UT06. EXCEPCIONES EN JAVA

Programación de 1DAW

C.I.F.P. Carlos III - Cartagena

Introducción

- **Excepción**
  - Cuando sucede un **evento anormal** en la ejecución de un programa y lo detiene.
  - Construimos programas robustos.
  - Es un objeto que contienen información del error que se ha producido y que heredan de la **clase *Throwable*** o de la **clase *Exception*.**
  - Si no se captura, interviene un manejador por defecto que visualiza un mensaje y detiene la aplicación.
  - Categorías.
    - **Excepciones verificadas**: lanzadas por objetos del usuario
    - **Excepciones no verificadas**: el compilador no obliga a su verificación.

Excepción

- **Una excepción es un error semántico que se produce en tiempo de ejecución.**
  - Puede ser que el código sea válido sintácticamente (compila), pero pueden producirse errores inesperados como puede ser:
    - Dividir por cero.
    - Intentar acceder a una posición de un array fuera de sus limites.
    - Al llamar al **nextInt()** de un **Scanner** el usuario no introduce un valor entero.
    - Intentar acceder a un fichero que no existe o que esta en un disco duro corrupto.
- Un método es capaz de tratar una excepción si se ha previsto en el código, se captura y se trata
- Cuando se lanza una excepción,
  - la máquina virtual de Java recorre la pila de llamadas en busca de un método capaz de tratar la clase de excepción que se ha lanzado
  - Examina el método que ha producido la excepción para ver si es capaz de tratarla.
  - Si no es capaz, examina el método desde el que se realizó la llamada al método donde se produjo la excepción
  - Y así sucesivamente hasta llegar al último
  - Si ninguno trata la excepción, la máquina virtual muestra un mensaje de error y el programa termina

Ejemplos

```java
public class Excepcion1 {

    /**...3 lines */
    public static void main(String[] args) {
        int num1, num2, result;

        num1=4;
        num2=0;
        result=num1/num2;
        System.out.println("División: "+result);
    }
}
```

```java
public class Excepcion2 {

    /**...3 lines */
    public static void main(String[] args) {
        Scanner teclado=new Scanner(System.in);
        int num;

        System.out.print("Dame un número: ");
        num=teclado.nextInt();

    }

}
```

```java
public class Excepcion3 {

    /**...3 lines */
    public static void main(String[] args) {
       int []numeros={1,2,3,4};

        System.out.println("acceso a un elemento que no existe");
        numeros[8]=9;
    }

}
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at solucionut06.Excepcion1.main(Excepcion1.java:22)
C:\Users\mcruz\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 1 second)
```

```
Dame un número: t
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:864)
        at java.util.Scanner.next(Scanner.java:1485)
        at java.util.Scanner.nextInt(Scanner.java:2117)
        at java.util.Scanner.nextInt(Scanner.java:2076)
        at solucionut06.Excepcion2.main(Excepcion2.java:24)
C:\Users\mcruz\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 6 seconds)
```

```
acceso a un elemento que no existe
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 8
        at solucionut06.Excepcion3.main(Excepcion3.java:21)
C:\Users\mcruz\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 1 second)
```

Excepción

- En los tres casos, Java finaliza el programa en la que se ha producido la excepción y muestra el mensaje correspondiente
- Por qué lanzar
  - Necesidad de controlar un suceso incorrecto o inesperado: valores incorrectos
- Cómo lanzar
  - Para lanzar una excepción: **throw**

```
throw new Exception();
```

```
Exception e=new Exception();
throw e;
```

Son equivalentes

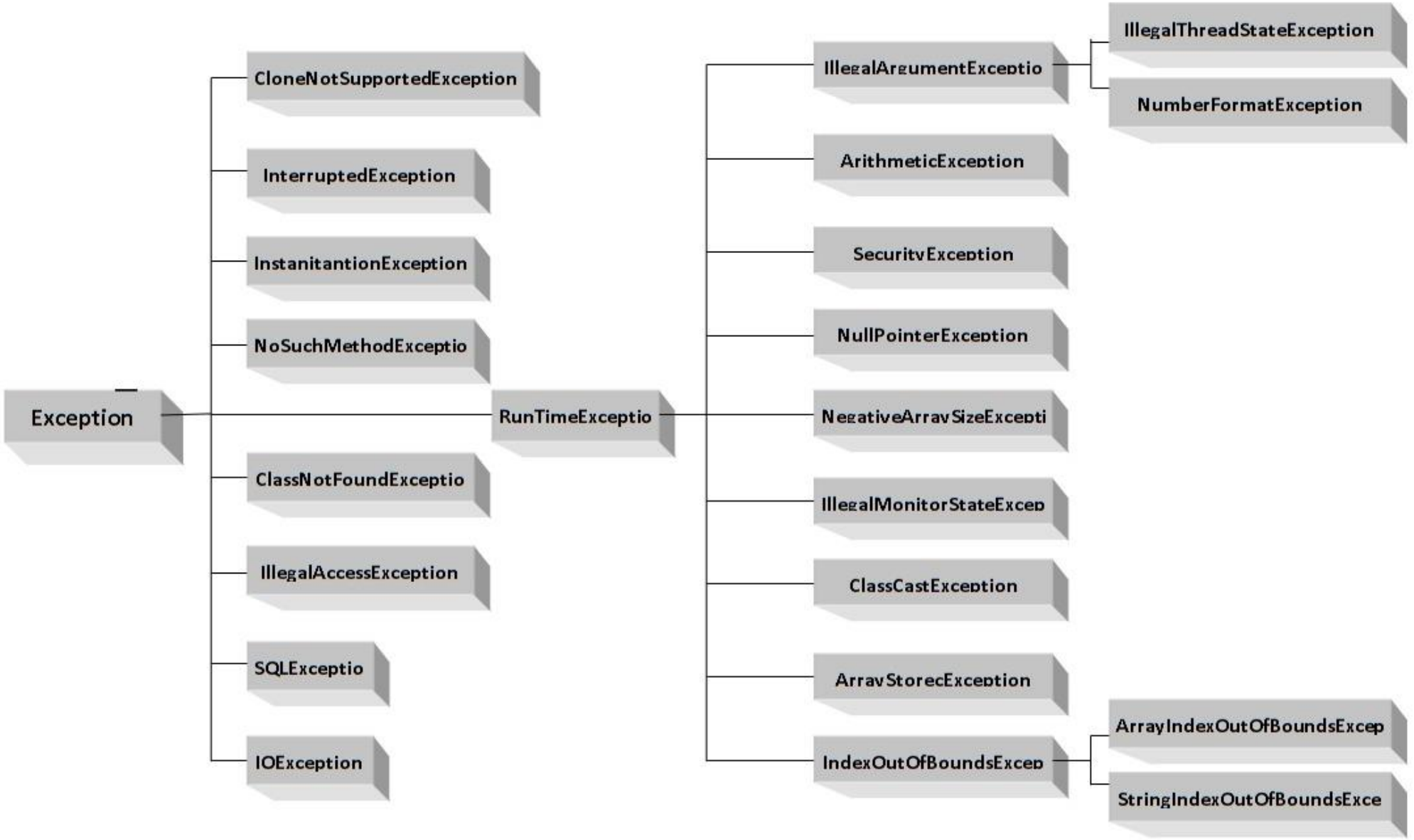
Excepción

- Podría añadir un mensaje como argumento

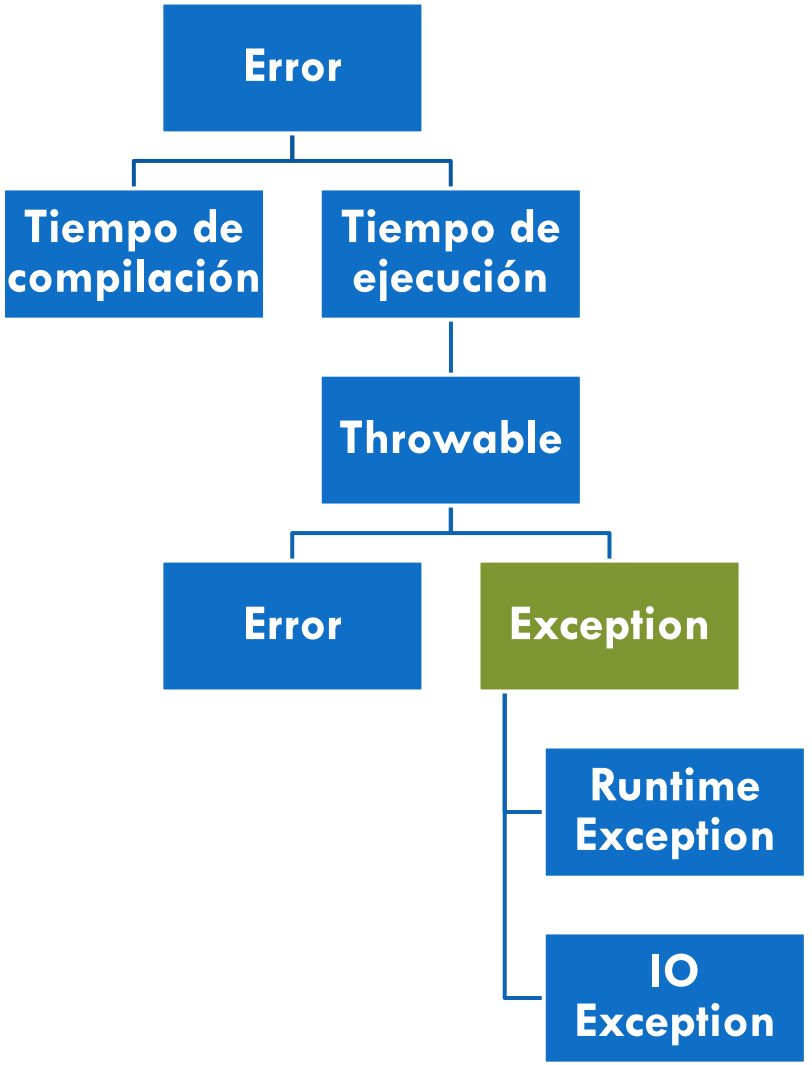
```
throw new Exception("Error en la entrada de datos" );
```

- Se puede ser más explícito en la excepción: ArrayIndexOutOfBoundsException, ArithmeticException, NumberFormatException, o crear una propia
- Al lanzar una excepción se parará la ejecución de dicho método (no se ejecutará el resto del código del método) y se lanzará la excepción al método que lo llamó.
- Un método puede lanzar distintos tipos de excepciones (si lo consideramos necesario). En tal caso hay que especificar todos los tipos posibles en la cabecera, separados por comas. Por ejemplo, imaginemos que el constructor de Persona toma como argumentos el dni y la edad, y queremos lanzar excepciones distintas según cada caso.

Jerarquía de Excepciones



Jerarquía de Excepciones



Error

- Excepción que se crea cuando ha ocurrido un problema serio.
- Involucra a la máquina virtual de Java, no al código.
- Una aplicación normal no puede manejar este tipo de excepción.
- Excepción implícita.

Exception

- Cubre las excepciones que una aplicación normal puede manipular.
- Error en un método → se crea un **objeto 'exception'** con información sobre el tipo de excepción y el estado del programa y la ejecución del bloque de código correspondiente (manejo) → **lanzar una excepción**

Manejar excepciones

- Anomalía en método.
  - Lanzar (**throw**) excepción en método
  - Atrapar (**catch**) quien lo llamó y la maneja.
  - Si no se atrapa, finaliza el programa.

```
try {
      <código que puede ocasionar una excepción>
} catch <tipo_excepción1 objeto_excepción>{
      <código para manejar la excepción>
} catch <tipo_excepción2 objeto_excepción>{
      <código para manejar la excepción>
}
```

Manejar excepciones

- Si se **lanza** una excepción, es atrapada por su correspondiente **catch**.
- Un único **try**, posibilidad de **varios catch**.
- El flujo del programa continúa después del bloque catch.

```
public class Excepcion1{
    public static void main(String[] args){
        int [] vector = new int[5];

        vector[5]=0;
    }
}
```

Genera una excepción **ArrayIndexOutOfBoundsException**

```
run:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
        at excepcion1.Excepcion1.main(Excepcion1.java:21)
Java Result: 1
BUILD SUCCESSFUL (total time: 1 second)
```

Manejar excepciones

```
public class Excepcion2{
    public static void main(String[] args){
        int [] vector = new int[5];

        try{
            vector[5]=0;
        }catch (ArrayIndexOutOfBoundsException e){
            System.out.println("Excepción generada por acceso no permitido al vector");
        }
    }
}
```

```
run:
Excepción generada por acceso no permitido al vector
BUILD SUCCESSFUL (total time: 0 seconds)
```

Manejar una excepción

- **Excepción.** Puede ser generada por un método y atrapado por el método que llamó al primero

```
public class GeneraExcepcion2{
  public void produceExcepcion(){
    int numero[]=new int[5];
    System.out.println("Tratando de acceder a una posición fuera del vector");
    numero[7]=0;
   }
}
```

Genera una excepción
ArrayIndexOutOfBoundsException

Atrapa una excepción
ArrayIndexOutOfBoundsException

```
public class PruebaGeneraExcepcion2{
    public static void main (String args[]){
        GeneraExcepcion2 objeto =new GeneraExcepcion2();
        try{
            objeto.produceExcepcion();
        } catch (ArrayIndexOutOfBoundsException excep){
            System.out.println("Excepción Generada en otro bloque");
        }
    }
}
```

Ejemplo varios bloques catch

De la más particular a la más general

```
try{
    // …
} catch (EOFException e) {
    //Manejar esta clase de excepción
} catch (IOException e) {
    //Manejar esta clase de excepción o de alguna de
    //sus subclases, excepto EOFException
} catch (Exception e) {
    //Manejar esta clase de excepción o de alguna de
    //sus subclases, excepto EOFException e
    //IOException
}
```

```
publicclass ExcepcionesMultiples{
      public static void divide(){
          int num[]={4,8,16,32,64,128,256};
          int den[]={2,0,4,4,0,8};

          for (int i=0;i<num.length+1;i++){
            try{
              System.out.println(num[i]+ "/" + "=" + num[i]/den[i]);
            }catch(java.lang.ArithmeticException excepcion){
              System.out.println("Dividiendo por cero");
            } catch(java.lang.ArrayIndexOutOfBoundsException excepcion){
              System.out.println("Error al acceder el vector");
            }
          } // for
       } //divide
}
public class PruebaExcepcionesMultiples{
       public static void main (String args[]){
        ExcepcionesMultiples.divide();
       }
}
```

```
class ExcepcionesAnidadas {
  public static void main(String args[]){
    int num[]= {4,8,16,32,64,128,256,512};
    int den[]= {2,0,4,4,0,8};
    try{
      for (int i=0; i<num.length; i++){
          try{
              System.out.println(num[i]+"/"+den[i]+"="+ num[i]/den[i]);
          }catch (ArithmeticException Excep){
              System.out.println("Fuera de limite"+ i);
          }
        }
    }catch (Throwable Excep){ //try externo
        System.out.println("Ocurrió una excepción fatal");
    }
    System.out.println("El programa puede continuar aquí");
  } // fin de main
}
```

Bloque de finalización

- Realización de acción por obligación ➔ **finally**.
- Siempre se ejecuta.

```
try{
    código que produce la(s)
      excepción(es)...
}catch (TipoDeExcepcion objeto){
      // Código para manejar la
      excepción
}finally{
       // código de finally
}
```

```
class UsoFinally{
  public static void generaExcepcion(int i){
     int t;
     int num[] = {2,4,6};
     System.out.println("Recibiendo ");
     try{
        switch(i){
          case 0: t=10/i; break;          //div por cero
          case 1: num[4]=4; break;    //genera un error
          }
     } catch(ArithmeticException exc){
       System.out.println("No puede dividir entre cero");
     } catch(ArrayIndexOutOfBoundsException exc)  {
        System.out.println("No hay elementos que coincidan");
     } finally {
       System.out.println("Ejecutando código de limpieza");
     }
   }  //fin de método
 }//clase
```

```
class PruebaUseFinally {
    public static void main(String args[]) {
        for (int i=0;i<4; i++) {
            UsoFinally.generaExcepcion(i);
            System.out.println();
        }
    }
}
```

Declarar excepciones. throws

- Java requiere que cualquier método que pueda lanzar una excepción la declare o la atrape.
- **throws** permite a un método declarar la lista de excepciones (separadas por comas).
- Si no atrapa el método las excepciones, las atrapará cualquier método que invoque el método que la ha lanzado.
- **Delegación de excepciones**: cuando un método utiliza una sentencia que puede generar una excepción pero la excepción que se puede generar no la captura y la trata, sino que la delega a quién la llamó → **throws** en la cabecera del método
- Las excepciones que derivan de **Error** o **RuntimeException** no necesitan ser especificados en **una lista throws.**
- Si un método lanza explícitamente una instancia de **Exception** o de sus subclases, se debe declarar su tipo con la sentencia **throws.**

```
valorRetorno nombreMetodo (parámetros) throws  excepción1…, excepción {
    //código del método
}
```

Crear y lanzar excepciones

- Creación de nuestras propias excepciones.
- Ejemplo:

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

- Superclase de **EValorNoValido** ➔ **Exception**
- Implementa 2 constructores
- El flujo de la ejecución se detiene inmediatamente después de la sentencia **throw,** y nunca se llega a la sentencia siguiente

Ejemplo

```
public Pelicula(String titulo, String autor, Formato formato, int duracion,
                        String actorPrin, String actrizPrin){
    super(titulo, autor, formato, duracion);
    if(actorPrin==null && actrizPrin==null)
        throw new IllegalArgumentException("Tiene que haber al menor un "
                + "actor o una actriz");
    this.actorPrin=actorPrin;
    this.actrizPrin=actrizPrin;
}
```

Crear y lanzar excepciones

```
public class AñoFueraDeRangoException extends Exception{
        public AñoFueraDeRangoException () { }

        public AñoFueraDeRangoException (String texto) {
                super(texto);
        }
}
```

```
public class Alumno{
   public void ponAñoDeNacimiento (int año)  throws AñoFueraDeRangoException {
        if (año<1900 || año > 1990)
          throw new AñoFueraDeRangoException ("Demasiado joven o demasiado viejo");
        añoDeNacimiento=año;
   }
}
```

Crear y lanzar excepciones

- Otra opción es que el método que lanza la excepción la atrape (pero es anticiparse a las necesidades que puede tener el usuario)

```
public Pelicula(String titulo, String autor, Formato formato, int duracion, String
actorPrin, String actrizPrin){
        super(titulo, autor, formato, duracion);
        if (actorPrin==null && actrizPrin==null)
            throw new IllegalArgumentException("Tiene que haber al menos un "
                    + "actor o una actriz");
        this.actorPrin=actorPrin;
        this.actrizPrin=actrizPrin;
    }
}
```

throws y throw

- **throws**
  - Sirve para indicar que, en caso de que se produzca una excepción, el método en el que se produce la misma, no la manejará, sino que la excepción será manejada por un método invocante o superior.
  - Por esta razón es necesario y obligatorio que se use en conjunto con **throw**, ya que si el método en el que se produce la excepción no tratará la misma, se la debe lanzar (**throw**) a un método superior.
  - En el ejemplo, si en el constructor de la **clase Alumno** no se permite que la edad de un alumno sea ni cero ni negativa, usaremos **throws** para indicar que no se manejará dicha excepción en la creación del **objeto Alumno**, sino en un método superior:

```
public class Alumno {
    private String nombre;
    private String apellido;
    private int edad;

    public Alumno(String nombre, String apellido, Integer edad)
throws Exception{
        this.nombre=nombre;
        this.apellido=apellido;
        if(edad<=0){
            throw new Exception("La edad debe ser mayor que 0");
        }else{
            this.edad=edad;
        }
    }

    public String getNombre() {
        return nombre;
    }

 public String getApellido() {
        return apellido;
    }

    public int getEdad() {
        return edad;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public void setApellido(String apellido) {
        this.apellido = apellido;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }
}
```

```
public static void main(String[] args) {
        try{
            Alumno alu1=new Alumno("Marcos","Fernández", -2);
        }catch(Exception e){
            System.out.println(e.getMessage());
        }
    }
```

Crear excepciones

- Crear una clase que maneje la excepción del set de una edad: **YearInvalidExcepcion.**
- Para crear nuestras propias excepciones
  - La clase debe heredar (**extends**) de la clase padre de todas las excepciones **Exception.**

```
public class AlumnoII {
    private String nombre;
    private String apellido;
    private int edad;

    public AlumnoII(String nombre, String apellido, Integer edad) throws
InvalidYearException {
        this.nombre=nombre;
        this.apellido=apellido;
        if(edad<=0){
            throw new InvalidYearException ("La edad debe ser mayor que 0");
        }else{
            this.edad=edad;
        }
    }

    public String getNombre() {
        return nombre;
    }

 public String getApellido() {
        return apellido;
    }

    public int getEdad() {
        return edad;
    }
```

```
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public void setApellido(String apellido) {
        this.apellido = apellido;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }
}
```

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

```
public static void main(String []args){
        try {
            Alumnoll alu1 = new Alumnoll("Marcos", "Fernandez", -2);
            System.out.println(alu1.getNombre());
        } catch (InvalidYearException e) {
            System.out.println("Se atrapo una excepción del tipo InvalidYearException");
            System.out.println(e.getMessage());
        }
    }
```