17. Programación Orientada a Objetos(III)

17.1 Excepciones.

Los errores que se producen en tiempo de ejecución devuelven excepciones. El programa compila y se puede ejecutar pero, por algún motivo, se produce un fallo y el programa no puede continuar su normal ejecución. Un programador debe prever esta situación. A continuación se muestra un programa que calcula la media de dos números.

```
double numero1;
double numero2;
Scanner s = new Scanner(System.in);
System.out.println("Media de dos números");
System.out.print("Introduzca el primer número: ");
numero1 = Double.parseDouble(s.nextLine());
System.out.print("Introduzca el segundo número: ");
numero2 = Double.parseDouble(s.nextLine());
System.out.println("La media es " + (numero1 + numero2) / 2);
```

Si introducimos un carácter no numérico en la entrada se produce lo siguiente:

```
Introduzca el primer número: puertas
Exception in thread "main" java.lang.NumberFormatException: For input string: "puertas"
        at java.base/jdk.internal.math.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:2054)
        at java.base/jdk.internal.math.FloatingDecimal.parseDouble(FloatingDecimal.java:110)
        at java.base/java.lang.Double.parseDouble(Double.java:556)
        at trycatch.Tryca.main(Tryca.java:14)
```

17.2 Try/catch/finally.

El bloque ***try - catch - finally*** sirve para encauzar el flujo del programa de tal forma que, si se produce una excepción, no se termine de forma drástica y se pueda reconducir el ejecución de una manera controlada.

El formato de este bloque es el siguiente:

```
try {
      Instrucciones que se pretenden ejecutar
      (si se produce una excepción puede que no se ejecuten todas ellas).
} catch {
      Instrucciones que se van a ejecutar cuando se produce una excepción.
} finally {
      Instrucciones que se van ejecutar tanto si se producen excepciones como
si no.

}
```

Se pueden especificar varios catch para controlar diferentes excepciones como veremos más adelante. La parte finally es opcional.

Siguiendo con el programa que calcula la media de dos números, vamos a introducir un bloque try - catch - finally para que el programa termine de forma controlada si se produce una excepción.

```
double numero1;
double numero2;
Scanner s = new Scanner(System.in);
System.out.println("Calcula la media de dos números");
try {
    System.out.print("Introduzca el primer número: ");
    numero1 = Double.parseDouble(s.nextLine());
    System.out.print("Introduzca el segundo número: ");
    numero2 = Double.parseDouble(s.nextLine());
    System.out.println("La media es " + (numero1 + numero2) / 2);
} catch (NumberFormatException e) {
    System.out.print("No se puede calcular la media. ");
    System.out.println("Los datos introducidos no son correctos.");
} finally {
    System.out.println("Fin de ejecución");
}
```

Se puede mostrar tanto el tipo de excepción como el error exacto que se produce. Para ello, se aplican los métodos getClass() y getMessage() respectivamente al objeto e. El tipo de excepción viene dado por el nombre de una clase que es subclase de Exception. Bastará con añadir las siguientes líneas al ejemplo anterior.

```
System.out.println("Excepción: " + e.getClass());

System.out.println("Error: " + e.getMessage());
```

Vamos a refinar un poco más el programa. Si se produce una excepción al introducir un dato, el programa volverá a pedirlo una y otra vez hasta que el dato sea correcto.

```
static public double pideDoubleTeclado(String mensaje) {
    double numero = 0;
    boolean haFallado = false;

    do {
        System.out.print(mensaje);
        Scanner s = new Scanner(System.in);
        try {

            if (s.hasNextLine()) {
                numero = Double.parseDouble(s.nextLine());
            }
            haFallado = false;
        } catch(NumberFormatException e) {
            System.out.println("Valor incorrecto.");
            haFallado = true;
        }

    } while(haFallado);

    return numero;

}



static public void main(String args[]) {

    double numero1;
    double numero2;

    System.out.println("Calcula la media de dos números");
    numero1 = pideDoubleTeclado("Introduzca el primer número: ");
    numero2 = pideDoubleTeclado("Introduzca el segundo número: ");

    System.out.println("La media es " + (numero1 + numero2) / 2);


}
```

```
Calcula la media de dos números
Introduzca el primer número: a
Valor incorrecto.
Introduzca el primer número: 3
Introduzca el segundo número: b
Valor incorrecto.
Introduzca el segundo número: 5
La media es 4.0
```

17.3 Lanzar excepciones.

La orden throw permite lanzar de forma explícita una excepción. Por ejemplo, la sentencia *throw new ArithmeticException()* crea de forma artificial una excepción igual que si existiera una línea como *System.out.println(1 / 0);*.

Como throw permite lanzar de forma explícita una excepción, nos servirá para lanzar excepciones propias como veremos más adelante. También es útil cuando se recoge la excepción en un método y luego, esa misma excepción se vuelve a lanzar para que la recoja, a su vez, otro método y luego otro y así sucesivamente hasta llegar al main.

En el constructor de un cuadrado nos puede interesar lanzar una excepción cuando tenga un lado cero o negativo.

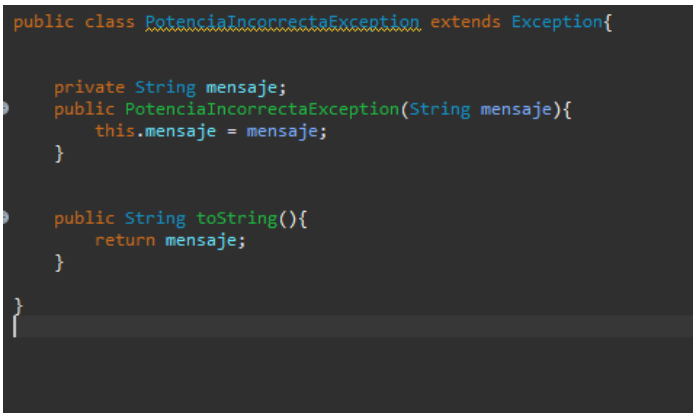
```
private double  lado;

public Cuadrado(double lado) {

    if (lado<=0) {
        throw new ArithmeticException("Error");
    }else {
        this.lado = lado;
    }
}
```

17.4 Crear excepciones y throws.

Java nos permite crear excepciones propias, hechas a medida. Para ello, no hay más que utilizar una de las características más importantes de la programación orientada a objetos: la herencia. Crear una nueva excepción será tan sencillo como implementar una subclase de Exception.



```
public class PotenciaIncorrectaException extends Exception{


    private String mensaje;
    public PotenciaIncorrectaException(String mensaje){
        this.mensaje = mensaje;
    }


    public String toString(){
        return mensaje;
    }

}
```

Para advertir de que un método puede lanzar una excepción utilizamos throws, lo que obligará a quien lo use a implementar un try/catch:

```java
public class Potencia {

    int base;
    int exponente;



    public Potencia(int base, int exponente){
        this.base = base;
        this.exponente = exponente;

    }

    public int getBase() {
        return base;
    }

    public void setBase(int base) {
        this.base = base;
    }

    public int getExponente() {
        return exponente;
    }

    public void setExponente(int exponente) {
        this.exponente = exponente;
    }

    /**
     * Multiplica potencias de la misma base
     * @param potencia
     */
    public Potencia multiplicar(Potencia potencia)throws PotenciaIncorrectaException{
        Potencia resultado= null;
        if (this.getBase()==potencia.getBase()){
            resultado = new Potencia(this.getBase(), this.getExponente()+potencia.getExponente());
        } else throw new PotenciaIncorrectaException("La base debe de ser igual");
        return resultado;

    }

/** Calcula la potencia de una potencia
 *
 * @param exponente
 * @return
 */

public Potencia potenciaDePotencia(int exponente){
    return new Potencia(this.getBase(), this.getExponente()*exponente);
}

@Override
public String toString() {
    return "Potencia [base=" + base + ", exponente=" + exponente + "]";
}
}
```

Ejercicio 17.1

Mejora tu función del signo del zodiaco haciendo que si la fecha es inválida devuelva una excepción llamada IllegalArgumentException (existe ya en Java).

Ejercicio 17.2

Crea una clase Triángulo, que en caso de que tenga un estado inconsistente con un triángulo de una excepción InvalidTriangleException (creada por ti).

Ejercicio 17.3

Crea una clase DNI, con los campos que aparecen en el documento. Si algún dato del DNI no es válido, sea una fecha o el propio DNI dará una excepción InvalidIDCardException, creada por ti. Puedes usar la clase Calendar para validar las fechas.