1 Excepcións

1.1 Sumario

1.2 Excepcións e erros

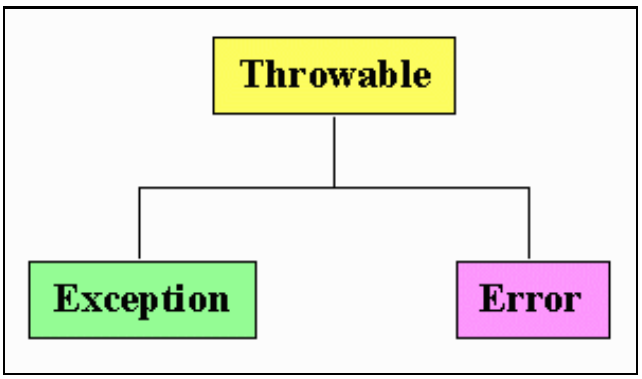
Unha **excepción** é unha situación que se orixina durante a execución do programa e que require que este acabe dunha maneira inmediata ou que faga unha acción especial para tratala. Por exemplo, un intento de división por 0; un acceso a unha posición dun array fora dos límites do mesmo ou un fallo durante a lectura de entrada/saída.

Java proporciona un mecanismo elegante e potente para tratar estas situacións que se poden producir nun programa. Cando se produce unha excepción no programa dicimos que se **lanza unha excepción** (*throw*)

O mecanismo de tratamento de excepcións pódese resumir no seguinte: transferir o control dende o lugar do programa no que aconteceu a excepción a un xestor (presente no programa) que a captura e a trata.

Ademais das excepcións, poden producirse erros. Un **erro** representa unha situación anormal e irreversible. Normalmente, son situacións que escapan ao control do programador e, polo tanto, non son tratadas por el.

Hai diferentes tipos de excepción. Cada un deles é unha subclase da clase **`Exception`**, mentres que os erros son subclases de `Error`. Ambas son subclases de `Throwable`



1.3 Tipos de excepcións

Se temos en conta como se debe tratar unha excepción dentro dun programa podemos distinguir dous tipos de excepcións:

- **Excepcións marcadas**. Son aquelas nas que é obrigatorio a súa captura. Todos os tipos de excpecións, salvo `RuntimeExcepction` e as súas subclases son excepcións marcadas. Se non se capturan o programa non compilará.
- **Excepcións non marcadas**. Non é obrigatoria a súa captura. Pertencen a este grupo as excepcións en tempo de execución, isto é, `RuntimeExcepction` e as súas subclases. Correspóndense normalmente cunha mala programación polo que a solución non debe pasar por preparar o programa para que se poda recuperar, senón evitar que se produzan. Por exemplo:

```
public class Division {
  public static void main (String [] args) {
    int j = 5/0; // División por cero. Dará un erro en tempo de execución.
  }
```

```
}
```

1.4 Captura de excepcións

No momento en que se produce unha excepción créase un obxecto do tipo de excepción correspondente e **lánzase** a excepción. Esta pode ser capturada premitindo realizar as accións oportunas. Por exemplo, se se produce unha excepción de tipo `IOException` se creará unha instancia desa clase.

1.4.1 Os bloques try e catch

Unha excepción en Java ten a seguinte forma:

```
try {
  // Sección de código que pode lanzar unha excepción
}
catch(unhaExecpcion) {
  // Sección de código que manexa esta excepción
}
catch(outraExecpcion) {
  // Sección de código que manexa esta execpción
}
// Outro código que pode executarse de xeito "normal"
```

O bloque try delimita aquela ou aquelas instruccións onde se pode producir unha excepción. Se isto sucede, o control do programa transfírese ao bloque catch definido para o tipo de excepción que se produciu, pasando como parámetro o obxecto do tipo da excepción que se creou.

O bloque catch define as instrucións que se deberán executar en caso de que se produza un determinado tipo de excepción.

Un exemplo en pseudocódigo dunha excepción é o seguinte:

```
try {
   descargaOFicheiroDendeARede
   leOFicheiroECreaUnhaTaboa
}
catch(NonPodoObterOFicheiroDaRede) {
   usaOFicheiroLocalNoSeuLugar
}
```

Non pode haber código entre un bloque try e outro catch. O seguinte exemplo fallará:

```
try {
   // Fai algo
}
System.out.println("Isto non se pode facer");
catch(Exception ex) { }
```

No caso de que existan varios bloques catch hai que ter en conta as seguintes consideracións:

- Pode haber varios bloques catch pero **non poden ter declarado a mesma clase de excepción**.
- Ainda que existan varios bloques catch **só se executará un deles** cando se produza a excepción.
- Os catch **máis específicos deben estar situados por diante dos máis xenéricos**, en relación á xerarquía de herdanza. Por exemplo:

  ```
  catch (IOException e)
  ```

  Ten que ir diante de:

  ```
  catch (Exception e)
  ```

Por último, o control do programa nunca se devolve ao lugar onde se produciu a excepción.

1.4.2 O bloque finally

As excepcións poden ter un terceiro bloque, non obrigatorio, chamado **finally**. O código que vai nun bloque finally **sempre** se executa (despois do bloque try), independentemente de que se lanzara ou non unha excepción. Incluso se hai unha sentenza `return` no bloque try! Por elo, sempre se pon neste bloque o código para pechar ficheiros, sockets e liberar recursos en xeral. Só se dentro dun bloque try ou catch se executa unha sentenza do tipo

`System.exit()` a cláusula finally non se executará.

Non podemos escribir un bloque try sen algún dos bloques, catch ou finally:

```
try {
  // fai algo
}
finally {
   //libera recursos
}
```

Outro exemplo cos tres bloques:

```
try {
   // fai algo
}
catch (UnhaExcepcion ex) {
  // trata a excepcion
}
finally {
   // libera recursos
}
```

1.5 Xerarquía de excepcións

Todas as excepcións son instancias dunha clase que ten á clase `Exception` como superclase da súa xerarquía. Noutras palabras, as excepcións son sempre subclases de `java.lang.Exception`. Cando se lanza unha excepción instánciase un obxecto da clase correspondente:

```
try {
       // algún código
}
catch (ArrayIndexOutOfBoundsException excepcion) {
       excepcion.printStackTrace();
}
```

Neste exemplo, `excepcion` é unha instancia da clase `ArrayIndexOutOfBoundsException`. Como tal obxecto pode chamar a métodos, neste caso a `printStackTrace()`. A xerarquía de excepcións en Java, non completa, é a seguinte:

```
java.lang.Exception
    * java.lang.ClassNotFoundException
    * java.lang.CloneNotSupportedException
    * java.lang.IllegalAccessException
    * java.lang.InstantiationException
    * java.lang.InterruptedException
    * java.lang.NoSuchFieldException
    * java.lang.NoSuchMethodException
    * java.lang.RuntimeException
          o java.lang.ArithmeticException
          o java.lang.ArrayStoreException
          o java.lang.ClassCastException
          o java.lang.EnumConstantNotPresentException
          o java.lang.IllegalArgumentException
                + java.lang.IllegalThreadStateException
                + java.lang.NumberFormatException
          o java.lang.IllegalMonitorStateException
          o java.lang.IllegalStateException
          o java.lang.IndexOutOfBoundsException
                + java.lang.ArrayIndexOutOfBoundsException
                + java.lang.StringIndexOutOfBoundsException
          o java.lang.NegativeArraySizeException
          o java.lang.NullPointerException
          o java.lang.SecurityException
          o java.lang.TypeNotPresentException
          o java.lang.UnsupportedOperationException
```

1.6 Matching de excepcións

Cando temos varios bloques catch se se lanza unha excepción esta irá comprobando se existe algún bloque catch polo que poder "entrar", empezando polo máis preto ao bloque try, que sempre será a excepción menos xenérica. O seguinte código fallará porque `IOException` é máis xenérica que `FileNotFoundException`:

```
try {
       // código
} catch (IOException e) {
      // manexa IOExceptions
} catch (FileNotFoundException ex) {
      // manexa só FileNotFoundException
}
```

O erro que devolverá o compilador será o seguinte:

```
TestEx.java:15: exception java.io.FileNotFoundException has
     already been caught
   } catch (FileNotFoundException ex) {
```

1.7 Lanzamento de excepcións

En determinados casos pode ser útil lanzar unha excepción dende dentro dun determinado método. Isto pode utilizarse como un medio para indicar que algo está sucedendo e non é posible continuar coa execución normal do método. Para lanzar unha excepción utilizamos a seguinte expresión:

```
throw obxecto_excepcion;
```

Cando se lanza unha excepción marcada desde un método esta **debe ser declarada na cabeceira do método**. Isto hai que indicalo coa palabra reservada `throw`, tal e como se amosa no seguinte exemplo:

```
void meuMetodo() throws Excepcion1, Excepcion2 {
   // código do método
}
```

A execución do método meuMetodo terá que ir nun bloque try se Excepcion1 e Excepcion2 son excepcións marcadas, xa que é obrigatorio capturalas. Se non, obteremos un erro en tempo de compilación.

Se a excepción é non marcada, é dicir, é unha subclase de `RuntimeExecption` non será necesario capturala.

1.8 Métodos útiles

Todas as excepcións herdan un conxunto de métodos que se poden usar no interior dos catch para completar o tratamento da excepción. Os máis útiles son:

```
/* Devolve unha mensaxe de texto asociada á excepción,
dependendo do tipo de excepción sobre o que se aplique.*/
void getMessage()

/*Envía á consola o volcado de pila asociado á excepción.
É moi útil durante a fase de desenvolvemento da aplicación,
axudando a detectar erros de programación.*/
void printStackTrace()
```

1.9 Creación de excepcións personalizadas

É posible crear as nosas propias excepcións xa que pode acontecer que os tipos de excepeción existentes non se adapten ás características da situación que se quere notificar. Para facelo simplemente hai que crear unha clase que herde da clase Exception cun construtor que chame ao construtor da superclase cun texto personalizado:

```
class unhaExcepcion extends Exception {
  public unhaExcepcion() {
    super("Excepcion personalizada!");
  }
}
```

1.10 Vantaxes de utilizar excepcións

Existen diferentes motivos polos que o uso de excepcións é recomendable:

- Proporcionan un **mecanismo estruturado** para o tratamento de erros, evitando a utilización de instrucións de control que dificultan a lectura do código e fan que o programa sexa máis propenso a erros.
- Proporcionan un xeito de agrupar diferentes tipos de erros nun programa.