Control de excepciones.

Desarrollo de Aplicaciones Multiplataforma. Programación.

Control de excepciones.

- Uno de los problemas más importantes al escribir aplicaciones es el tratamiento de los errores.
- Los errores detienen la ejecución del programa e impiden su desarrollo normal y, lo peor, además provocan que el usuario esté desinformado. Toda programadora o programador tiene que reconocer las situaciones que pueden provocar el fin de la ejecución normal del programa por un error no controlado. Dicho de otra forma, todos los posibles errores en un programa deben de estar controlados.
- Java nos echa una mano para ello a través de las **excepciones**. Se denomina excepción a un hecho que podría provocar la detención del programa; es decir, una condición de error en tiempo de ejecución pero que puede ser controlable (a través de los mecanismos adecuados).
- En Java sin embargo se denomina **error** a una condición de error incontrolable (ejemplos son el error que ocurre cuando no se dispone de más memoria o errores de sintaxis).
- Ejemplos de excepciones son:
    - El archivo que queremos abrir no existe
    - Falla la conexión a una red
    - Se intenta dividir entre cero

try-catch

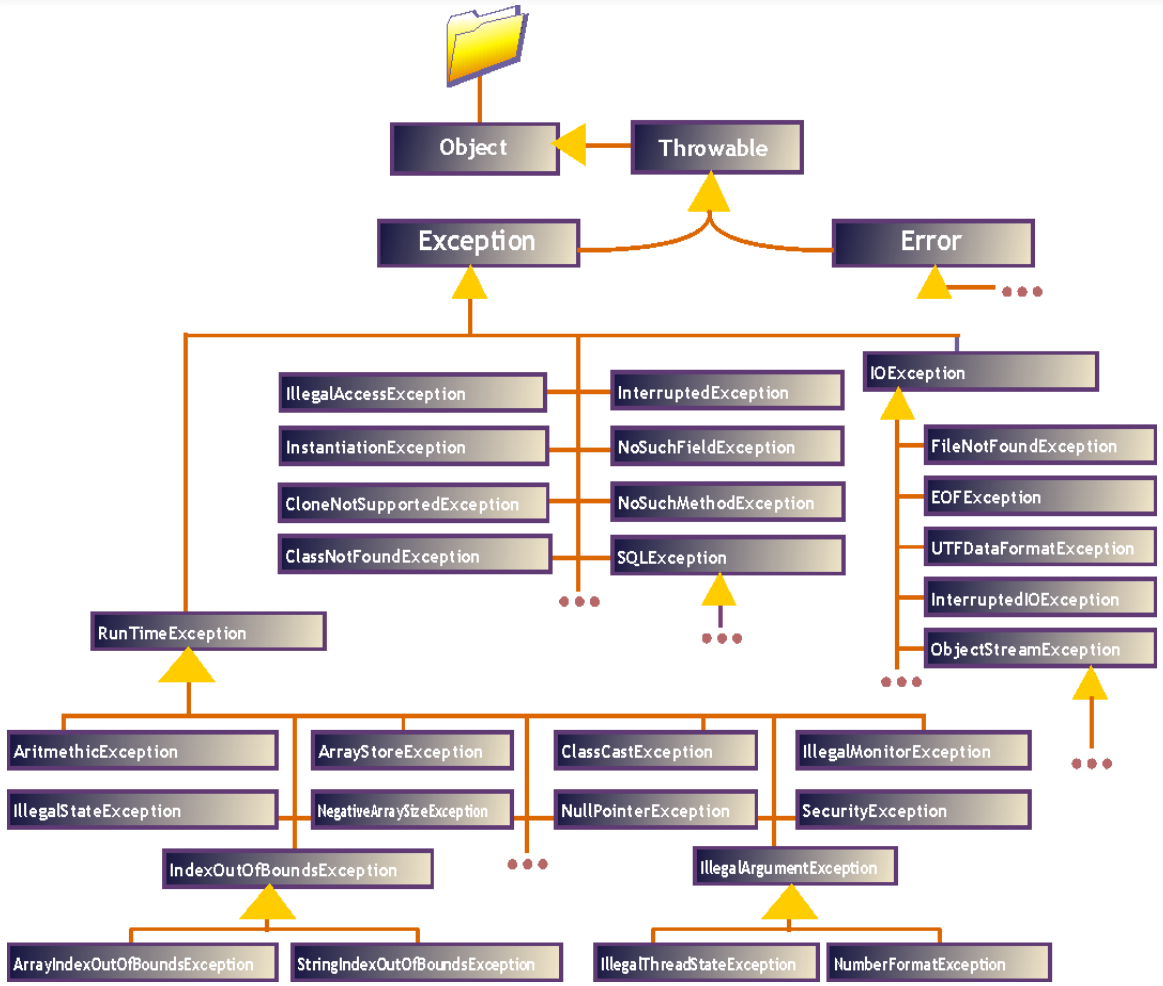
- El control de las excepciones se realiza mediante las sentencias try y catch. La sintaxis es:

```
try {
        instrucciones que se ejecutan salvo que haya un error
}
catch (ClaseExcepción objetoQueCapturaLaExcepción) {
        instrucciones que se ejecutan si hay un error
}
```

- Puede haber más de una sentencia catch para un mismo bloque try. Ejemplo:

```
try {
        readFromFile("arch");
        ...
}
catch(FileNotFoundException e) {
        //archivo no encontrado
        ...
}
catch (IOException e) {
        ...
}
```

Diagrama para el control de excepciones.



try-catch

- Dentro del bloque **try** se colocan las instrucciones susceptibles de provocar una excepción, el bloque **catch** sirve para capturar esa excepción y evitar el fin de la ejecución del programa. Desde el bloque catch se maneja, en definitiva, la excepción.
- Cada catch maneja un tipo de excepción. Cuando se produce una excepción, se busca el catch que posea el manejador de excepción adecuado, será el que utilice el mismo tipo de excepción que se ha producido. Esto puede causar problemas si no se tiene cuidado, ya que la clase Exception es la superclase de todas las demás. Por lo que si se produjo, por ejemplo, una excepción de tipo AritmethicException y el primer catch captura el tipo genérico Exception, será ese catch el que se ejecute y no los demás.

try-catch

- Por eso el último catch debe ser el que capture excepciones genéricas y los primeros deben ser los más específicos. Lógicamente si vamos a tratar a todas las excepciones (sean del tipo que sean) igual, entonces basta con un solo catch que capture objetos Exception. Ejemplo (de mal uso):

```
int x;
try{
        x=Integer.parseInt( JOptionPane.showInputDialog("Escriba un número"));
}
catch (Exception e){
        JOptionPane.showMessageDialog(null, "Error indeterminado");
}
catch (NumberFormatException e){
        JOptionPane.showMessageDialog(null, "El número no es válido");
}
```

- El código enmarcado en rojo no es alcanzable, porque siempre se ejecutaría el primer catch. De hecho hoy en día Java marca como error ese código.

Manejo de excepciones

- Cuando se produce un error, se asume que es irrecuperable. Por tanto, se anula la instrucción que dio pie al error.

- En caso de querer que se maneje el error y regresar de nuevo al código que lo provocó, deberíamos ser nosotros los que lo controlemos encerrando el bloque **try** en un bucle que se repetirá hasta que el error deje de existir.

```
public static void main(String[] args) {
    boolean indiceNoValido = true;
    int i; //Entero que tomará nos aleatorios de 0 a 9
    String texto[] = {"Uno", "Dos", "Tres", "Cuatro", "Cinco"};
    while (indiceNoValido) {
        try {
            i = (int) (Math.round(Math.random() * 9));
            System.out.println(texto[i]);
            indiceNoValido = false;
        } catch (ArrayIndexOutOfBoundsException exc) {
            System.out.println("Fallo en el índice");
        }
    }
}
```

Métodos de la clase Exception.

- La clase Exception es la superclase de todos los tipos de excepciones. Esto permite utilizar una serie de métodos comunes a todas las clases de excepciones:
- String getMessage(): obtiene el mensaje descriptivo de la excepción o una indicación específica del error ocurrido:
- ```
  try{
  ```
- ```
  } catch (IOException ioe){
  ```
- ```
  System.out.println(ioe.getMessage());
  ```
- ```
  }
  ```
- String toString(): escribe una cadena sobre la situación de la excepción. Suele indicar la clase de excepción y el texto de getMessage().
- void printStackTrace(): escribe el método y mensaje de la excepción (la llamada información de pila). El resultado es el mismo mensaje que muestra el ejecutor (la máquina virtual de Java) cuando no se controla la excepción.

Métodos de la clase Exception.

- La clase Exception es la superclase de todos los tipos de excepciones. Esto permite utilizar una serie de métodos comunes a todas las clases de excepciones:
- String getMessage(): obtiene el mensaje descriptivo de la excepción o una indicación específica del error ocurrido:
- ```
  try{
  ```
- ```
  } catch (IOException ioe){
  ```
- ```
  System.out.println(ioe.getMessage());
  ```
- ```
  }
  ```
- String toString(): escribe una cadena sobre la situación de la excepción. Suele indicar la clase de excepción y el texto de getMessage().
- void printStackTrace(): escribe el método y mensaje de la excepción (la llamada información de pila). El resultado es el mismo mensaje que muestra el ejecutor (la máquina virtual de Java) cuando no se controla la excepción.

Métodos de la clase Exception.

```
try {
    String name = null;
    System.out.println("imprime el objeto" +name);
    // al invocar en un objeto nulo un método
    // origina una excepción
    System.out.println(name.length());
} catch (NullPointerException e) {
    System.out.println("Información del Método toString() " + e.toString());
    System.out.println("Informacion de la pila ");
    e.printStackTrace();
    System.out.println("Información del método getMessage() "+e.getMessage());
}
```

throws

- En la orientación a objetos toda acción la realiza un método.
- Si llamamos a un método que puede generar un error, ¿quién la manejará?
- ¿El propio método?
- ¿O el código que hizo la llamada al método?
- Con lo visto hasta ahora, sería el propio método quien se encargara de sus excepciones, pero esto complica el código ya que descentraliza el control de excepciones y dificulta el mantenimiento del código. Por eso otra posibilidad es hacer que la excepción la maneje el código que hizo la llamada.
- Esto se hace añadiendo la palabra throws tras la primera línea de un método. Tras esa palabra se indica qué excepciones puede provocar el código del método. Si ocurre una excepción en el método, el código abandona ese método y regresa al código desde el que se llamó al método. Allí se posará en el catch apropiado para esa excepción. Ejemplo:
-
- void usarArchivo (String archivo) throws IOException, InterruptedException {
- ...
- }

throws

- En este caso se está indicando que el método usarArchivo puede provocar excepciones del tipo IOException y InterruptedException. Lo cual obliga a que el código que invoque a este método deba preparar el (o los) catch correspondientes.
- Para utilizar el método:
- try{
- ...
- objeto.usarArchivo("C:\texto.txt");//puede haber excepción
- ..
- }
- catch(IOException ioe){
- ...
- }
- catch(InterruptedException ie){
- ...
- }
- ...

throw

- La instrucción throw nos permite provocar a nosotros una excepción (o lo que es lo mismo, crear artificialmente nosotros las excepciones). Ante:
- throw new Exception();
- El flujo del programa se dirigirá a la instrucción try…catch más cercana. Se pueden utilizar constructores en esta llamada (el formato de los constructores depende de la clase que se utilice):
- throw new Exception("Error grave, grave");
- Eso construye una excepción con el mensaje indicado.
- throw permite también relanzar excepciones. Esto significa que dentro de un catch podemos colocar una instrucción throw para lanzar la nueva excepción que será capturada por el catch correspondiente:

```
try{
        ...
} catch(ArrayIndexOutOfBoundsException exc){
        throw new IOException();
} catch(IOException){
        ...
}
```

- El segundo catch capturará también las excepciones del primer tipo.

throw

```
public static int introducir() throws Exception, NumberFormatException
{
    Scanner teclado =new Scanner(System.in);
    String texto;
    int numero=0;

    System.out.println("Introduce numero");
    texto=teclado.nextLine();
    numero=Integer.parseInt(texto);

    if (numero <0 || numero >10)
    {
        //llamada al constructor de la clase exception con un mensaje
        throw new Exception("Fuera de rango");
    }
    return numero;
}
```

throw

```
public static void main(String[] args) {
    int num=0;

    try{
        num=introducir();
    }catch(NumberFormatException e)
    {
        System.out.println("Error formato");
    }
    catch(Exception e){
        System.out.println(e.getMessage());
    }
    System.out.println("numero"+num);
}
```

finally

- La cláusula finally está pensada para limpiar el código en caso de excepción. Su uso es:
- try{
- ...
- }catch (FileNotFoundException fnfe){
- ...
- }catch(IOException ioe){
- ...
- }catch(Exception e){
- ...
- }finally{
- ...//Instrucciones de limpieza
- }
- Las sentencias finally se ejecutan tras haberse ejecutado el catch correspondiente. Si ningún catch capturó la excepción, entonces se ejecutarán esas sentencias antes de devolver el control al siguiente nivel o antes de romperse la ejecución.
- Hay que tener muy en cuenta que las sentencias finally se ejecutan independientemente de si hubo o no excepción. Es decir esas sentencias se ejecutan siempre, haya o no excepción. Son sentencias a ejecutarse en todo momento. Por ello se coloca en el bloque finally código común para todas las excepciones (y también para cuando no hay excepciones).

finally

```
public static void main(String[] args) {
    int numero;
    String linea;
    try {
        Scanner teclado = new Scanner(System.in);
        System.out.print("Teclea un numero entre 0 y 100: ");
        linea = teclado.next();
        numero = Integer.parseInt(linea);

    } catch (NumberFormatException e) {
        System.out.println("Debe introducir un numero entero");

    } finally {
        System.out.println("Ejecuto el finally antes de salir");
    }

}
```