 1","Cadena 2","Cadena 3","Cadena 4"};
    try {
         for (int i=0; i<=4; i++) System.out.println(cadenas[i]);
    } catch( ArrayIndexOutOfBoundsException aie ) {
        System.out.println("\nError: Fuera del índice del array\n");
    } catch( Exception e ) {
        // Captura cualquier otra excepción
        System.out.println("Excepción: " + e.toString() );
    } finally {
        System.out.println( "Esto se imprime siempre." );
    }
  }
}
```

- Como resultado de la ejecución del programa obtendremos:

```
> java EjExcepcionBien
Cadena 1
Cadena 2
Cadena 3
Cadena 4

Error: Fuera del índice del array
Esto se imprime siempre.
```


Universidad de Alcalá
Departamento Ciencias de la Computación

MANEJO DE EXCEPCIONES

- Vamos a modificar el programa para que se produzca otro tipo de excepción, en este caso una división por cero y se ejecute el bloque del segundo *catch*:

```
public class EjExcepcionBien {
  public static void main (String args[]) {
    int valor = 0;
    String cadenas[] = {
         "Cadena 1",
         "Cadena 2",
         "Cadena 3",
         "Cadena 4"
    };
    try {
         for (int i=0; i<=4; i++) {
                 System.out.println(cadenas[i]);
                 valor = i/0;
         }
    } catch( ArrayIndexOutOfBoundsException aie ) {
        System.out.println("\nError: Fuera del índice del array\n");
    } catch( Exception e ) {
        // Captura cualquier otra excepción
        System.out.println("Excepción: " + e.toString() );
    } finally {
        System.out.println( "Esto se imprime siempre." );
    }

  }
}
```

MANEJO DE EXCEPCIONES

- Como resultado de la ejecución del programa obtendremos:

```
> java EjExcepcionBien
Cadena 1
Excepción: java.lang.ArithmeticException: / by zero
Esto se imprime siempre.
```

- En este caso el programa ha ejecutado:

```
catch( Exception e ) {
        // Captura cualquier otra excepción
        System.out.println("Excepción: " + e.toString() );
}
```

que funciona como un mecanismo general de captura de excepciones, es decir, cualquier excepción no tratada anteriormente es tratada aquí.

MANEJO DE EXCEPCIONES - try

- ***try***
- **Define un bloque de código donde se puede generar una excepción.** El bloque *try* va seguido inmediatamente de uno o más bloques *catch* y opcionalmente de una cláusula *finally*. Cuando se lanza una excepción el control sale del bloque try actual y pasa a un manejador *catch* apropiado.
- La **sintaxis** general del bloque *try* consiste en la palabra clave **`try`** y una o más sentencias entre llaves.

```
try {
    // Sentencias Java
}
```

- Se pueden presentar dos **situaciones** diferentes a la hora de definir el bloque *try*:
  - Podemos tener más de una sentencia que generen excepciones, en cuyo caso podemos definir bloques individuales para tratarlos.
  - Podemos tener agrupadas en un mismo bloque *try* varias sentencias que puedan generar excepciones, con lo que habría que asociar múltiples controladores a ese bloque.

Será la decisión del programador utilizar una forma u otra de controlar las excepciones.

MANEJO DE EXCEPCIONES - catch

- ***catch***
- **Define el bloque de sentencias que se ejecutarán cuando se haya producido una excepción** en un bloque *try*.
- La **sintaxis** general de la sentencia `catch` en Java es la siguiente:

```
catch( TipoExcepcion nombreVariable ) {
  // sentencias Java
}
```

- Se pueden colocar sentencias *catch* sucesivas, cada una controlando una excepción diferente.
- No debería intentarse capturar todas las excepciones con una sola cláusula ya que representa un uso demasiado general y podrían llegar muchas excepciones.

```
catch( Exception e ) { … } //captura genérica
```


MANEJO DE EXCEPCIONES - finally

- ***finally***
- El bloque de código definido en *finally* se ejecuta siempre, haya o no excepción.

```
public class EjExcepcionFinally {
    public static void main(String[] args) {
        String cadenas[] = {"Cadena 1", "Cadena 2", "Cadena 3", "Cadena 4"};
        try {
            for (int i = 0; i <= 3; i++) { //Correcto
                System.out.println(cadenas[i]);
            }
        } catch (ArrayIndexOutOfBoundsException aie) {
            System.out.println("\nError: Fuera del índice del array\n");
        } catch (Exception e) {
            // Captura cualquier otra excepción
            System.out.println("Excepción: " + e.toString());
        } finally {
            System.out.println("Esto se imprime siempre.");
        }
    }
}
```

–Ejecución del programa:

```
Cadena 1
Cadena 2
Cadena 3
Cadena 4
Esto se imprime siempre.
```

MANEJO DE EXCEPCIONES - throw

- ***throw***

- La sentencia *throw* se ejecuta para indicar que ha ocurrido una excepción, o **lanzamiento de una excepción**. La sentencia *throw* especifica el objeto que se lanzará. La forma general de la sentencia *throw* es:

```
throw ObjetoThrowable;
```

- **El flujo de la ejecución se detiene inmediatamente** después de la sentencia *throw*, y nunca se llega a la sentencia siguiente, ya que el control sale del bloque *try* y pasa a un manejador *catch* cuyo tipo coincide con el del objeto. Si se encuentra, el control se transfiere a esa sentencia. Si no, se inspeccionan los siguientes bloques hasta que el gestor de excepciones más externo detiene el programa.

MANEJO DE EXCEPCIONES - throw

```
// Ejemplo de tratamiento de excepciones con throw
import java.io.*;
public class ExcepcionThrow {
   public static void main(String args[]) {
      double op1, op2, resd;       String resp = "";       BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
      //bucle infinito aunque se produzca una excepción
      while (true) {
         try {
            System.out.println("\n######## Nueva División ########");
            System.out.print("\nNumerador: ");
            op1 = Double.parseDouble(entrada.readLine());
            System.out.print("\nDenominador: ");
            op2 = Double.parseDouble(entrada.readLine());
            if (op2 == 0) {
               throw new ArithmeticException("División por cero");
            }
            resd = op1 / op2;
            System.out.println("\nResultado: " + Double.toString(resd));
            System.out.println("\nPulse s para salir.");
            resp = entrada.readLine().toUpperCase();
            if (resp.equals("S")) {   return;   }
         } catch (ArithmeticException aie) {   System.out.println("\nError aritmético: " + aie.toString());
         } catch (NumberFormatException nfe) {   System.out.println("\nError de formato numérico: " + nfe.toString());
         } catch (IOException ioe) {  System.out.println("\nError de entrada/salida: " + ioe.toString());
         } catch (Exception e) {  System.out.println("Excepción: " + e.toString());
         }
      }
   }
}
```

MANEJO DE EXCEPCIONES - throws

- ***throws***

- Con ***throws*** **un método lista las excepciones que puede lanzar, y que no va a manejar**. Esto sirve para que todos los métodos que lo llamen puedan colocar protecciones frente a esas excepciones.

- Para muchas de las subclases de la clase **Exception**, el compilador **Java obliga a declarar qué tipos de excepciones podrá lanzar un método**.

- Si un método lanza explícitamente una instancia de **Exception** o de sus subclases, se debe declarar su tipo con la sentencia ***throws***. La declaración del método sigue ahora la sintaxis siguiente:

```
tipo nombreMetodo( argumentos ) throws excepciones { … }
```


MANEJO DE EXCEPCIONES - throws

```
// Ejemplo de tratamiento de excepciones con throws
import java.io.*;
public class ExcepcionThrows {
   public static void main(String args[]) {
      double op1, op2, resd;    String resp = "";   BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
      //bucle infinito aunque se produzca una excepción
      while (true) {
         try {
            System.out.println("\n######## Nueva División ########");
            System.out.print("\nNumerador: ");
            op1 = capturanum();
            System.out.print("\nDenominador: ");
            op2 = capturanum();
            if (op2 == 0) {
               throw new ArithmeticException("División por cero");
            }
            resd = op1 / op2;
            System.out.println("\nResultado: " + Double.toString(resd));
            System.out.println("\nPulse s para salir.");
            resp = entrada.readLine().toUpperCase();
            if (resp.equals("S")) {  return;  }
         } catch (ArithmeticException aie) {  System.out.println("\nError aritmético: " + aie.toString());
         } catch (NumberFormatException nfe) {  System.out.println("\nError de formato numérico: " + nfe.toString());
         } catch (IOException ioe) {  System.out.println("\nError de entrada/salida: " + ioe.toString());
         } catch (Exception e) {  System.out.println("Excepción: " + e.toString());
         }
      }
   }
```

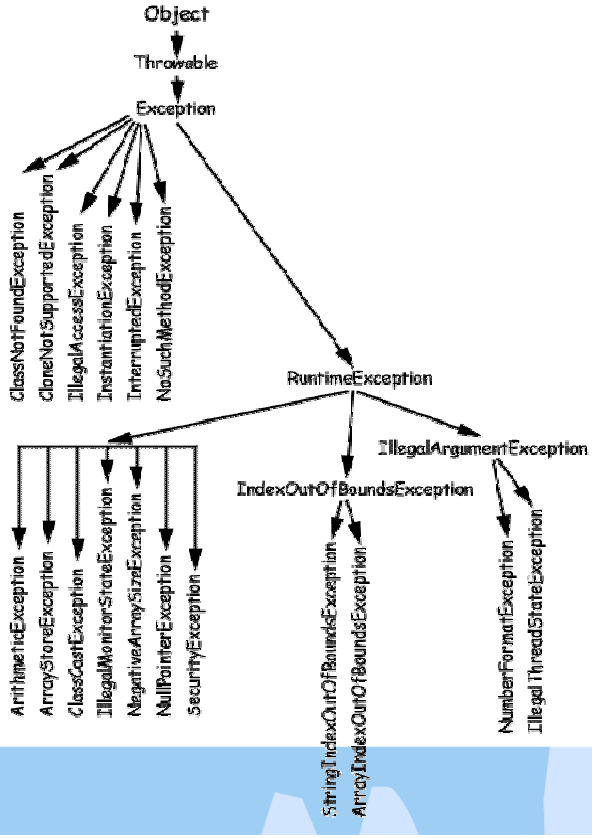
MANEJO DE EXCEPCIONES - throws

```
    public static double capturanum()
                            throws NumberFormatException, IOException
    {
            BufferedReader entrada=
                    new BufferedReader(new InputStreamReader(System.in));

            String txt = entrada.readLine();
            double num = Double.parseDouble(txt);
            return num;
    }

}
```


TIPOS DE EXCEPCIONES

- En Java las excepciones forman una **completa jerarquía de clases**, cuya clase base es **Throwable**. La clase **Throwable** tiene dos subclases: **Error** y **Exception**.
- Un ***Error*** indica que se ha producido un fallo del que **no se puede recuperar la ejecución normal del programa**, por lo tanto, este tipo de errores no se pueden tratar.
- Una ***Exception*** indicará una **condición anormal que puede ser resuelta** para evitar la terminación de la ejecución del programa. Hay varias **subclases** de la clase **Exception** conocidas como *excepciones predefinidas*, y una de ellas *RuntimeException*, a su vez, tiene numerosas subclases.
- **El usuario puede definirse sus propias excepciones**, tan solo tendrá que extender la clase **Exception** y proporcionar la funcionalidad extra que requiera el tratamiento de esa excepción.
- Todas las excepciones deben llevar un **mensaje asociado** a ellas al que se puede acceder utilizando el método ***getMessage()***, que presentará un mensaje describiendo el error o la excepción que se ha producido. El método ***toString()*** aplicado a una excepción produce un resultado similar.

TIPOS DE EXCEPCIONES
EXCEPCIONES PREDEFINIDAS



TIPOS DE EXCEPCIONES
EXCEPCIONES PREDEFINIDAS

- **Excepciones *predefinidas* más comunes:**

  –***ArithmeticException***
  Las excepciones aritméticas son típicamente el resultado de una división por 0:
  ```
  int i = 12 / 0;
  ```

  –***NullPointerException***
  Se produce cuando se intenta acceder a una variable o método antes de ser definido.

  –***ClassCastException***
  El intento de convertir un objeto a otra clase que no es válida.
  ```
  y = (Prueba)x;        // donde x no es de tipo Prueba
  ```

  –***NegativeArraySizeException***
  Puede ocurrir si se intenta definir el tamaño de un array con un número negativo.

  –***ArrayIndexOutOfBoundsException***
  Se intenta acceder a un elemento de un array que está fuera de sus límites.

  –***NoClassDefFoundException***
  Se referenció una clase que el sistema es incapaz de encontrar.

TIPOS DE EXCEPCIONES
EXCEPCIONES PREDEFINIDAS

- El compilador Java **obliga al programador a proporcionar el código de manejo o control de algunas de las excepciones predefinidas** por el lenguaje.
- Por ejemplo, el siguiente programa, no compilará porque no se captura la excepción *IOException* que puede lanzar el método *readLine()* definido de la siguiente forma: *public String readLine() throws IOException*

```
import java.io.*;
class ErrorExcepcion {
    public static void main( String args[] ) {
        String cadena;
        BufferedReader entrada =
            new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Introduce una cadena:");
        cadena = entrada.readLine();
    }
}
```

- Al intentar compilar, producirá el siguiente error de compilación:

```
> javac ErrorExcepcion.java
ErrorExcepcion.java:9: unreported exception java.io.IOException; must be
caught or declared to be thrown
```


TIPOS DE EXCEPCIONES
EXCEPCIONES DEFINIDAS POR EL USUARIO

- También podemos crear **nuestras propias excepciones**, tan solo tenemos que realizar una clase que herede de la clase **Exception** y asociarle un mensaje descriptivo del error.
- El siguiente programa hace uso de una excepción definida por el usuario. El programa realiza la media de una serie de notas introducidas por el usuario, cuando éste intenta poner una nota inferior a 0 o superior a 10, salta la excepción NotaMal. También se han tenido en cuenta otras posibles excepciones predefinidas como errores de formato numérico, errores de entrada salida, etc.

```
// Notas.java

import java.io.*;

public class Notas {
    public static void main( String args[] ) {
        double media=0, total=0, notanum=0;
        int contador=0;
        String notatxt="";
        BufferedReader entrada = new BufferedReader(new InputStreamReader(System.in));
        while ( ! notatxt.equals("Z") ) {
            try {
                System.out.print( "\nTeclee calificación (0-10), Z para terminar: " );
                notatxt = entrada.readLine().toUpperCase();
                notanum = Double.parseDouble(notatxt);
                if (notanum < 0 || notanum > 10) throw new NotaMal();
                total += notanum;
                contador++;
            }
```

TIPOS DE EXCEPCIONES

EXCEPCIONES DEFINIDAS POR EL USUARIO

```
                    } catch (NotaMal nm) {
                        System.out.println( "\n" + nm.getMessage() );
                    } catch( NumberFormatException nfe ) {
                        if ( ! notatxt.equals("Z") )
                            System.out.println("\nError de formato numérico: " + nfe.toString());
                    } catch( IOException ioe ) {
                        System.out.println( "\nError de entrada/salida: " + ioe.toString() );
                    } catch( Exception e ) {
                        // Captura cualquier otra excepción
                        System.out.println( "Excepción: " + e.toString() );
                    }
                }

                if ( contador != 0 ) {
                    media = (double) total / contador;
                    System.out.println( "\nEl promedio del grupo es: " + media );
                } else
                    System.out.println( "\nNo se introdujeron calificaciones." );
        }
}

class NotaMal extends Exception
{
        public NotaMal()
        {
                super( "Excepción definida por el usuario: NOTA INCORRECTA." );
        }
}
```


TIPOS DE EXCEPCIONES

EXCEPCIONES DEFINIDAS POR EL USUARIO

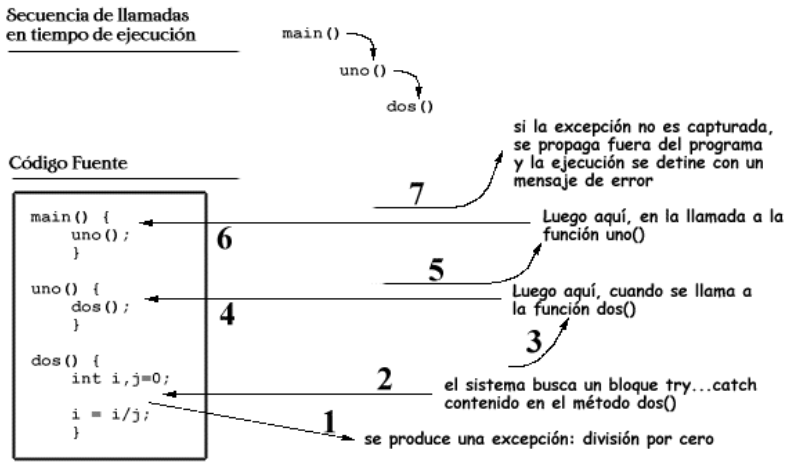
- Podemos modificar la excepción NotaMal para que admita un mensaje que le pasamos como parámetro.

```
// Notas2.java
import java.io.*;
public class Notas2 {
   …
   public static void main(String[] args) {
      while (!notatxt.equals("Z")) {
         try {
            System.out.print("\nTeclee calificación (0-10), Z para terminar: ");
            notatxt = entrada.readLine().toUpperCase();
            notanum = Double.parseDouble(notatxt);
            if (notanum < 0) {
               throw new NotaMal2("No se admiten calificaciones menores que 0.");
            } else if (notanum > 10) {
               throw new NotaMal2("No se admiten calificaciones mayores que 10.");
            }
            total += notanum;
            contador++;
         } catch (NotaMal2 nm) {
            System.out.println("\n" + nm.getMessage());
         }
…
}

class NotaMal2 extends Exception {
   public NotaMal2() {
      super("Excepción definida por el usuario: NOTA INCORRECTA.");
   }
   public NotaMal2(String msg) {
      super("Excepción definida por el usuario: " + msg);
   }
}
```

PROPAGACIÓN DE EXCEPCIONES

- Cuando una excepción no es tratada en el método en donde se produce, se busca en el bloque *try..catch* del método que hizo la llamada. Si la excepción se propaga de todas formas hasta lo alto de la pila de llamadas sin encontrar un controlador específico para la excepción, entonces la ejecución se detendrá dando un mensaje de error con la excepción no tratada.

- En el siguiente diagrama se muestra gráficamente cómo se propaga una excepción que se genera en un método, a través de la pila de llamadas durante la ejecución de un programa:



PROPAGACIÓN DE EXCEPCIONES

```
// PropExcepciones.java
public class PropExcepciones {
    public static void main(String args[]) {
        try {
            metodo1();
        } catch (ExcepcionUsuario eu) {
            System.err.println("\nCapturada excepción en método1: " + eu.getMessage() +
                    "\n\nTrazado de la pila:\n");
            eu.printStackTrace(System.out);
        }
    }

    public static void metodo1() throws ExcepcionUsuario {
        metodo2();
    }

    public static void metodo2() throws ExcepcionUsuario {
        metodo3();
    }

    public static void metodo3() throws ExcepcionUsuario {
        throw new ExcepcionUsuario();
    }
}

class ExcepcionUsuario extends Exception {
    public ExcepcionUsuario() {
        super("Excepción definida por el usuario.");
    }
}
```

PROPAGACIÓN DE EXCEPCIONES

- En este programa se hace uso del método ***printStackTrace(System.out)***. Invocando a este método sobre una excepción se volcará a pantalla todas las llamadas hasta el momento en donde se generó la excepción.

- Por ejemplo, la ejecución del programa presentará por pantalla:

```
>java PropExcepciones
Capturada excepción en método1: Excepción definida por el usuario.

Trazado de la pila:

ExcepcionUsuario: Excepción definida por el usuario.
   at PropExcepciones.metodo3(PropExcepciones.java:29)
   at PropExcepciones.metodo2(PropExcepciones.java:24)
   at PropExcepciones.metodo1(PropExcepciones.java:19)
   at PropExcepciones.main(PropExcepciones.java:7)
```