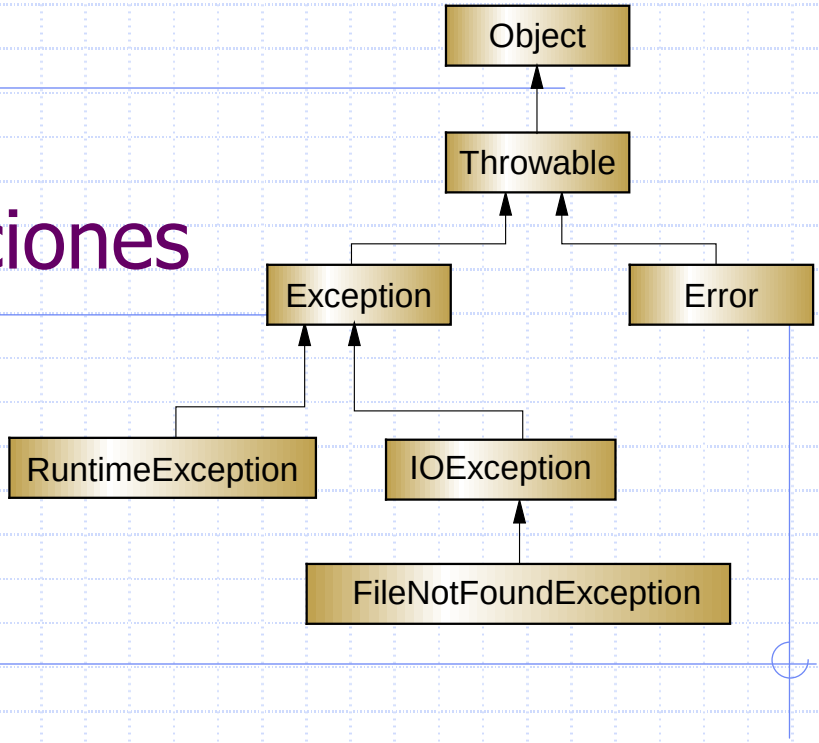
Excepciones



Excepciones

- Situaciones anómalas que aparecen durante la ejecución de un programa
  - Cuando aparece una condición excepcional se crea un objeto *Throwable* que se envía al método que la ha generado

Excepciones

- Su gestión permite la detección y corrección de errores en ejecución
  - Simplifican los programas ya que se diferencia el código normal del código de tratamiento de errores
  - Se crean programas mas robustos ya que en muchos casos si no se trata la excepción el programa no compila
  - Sólo se deben usar cuando no se puede resolver la situación anómala directamente en ese contexto
    - Se tiene que seguir haciendo el control de errores habitual

Dos tipos de situaciones excepcionales

- Excepciones
  - Situaciones más o menos habituales que impiden completar la ejecución correcta del código
  - Generalmente el programador debe proporcionar el código que las trate o gestione
  - Ejemplos
    - Error en el código o en los datos
    - Uso inadecuado de un métod

Ejemplo, la excepción ArrayIndexOutOfBoundsException no debería lanzarse nunca si los índices de acceso a un vector no se salen de los límites.

Dos tipos de situaciones excepcionales

- Errores
  - Representan situaciones de error normalmente no recuperables
  - El programador normalmente no tiene que proporcionar un tratamiento para ellas
  - Ejemplos
    - No se puede localizar y cargar una clase, Se agota la memoria

Tipos de excepciones

- Predefinidas en el sistema
  - Se lanzan automáticamente cuando se realiza alguna operación no valida
    - acceso a un objeto que no existe,
    - acceso a una posición de un array que no existe,
    - división por cero
- Generadas por el programador
  - El programa explícitamente genera una excepción al detectar una situación de error que no se puede resolver en ese contexto
  - Útil en situaciones de prueba y depuración

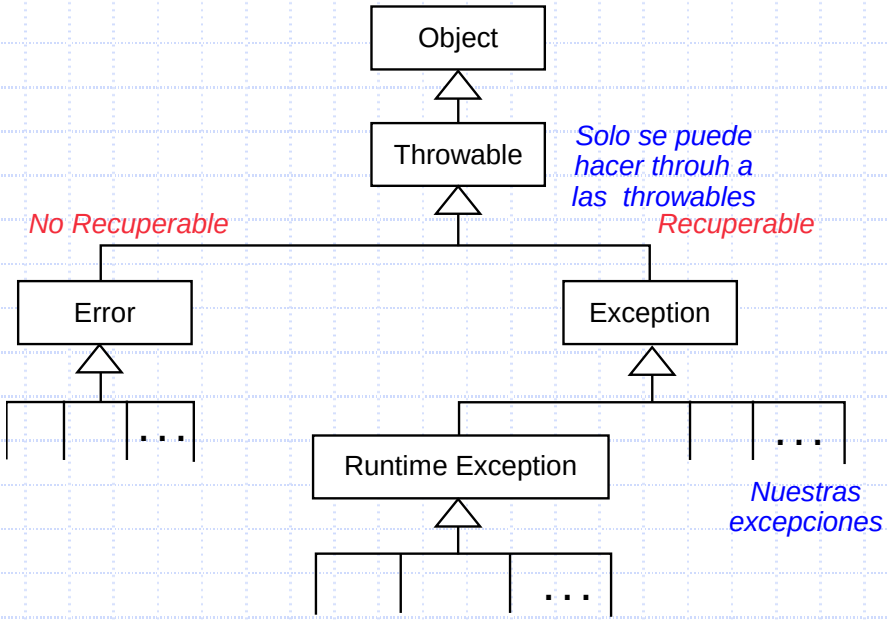
Ejemplo

```
public class HolaMundo {
 public static void main (String args[]){
 int i = 0;
 String  vectorS [] = {
 "Hola mundo1",
 "Hola mundo 2",
 "Hola mundo 3" };
 while (i < 4 ) {
 System.out.println(vectorS[i]);
 i++; }
 }
}
```


Ejemplo

```
c:\...\>java HolaMundo
HolaMundo 1
HolaMundo 2
HolaMundo 3
java,lang.ArrayIndexOutBoundsException : 3
 at HolaMundo.main(HolaMundo.java:12)
```

Tipos de excepciones



Gestión de excepciones

Para gestionar excepciones hay que insertar el código donde se puede dar dentro de un bloque try.

Se crea una lista de bloques catch adyacentes, uno por cada posible excepción.

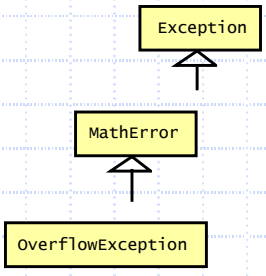
Gestión de excepciones

- Sintaxis

```
try {
    // bloque de código donde puede producirse una excepción
}catch( TipoExcepción1 e ) {
    // gestor de excepciones para TipoExcepción1
    // se ejecuta si se produce una excepción de tipo TipoExcepció
}catch( TipoExcepcion2 e ) {
 // gestor de excepciones para TipoExcepción2
  throw(e);  // se puede volver a lanzar la excepción propagar
} finally {
  // bloque de código que se ejecuta siempre, haya o no excepción
}
```


Gestión de excepciones

```
try { /*...*/}
catch (OverflowException e) { /*...*/}
catch (MathError e) { /*...*/}
catch (Exception e) { /*...*/}
finally{ /*...*/}
```



Gestión de excepciones

- finally
  - Es el bloque de código que se ejecuta siempre, haya o no excepción

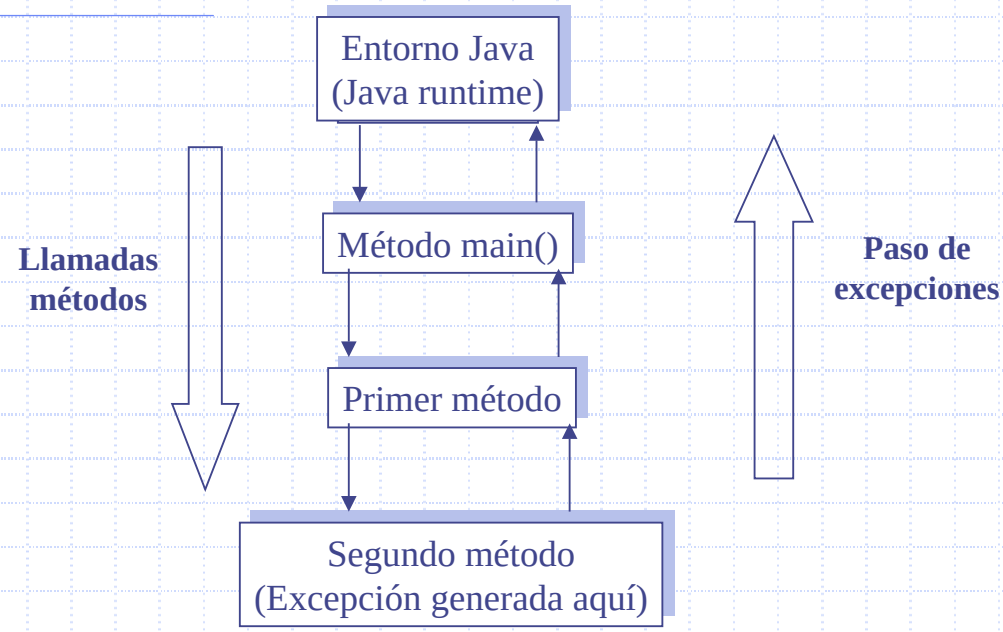
```
try {
        inicioProceso ();
        gestionProceso();
}
finally {
        finProceso ();
}
```

  - La única situación en la que la sentencia finally no se ejecuta es si se llama al método System.exit (), que termina la ejecución del programa.

Gestión de excepciones

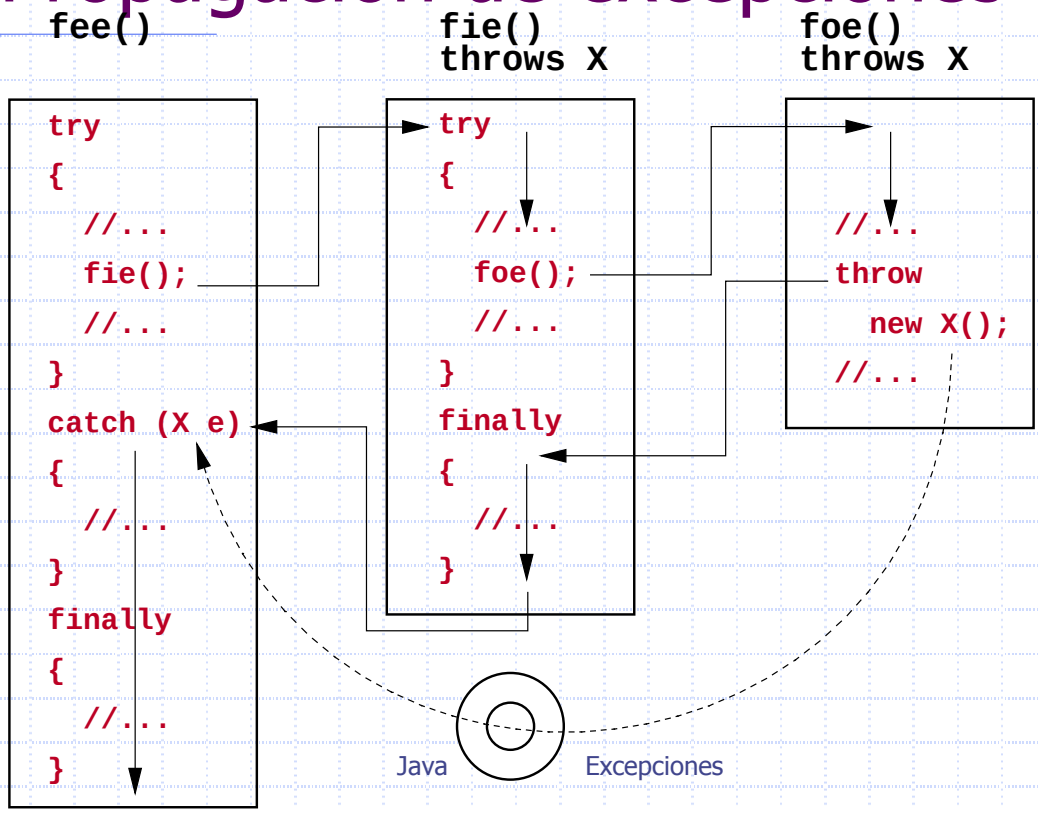
- throw
  - Permite lanzar de forma explícita una excepción
  - Se puede utilizar para propagar una excepción tratada parcialmente

Propagación de excepciones



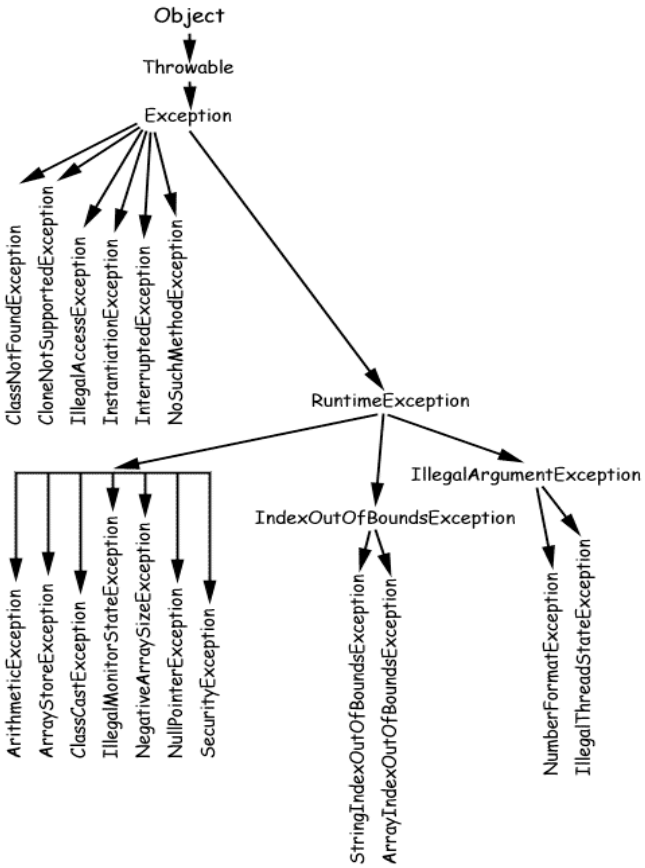
Todas las excepciones que se puedan generar se han de manejar o delegar

Propagación de excepciones



Excepciones

- **RuntimeExceptions**
  - ArrayIndexOutOfBoundsException
  - ArithmeticException
  - NullPointerException
  - ClassCastException
  - IllegalArgumentException
- **No es obligatorio incluir código de tratamiento de este tipo de excepciones**
  - **aunque se puede incluir dicho código**



Ejemplo simple excepciones

```
public class PruebaExcepciones {

 public static void main(String args[]) {
  int valor=5, cero=0;
  int[] array = {1, 2, 3};
  try {
       valor = valor/cero;  //división por cero
       array[4]= 5; //acceso a una posición no disponible
        }
  catch( ArithmeticException e ) {
       System.out.println( "Division por cero" );
       }
  catch( Exception e ) {
        System.out.println( "Se ha producido un error" );
       }
       }
}
```

Propagación de excepciones

```
public class PruebaExcepcion {
 public static void main(String args[]) {
  int valor=5, cero=0;
  int[] array = {1, 2, 3};
  try {
       try {
       valor = valor/cero;  //división por cero
       array[4]= 5; //acceso a una posición no disponible
       }catch( ArithmeticException e ) {
       System.out.println( "Division por cero" );
       throw e;
       }catch( Exception e ) {
       System.out.println( "Se ha producido un error" );
       }
       }catch(Exception e){
       System.out.println(e.getMessage());
       }
       }
}
```

throws

Indica que el código producirá una excepción, que no se tratará dentro de él y se pasará al método superior, utilizando la cláusula throws.

```
public void ejemploExcep () throws
IOException
```

A continuación de la palabra reservada throws aparece una lista de todas las excepciones que se pueden dar dentro del método y no serán gestionadas.

Gestión incompleta de excepciones

TipoDevuelto nombreMetodo(argumentos) ***throws*** *listaExcepciones*
{ / cuerpo del método */ }*

- Si un método no gestiona o captura todas las excepciones que puede generar (excepto del tipo *Error* o *RuntimeException*) debe especificarlo mediante *throws*

TipoDevuelto nombreMetodo(argumentos) ***throws*** *listaExcepciones*
{ / cuerpo del método */ }*

Gestión incompleta de excepciones

```
import java.io.*;
public class PruebaExcepciones {
       public static char leer() throws IOException {
              return (char) System.in.read();
       }
       public static void main(String args[])  {
       try {
       char car=leer();
       System.out.println("Caracter: "+car);
       } catch  (IOException e) {
       System.out.println("Error de entrada de datos");
        }
}
```

Lanzamiento de excepciones

```
public class Alumno{
      public Alumno(String nombre, String apellidos)
throws Exception{
      if(nombre == null || apellidos == null)
      throw new Exception("Argumentos no válidos");
      //si el constructor lanza la excepción el objeto
      //no se crea
      }
      public static void main(String args[]) {
            try{
            Alumno alum = new Alumno(null, "hola");
            }catch (Exception e){
            System.out.println("Excepcion: "+
                                    e.getMessage());
            }
      }
}
```

Excepciones definidas por el programador

- El programador puede definir sus propias clases de excepciones
  - Se define una clase que herede de *Throwable* o más normalmente de *Exception*

```
public class EdadFueraDeRangoException extends Exception {
      public EdadFueraDeRangoException (String texto) {
            super(texto);
      }
}
```

Excepciones definidas por el usuario

```
public class FileInputStream
                    extends InputStream{
   public FileInputStream(String aFileName)
                    throws IOException   {
      if (...) {
         throw new IOException("No Such
   File");
       }
        ...
    }
    ...
}
```


Excepciones definidas por el usuario

- Gestiona la excepción, incluyendo en el código los bloques try-catch-finally
- Se considera que una excepción está tratada incluso si el bloque catch está vacío.
- Indica que el código producirá una excepción, que no se tratará dentro de él y se pasará al método superior, utilizando la cláusula throws.

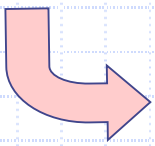
Excepciones definidas por el usuario

- public void ejemploExcep () throws IOException
- A continuación de la palabra reservada throws aparece una lista de todas las excepciones que se pueden dar dentro del método y no serán gestionadas.

Ejemplo excepción definida por el usuario

```
Public class Persona{
      int edad;
.....
public void ponEdad(int ed) throws EdadFueraDeRangoException  {
    if ((ed < 0) || (ed > 130))
        throw new EdadFueraDeRangoException("Demasiado joven o demasiado viejo");
    edad = ed;
}}
```

```
try {
      alguien.ponEdad(150);
} catch (EdadFueraDeRangoException e){
              System.out.println("se ha producido la excepción");
              e.printStackTrace();
              System.out.println(e.getMessage());
}
```



```
EdadFueraDeRangoException: Demasiado joven o demasiado viejo
              at Persona.ponEdad<Persona.java>
              at Persona.main<Persona.java>
```

Ejemplo

```
public void connectMe (String serverName)
  throws ServerTimedOutException {
   int success;
   int portToConnect = 80;
   success = open (serverName,
  portToConnect);
   if (success == -1) {
      throw new
  ServerTimedOutException("Imposible
  conectarse", 80);
   }
}
```


Ejemplo

```
public void findServer(){
    ...
    try {
      connectMe(defaultServer);
} catch (ServerTimedOutException e) {
     System.out.println ("Server timed out,
                         trying alternate");
    try {
    connectMe(alternateServer);
    } catch (ServerTimedOutException e1) {
       System.out.println("No server
       currently available");}
}....
```

Excepciones más frecuentes

ArithmeticException

int i = 12 / 0;

NullPointerException

Date d = null;

System.out.println (d.toString());

NegativeArraySizeException

Intento de creación de un vector con un número negativo de elementos

Excepciones más frecuentes

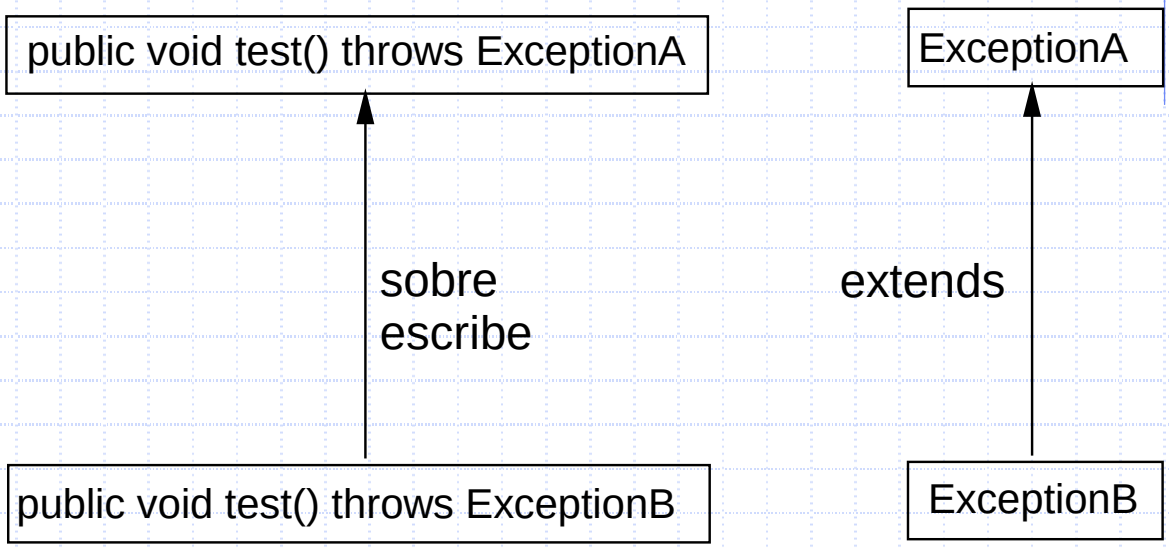
ArrayIndexOutOfSizeException

Intento de acceso a un elemento de un vector fuera de rango

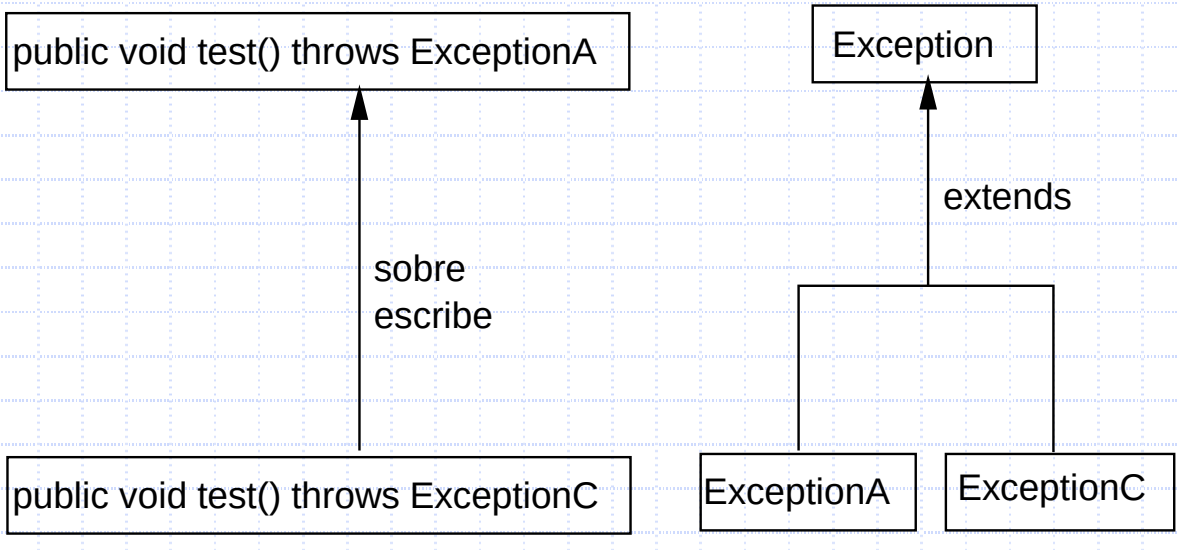
SecurityException

Error de seguridad en los navegadores.

Sobreescritura de métodos



Sobreescritura de métodos



No compila

Ejercicio 1: Exception

Un banco contiene las Cuentas de sus clientes. Las CuentasDeAhorro no pueden tener números rojos. Las CuentasCorrientes pueden tener una CuentaDeAhorro asociada, de forma que si se intenta retirar más dinero del saldo actual, se debe retirar el dinero que falte de la CuentaDeAhorro asociada.

1) Define Cuenta de forma que no pueda instanciarse. De toda Cuenta se debe poder ingresar y retirar dinero, preguntar por el saldo, por el DNI del titular y debe tener un método toString de devuelva al menos el saldo y el DNI del titular.

Ejercicio

2) Implementa las clases CuentaCorriente y CuentaDeAhorro.
3) Crea una especialización CuentaDeAhorroEsp de CuentaDeAhorro en la que se añade un entero penalización, de forma que se penaliza la retirada con un penalización % del dinero retirado. Sobrescribe sólo los métodos necesarios (incluyendo constructor y toString).
4) Si el saldo de CuentaDeAhorro fuese a quedar negativo, antes de debe lanzar una excepción SaldoNegativo (que hereda de Exception).

Ejercicio

5) Implementa la clase Banco que contiene un array polimórfico de Cuentas, incluyendo el constructor que consideres más apropiado.

6) Incluye el método totalSaldoMaxPenalización en la clase Banco que devuelva la suma de los saldos de todas las cuentas corrientes y la máxima penalización entre las CuentaDeAhorroEsp .

Crea una clase con método main en la que instanciamos un Banco de 100 Cuentas, nos creamos una CuentaCorriente con 500 euros y una cuenta de ahorrro especial con una penalización del 5%. Finalmente debe mostrar información de todas las Cuentas del Banco. En un bloque try & catch retirar dinero de una cuenta de Ahorro.