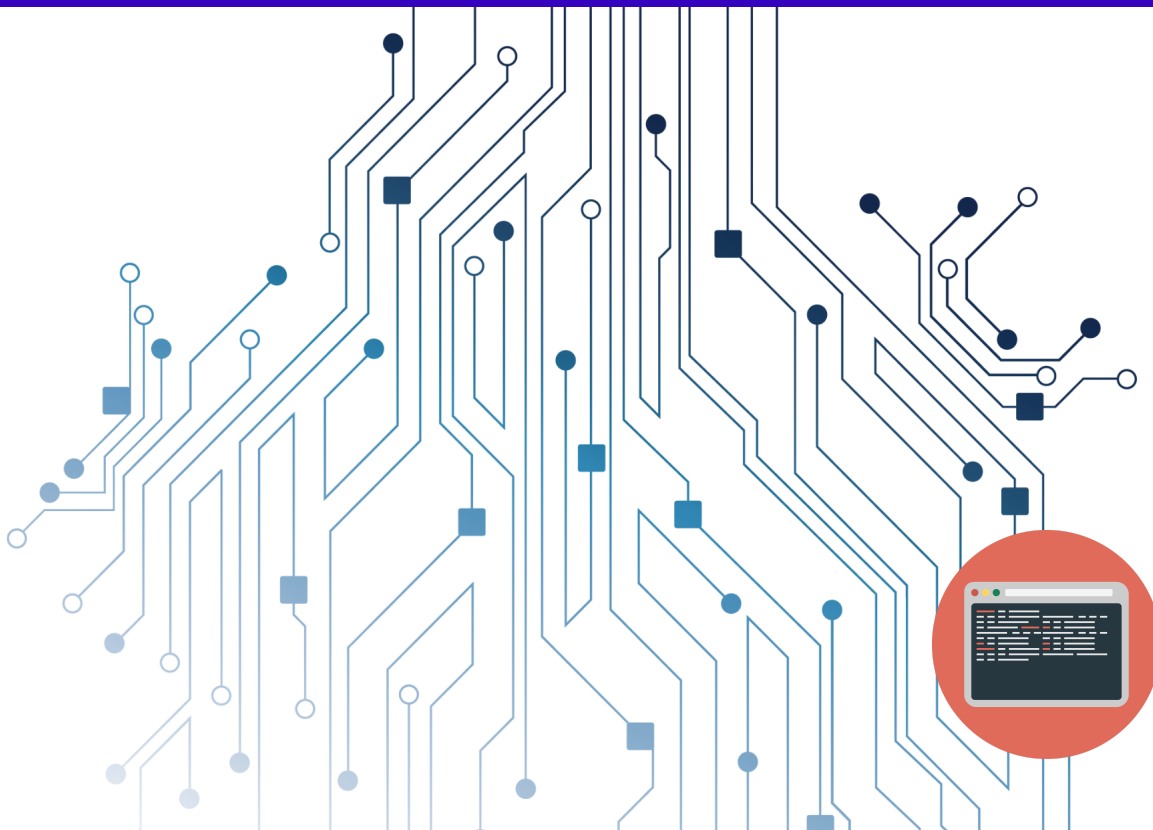


CFGS APLICACIONES MULTIPLATAFORMA - PROGRAMACIÓN



UNIDAD 6

EXCEPCIONES



DIEGO VALERO ARÉVALO

BASADO EN APUNTES Y EJERCICIOS PROPORCIONADOS POR
M^{ra} CARMEN DÍAZ GONZÁLEZ - IES VIRGEN DE LA PALOMA

U6 - EXCEPCIONES

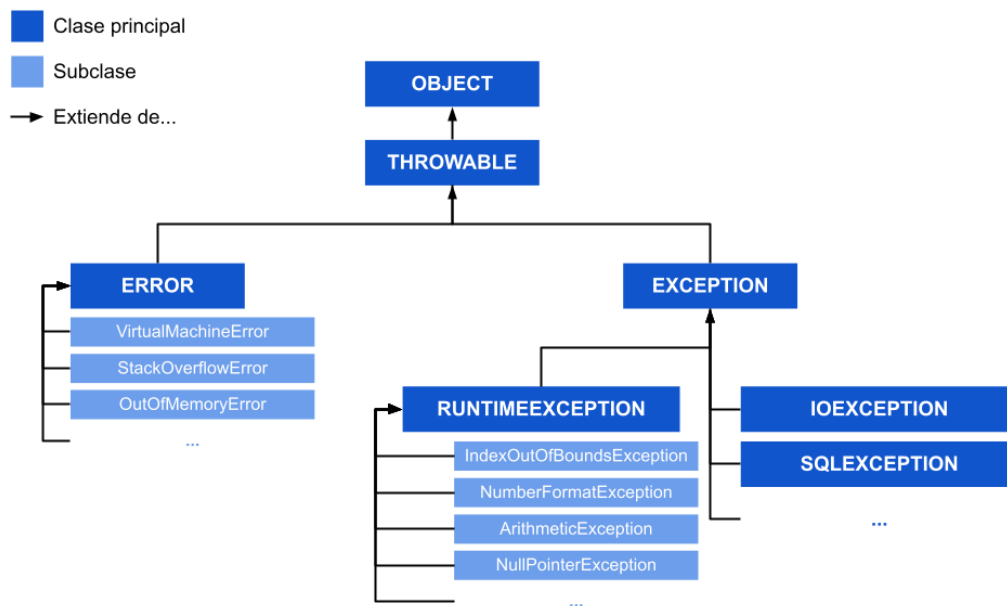
ÍNDICE

6.1 - EXCEPCIONES	1
· Introducción, diagrama de relación de clases.	
>> 6.1.1- MANEJO DE EXCEPCIONES (TRY-CATCH-FINALLY)	1
6.1.1.1 - Manejadores de excepciones	1
6.1.1.2 - Catch múltiples	3
>> 6.1.2- LANZAR EXCEPCIONES	4
6.1.2.1 - Lanzar excepciones desde métodos	5
6.1.2.2 - Llamar desde un método a otro que lanza excepciones	5
6.1.2.3 - Imprimir los mensajes de las excepciones	6
>> 6.1.3- CREAR EXCEPCIONES PERSONALIZADAS	6
>> 6.1.4- EJEMPLO DEL USO DE EXCEPCIONES	7
U6 - BATERÍA DE EJERCICIOS	10

6.1 - EXCEPCIONES

Una **excepción** es un **error semántico que se produce durante el tiempo de ejecución de un programa**.

Durante el curso y seguramente en tus propios programas has recibido algún tipo de error, tipo `OutOfBoundsException`, `NumberFormatException`, `InputMismatchException`, ... Todos estos son objetos de la clase **Exception**, que se generan cuando se detectan datos erróneos con los que el programa no puede trabajar. Otros tipos de errores internos provienen de la clase **Error**, los cuales vienen a su vez de la clase **Throwable**. Aquí tienes un diagrama con varios ejemplos de los dos:



Cuando se genera una excepción de cualquier tipo, decimos que esta se “lanza” (**throw**). Si no tratamos esos objetos, el programa se parará mostrando el error, por lo que existen formas de “cazar” (**catch**) esas excepciones para evitar ese parón.

Podemos **tratar** las excepciones que lance JAVA o **crear** las nuestras propias.

>> 6.1.1- MANEJO DE EXCEPCIONES (TRY-CATCH-FINALLY)

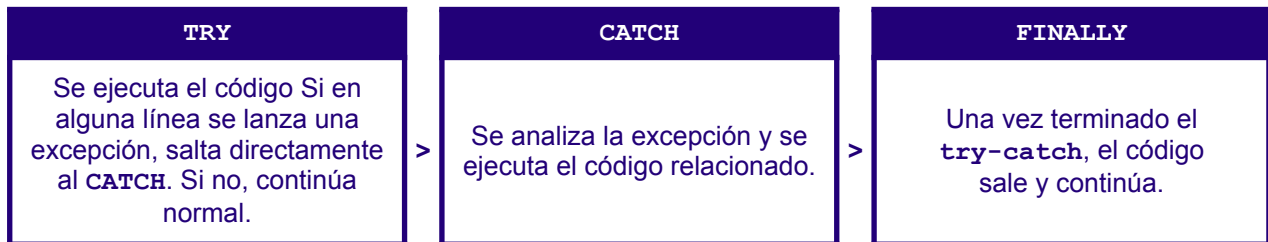
· 6.1.1.2- MANEJADORES DE EXCEPCIONES

Los **manejadores de excepciones** son mecanismos que trabajan en conjunto para tratar estas. Se conocen como el **bloque try-catch-finally**.

```
try{
    ...
}
catch(TipoException variable){
    ...
}
finally{
    ...
}
```

<code>try{ ... }</code>	El manejador <code>try</code> guarda el código que queremos ejecutar en el cual pueden darse excepciones.
<code>catch(TipoException variable){ ... }</code>	El manejador <code>catch</code> ejecutará su código si en el <code>try</code> se da la excepción especificada según su tipo.
<code>finally{ ... }</code>	El manejador <code>finally</code> ejecutará el código se dé o no la excepción. No es obligatorio y muchas veces no se pone.

El flujo de código es el siguiente:



EJEMPLO
<p style="text-align: center;">Ejemplo.java</p> <pre> 1 public class Ejemplo{ 2 public static void main(String[] args){ 3 4 int div; 5 int x = 6; 6 int y = 0; 7 8 div = x/y; 9 10 System.out.println("El resultado es "+div); 11 12 System.out.println("-Fin del programa-"); 13 } 14 }</pre> <p style="text-align: center;">Console</p> <pre> Exception in thread "main" java.lang.ArithmeticException: / by zero at Ejemplo.main(Ej00ExTry.java:8)</pre> <p>En este ejemplo puedes ver que al ejecutar salta un error y la ejecución se detiene. Vamos a probar a hacer try-catch para que esto no sea tan agresivo.</p>

EJEMPLO
<p style="text-align: center;">Ejemplo.java</p> <pre> 1 public class Ejemplo{ 2 public static void main(String[] args){ 3 4 int div; 5 int x = 6; 6 int y = 0; 7 8 try{ 9 div = x/y; 10 System.out.println("El resultado es "+div); 11 } 12 catch(ArithmeticException e){ 13 System.out.println("(!) No se puede dividir entre 0."); 14 } 15 } 16 }</pre>

```

14     }
15
16     System.out.println("-Fin del programa-");
17 }
18 }
19

```

Console

```

(!) No se puede dividir entre 0.
-Fin del programa-

```

Ahora mucho mejor. Podemos ver que **como se ha dado una excepción en el try**, el programa **ha saltado a los catch**, buscando la primera que corresponda y ejecutando el interior, que en este caso es un **sys0** con un mensaje. Después de esto, el programa termina porque no tiene nada más, pero si tuviese el código apropiado podría continuar y hacer más cosas.

· 6.1.1.2- CATCH MÚLTIPLES

Podríamos manejar todos los tipos de excepciones simplemente especificando un **catch** de tipo genérico **Exception**, pero lo ideal es manejar **cada tipo que creamos útil de diferente manera**. Para ello podemos hacer **catch** múltiples.

```

    try{
        ...
    }
    catch(TipoException1 variable){
        ...
    }
    catch(TipoException2 variable){
        ...
    }
    catch(TipoException3 variable){
        ...
    }
}

```

Podemos tener varias excepciones especificadas y luego tratar aquellas que no contemplemos con la genérica **Exception**, pero esta debe estar **al final del resto**, ya que si no se ejecutará la primera y no buscará el tipo específico.

EJEMPLO

Ejemplo.java

```

1  public class Ejemplo{
2      public static void main(String[] args){
3
4          Scanner keyboard = new Scanner(System.in);{
5
6              int div;
7              int x;
8              int y;
9
10             try{
11                 System.out.println("Introduce el dividendo:");
12                 x = keyboard.nextInt();
13
14                 System.out.println("Introduce el divisor:");
15                 y = keyboard.nextInt();
16
17                 div = x/y;
18

```

```

19         System.out.println("El resultado es "+div);
20     }
21     catch(ArithmeticException e){
22         System.out.println("(!) No se puede dividir entre 0.");
23     }
24     catch(InputMismatchException e){
25         System.out.println("(!) Los datos deben ser números.");
26     }
27     catch(Exception e){
28         System.out.println("(!) Ha habido algún error.");
29     }
30
31     System.out.println("-Fin del programa-");
32
33     keyboard.close();
34 }
35 }

```

Console

```

Introduce el dividendo:
6
Introduce el divisor:
a
(!) Los datos deben ser números.
-Fin del programa-

```

Ahora tenemos **múltiples catch** para **diferentes excepciones** que se puedan dar. La introducción de datos está en el **try** porque también se tiene que comprobar el dato que recogemos, si esperamos un **int** pero el usuario introduce cualquier otro caracter, la excepción saltará ahí. En este caso ha saltado en la línea 15.

>> 6.1.2- LANZAR EXCEPCIONES

Ahora que ya hemos visto el funcionamiento con excepciones por defecto, pasemos a **lanzar nuestras propias excepciones**. De esta manera podemos controlar aún más los datos que maneja el programa para no permitir ciertos valores, como por ejemplo que al pedir una edad no se permita un número negativo o que un DNI no tenga más dígitos de lo permitido.



Lanzar excepciones **no sirve para crear excepciones nuevas**. Simplemente usamos las que nos proporciona JAVA para adaptarlas a nuestro código y manejarlas de la forma que queramos.

Puedes consultar todos los tipos de excepciones y errores en la **API** de JAVA.

Para poder lanzar una **excepción** se escribe la siguiente línea:

```
throw new TipoException(["mensaje"]);
```

throw new

Los indicadores que indican el **lanzamiento de una excepción** y su instanciación.

tipoException();

El **tipo de excepción** que queremos tratar. Aquí podemos colocar cualquiera de las excepciones que nos proporciona JAVA.

["mensaje"]

Como opcional, podemos **escribir un string** que actúa como **mensaje** a imprimir por consola si se da la excepción.



La **línea** de lanzamiento de excepciones resume lo siguiente:

```

1 TipoException e = new TipoException();
2 throw e;

```

Los lanzamientos de excepciones se suelen usar **dentro de métodos** y forman parte de la **comprobación de los datos**. Estos **devolverán la excepción cuando se de la condición que le indiquemos para ser tratada en el catch**.



No podemos lanzar dos excepciones a la vez. Cada ámbito de código sólo puede tener una línea de lanzamiento de excepción. Si queremos preparar varias para lanzar, deberán estar en ámbitos diferentes. *(Por ejemplo, no podemos tener dos `throw new ...` en un `else`. Podemos tener uno pero el otro deberá estar en su propia condición).*

· 6.1.2.1- LANZAR EXCEPCIONES DESDE MÉTODOS

Para poder hacer que un método lance una excepción, debemos indicarlo en su cabecera

```
public/private [static] tipo/void nombreDelMetodo([parámetro/s]) throws TipoExcepcion {  
    ...  
}
```

`throws TipoExcepcion`

Debemos indicar la palabra `throws` junto a **qué excepción/es lanza el método**.



Un método puede lanzar **más de una excepción**, para indicarlo simplemente listamos todos los tipos en la cabecera **separados por comas (,)**:

```
1    ... throws TipoExcepcion1, ..., TipoExcepcionN {
```

· 6.1.2.2- LLAMAR DESDE UN MÉTODO A OTRO QUE LANZA EXCEPCIONES

Si hacemos que un método lance una excepción y **lo llamamos desde otro método**, el método que está más arriba **también debe indicar que lanzará esas excepciones**.

EJEMPLO

Excepciones.java

```
1  public class Excepciones {  
2      public static void main(String[] args) throws Exception {  
3          Scanner keyboard = new Scanner(System.in);  
4  
5          int n;  
6  
7          try {  
8              System.out.println("Introduce un número positivo");  
9              n = keyboard.nextInt();  
10             compruebaPositivo(n);  
11         }  
12         catch(ArithmeticException e) {  
13             System.out.println(e.getMessage());  
14         }  
15         keyboard.close();  
16     }  
17  
18     public static void compruebaPositivo(int n) throws ArithmeticException {  
19         if(n>0) {  
20             System.out.println("> ¡Bien hecho!");  
21         }  
22         else{  
23             throw new ArithmeticException("> ¡Eh! ¡Ese número no vale!");  
24         }  
25     }  
26 }
```

```

24     }
25 }
26 }

```

Console

```

Introduce un número positivo: -4
> ¡Eh! ¡Ese número no vale!

```

En este ejemplo, el método `compruebaPositivo` lanza una `ArithmeticException`, y como se llama desde el `main`, el propio `main` debe llevar un `throws Exception()`.



En el `main` se puede especificar la clase genérica `Exception` para así poder recibir y lanzar **cualquier tipo de excepción** porque la excepción en específico **se lanza desde otra clase**, el `main` solo debe estar preparado para recibirla y poder trabajar con ella.

· 6.1.2.3- IMPRIMIR LOS MENSAJES DE LAS EXCEPCIONES

Podemos imprimir por consola los **mensajes que traen las excepciones al “cazarlas”** a través del método `.getMessage()`:

```
nombreExcepcionCazada.getMessage()
```

Que se puede implementar de esta manera dentro de un `catch`:

```

catch (InvalidParameterException e) {
    System.out.println(e.getMessage());
}

```

>> 6.1.3- CREAR EXCEPCIONES PERSONALIZADAS

Podemos crear excepciones personalizadas a través de **clases que extiendan de la clase `Exception`**. Su estructura es la misma que cualquier otra clase, añadiendo la especificación de `extends` a la cabecera, un **identificador de serial** (opcional, que suele ser calculado por JAVA) y el **método que definirá qué ocurre cuando se llama a la excepción**, que suele ser un mensaje a través del constructor `super()`.

```

public class nombreExcepcion extends Exception {
    private static final long serialVersionUID = X;

    public nombreExcepcion() {
        super("mensaje");
    }
}

```



Cada clase de excepción personalizada sólo puede tener **una excepción**. Si queremos crear más, deberemos hacer que cada una tenga su propia clase.

Para usarla, simplemente la tratamos como un **tipo más de extensión**, de la misma manera que las demás.

EJEMPLO

ExcepcionNumeroNegativo.java

```

1 public class ExcepcionNumeroNegativo extends Exception {
2
3     private static final long serialVersionUID = 1L;
4
5     public ExcepcionNumeroNegativo() {
6         super("Los números no pueden ser negativos.");
7     }
8 }

```



```
7 }
8 }
```

Excepciones.java

```
1 public class Excepciones {
2     public static void main(String[] args) throws Exception {
3         Scanner keyboard = new Scanner(System.in);{
4
5         int n;
6
7         try {
8             System.out.println("Introduce un número positivo");
9             n = keyboard.nextInt();
10            compruebaPositivo(n);
11        }
12        catch(ExcepcionNumeroNegativo e) {
13            System.out.println(e.getMessage());
14        }
15        keyboard.close();}
16    }
17
18    public static void compruebaPositivo(int n) throws ExcepcionNumeroNegativo {
19        if(n>0) {
20            System.out.println("> ¡Bien hecho!");
21        }
22        else{
23            throw new ExcepcionNumeroNegativo();
24        }
25    }
26 }
```

Console

```
Introduce un número positivo: -4
Los números no pueden ser negativos.
```

Como ves, hemos **cambiado la excepción** del ejemplo anterior por una **personalizada**.

>> 6.1.4- EJEMPLO DEL USO DE EXCEPCIONES

Veamos un ejemplo con todo lo que hemos visto sobre excepciones:

EJEMPLO

MetodosConExcepciones.java

```
1 import java.security.InvalidParameterException;
2
3 public class MetodosConExcepciones {
4
5     public static void imprimePositivo(int p) throws InvalidParameterException {
6         if(p>0) {
7             System.out.println("> ¡Ese número es positivo!");
8         }
9         else{
10             throw new InvalidParameterException("> ¡Eh! ¡Ese número no vale!");
11         }
12     }
13
14     public static void imprimeNegativo(int p) throws InvalidParameterException {
15         if(p<=0) {
16             System.out.println("> ¡Ese número es negativo!");
17         }
18         else{
19             throw new InvalidParameterException("> ¡Eh! ¡Ese número no vale!");
20         }
21     }
22 }
```

```
21     }
22 }
23
```

PruebaExcepciones.java

```
1  import java.security.InvalidParameterException;
2  import java.util.InputMismatchException;
3  import java.util.Scanner;
4
5  public class PruebaExcepciones {
6      public static void main(String[] args) throws Exception {
7          Scanner keyboard = new Scanner(System.in);{
8
9              int n = 1;
10
11              while(true){
12                  do {
13                      System.out.println("Introduce un número positivo: ");
14
15                      try {
16                          n = keyboard.nextInt();
17                          MetodosConExcepciones.imprimePositivo(n);
18                      }
19                      catch(InputMismatchException e) {
20                          System.err.println("(!) Carácter no válido.");
21                          n = -1;
22                      }
23                      catch (InvalidParameterException e) {
24                          System.out.println(e.getMessage());
25                      }
26
27                      keyboard.nextLine();
28                      System.out.println();
29
30                  }while(n<0);
31
32                  do {
33                      System.out.print("Introduce un número negativo: ");
34
35                      try {
36                          n = keyboard.nextInt();
37                          MetodosConExcepciones.imprimeNegativo(n);
38                      }
39                      catch(InputMismatchException e) {
40                          System.err.println("(!) Carácter no válido.");
41                          n = 1;
42                      }
43                      catch (InvalidParameterException e) {
44                          System.out.println(e.getMessage());
45                      }
46                      keyboard.nextLine();
47                      System.out.println();
48                  }while(n>=0);
49              }
50          keyboard.close();}
51      }
52 }
```

Console

```
Introduce un número positivo: 2
> ¡Ese número es positivo!

Introduce un número negativo: -2
> ¡Ese número es negativo!

Introduce un número positivo: -4
> ¡Eh! ¡Ese número no vale!
```

```
Introduce un número positivo: 2  
> ¡Ese número es positivo!
```

```
Introduce un número negativo: ...
```

Como has podido observar, este es un programa que **irá pidiendo un número positivo y otro negativo y comprobará** si se han introducido correctamente. Al introducir un dato erróneo, según el método, **lanzan excepciones** e imprimirá el mensaje que se les ha especificado en la clase **MetodosConExcepciones**.

Como la clase **MetodosConExcepciones** lanza excepciones, el **main** debe tener un **throws Exception** en su cabecera.

U6 - BATERÍA DE EJERCICIOS

1	<p>Crea una biblioteca de funciones (librería) para arrayList de números enteros que contenga las siguientes funciones:</p> <p>a) generaArrayListInt: Genera un arrayList de tamaño n con números aleatorios cuyo intervalo (mínimo y máximo) se indica como parámetro.</p> <p>b) minimoArrayListInt: Devuelve el mínimo del arrayList que se pasa como parámetro.</p> <p>c) maximoArrayListInt: Devuelve el máximo del arrayList que se pasa como parámetro.</p> <p>d) mediaArrayListInt: Devuelve la media del arrayList que se pasa como parámetro.</p> <p>e) estaEnArrayListInt: Dice si un número está o no dentro de un arrayList.</p> <p>f) posicionEnArrayList: Busca un número en un arrayList y devuelve la posición (el índice) en la que se encuentra.</p> <p>g) volteaArrayListInt: Le da la vuelta a un arrayList.</p> <p>Crea una clase para probar todas las funciones de la librería, además está podrá ser utilizada en cualquier otro ejercicio.</p>
2	<p>Crea un programa para realizar cálculos relacionados con la altura (en metros) de personas. Se pedirá la altura de personas hasta que se introduzca un 0, y se almacenarán en un arrayList las alturas introducidas por teclado. Luego mostrará la altura media, máxima y mínima así como cuántas personas miden por encima y por debajo de la media.</p>
3	<p>Crea un programa que cree un arrayList de enteros y luego muestre el siguiente menú con distintas opciones:</p> <p>a) Mostrar valores. (inicialmente no tendrá ningún valor)</p> <p>b) Introducir valor al final.</p> <p>c) Introducir valor en una posición.</p> <p>d) Salir.</p> <p>La opción 'a' mostrará todos los valores por pantalla. La opción 'b' pedirá un valor V, y luego escribirá V en el arrayList La opción 'c' pedirá un valor V y una posición P, luego escribirá V en la posición P del arrayList.</p> <p>El menú se repetirá indefinidamente hasta que el usuario elija la opción 'd' que terminará el programa.</p>
4	<p>Crea un programa que permita al usuario almacenar una secuencia aritmética en un arrayList y luego mostrarla. Una secuencia aritmética es una serie de números que comienza por un valor inicial V, y continúa con incrementos de I. Por ejemplo, con V=1 e I=2, la secuencia sería 1, 3, 5, 7, 9... Con V=7 e I=10, la secuencia sería 7, 17, 27, 37... El programa solicitará al usuario V, I además de N (número de valores a crear)</p>
5	<p>Crea un programa que pida al usuario valores hasta que se introduzca -1 y los guarde en un arrayList. Luego muestra el arrayList por pantalla.</p>
6	<p>Crea un programa que cree un arrayList de enteros e introduzca la siguiente secuencia de valores: 1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, etc. hasta introducir 10 diez veces, y luego la muestre por pantalla.</p>

7	<p>Crea un programa que pida al usuario 10 cadenas de texto e introduzca las 5 primeros en un arrayList y los 5 últimos en otro arrayList. Por último, comparará ambos arrayList y le dirá al usuario si son iguales o no.</p>
8	<p>Supongamos una clase Producto con dos atributos:</p> <ul style="list-style-type: none"> • String nombre • int cantidad <p>Implementa esta clase con un constructor (con parámetros) además de los getters y setters de sus dos atributos, el método toString y el método equals. No es necesario comprobar los datos introducidos. A continuación, en el programa principal haz lo siguiente:</p> <ol style="list-style-type: none"> 1. Crea 5 instancias de la Clase Producto. 2. Crea un ArrayList. 3. Añade las 5 instancias de Producto al ArrayList. 4. Visualiza el contenido de ArrayList utilizando Iterator. 5. Elimina dos elementos del ArrayList. 6. Inserta un nuevo objeto producto en la posición 2 de la lista. 7. Visualiza de nuevo el contenido de ArrayList utilizando for-each. 8. Elimina todos los valores del ArrayList y muestra el tamaño.
9	<p>Crea un ArrayList con los nombres de 6 compañeros de clase. A continuación, muestra esos nombres por pantalla. Utiliza para ello un bucle for que recorra todo el ArrayList sin usar ningún índice.</p>
10	<p>Crear un programa que use ArrayList de números reales. El programa debe permitir mostrar un menú donde se pueda agregar un número, buscar un número, modificar un número, eliminar un número e insertar un número en una posición dada.</p>
11	<p>Implementa el control de acceso al área restringida de un programa. Se debe pedir un nombre de usuario y una contraseña. Si el usuario introduce los datos correctamente, el programa dirá "Ha accedido al área restringida". El usuario tendrá un máximo de 3 oportunidades. Si se agotan las oportunidades el programa dirá "Lo siento, no tiene acceso al área restringida". Los nombres de usuario con sus correspondientes contraseñas deben estar almacenados en una estructura de la clase HashMap.</p>
12	<p>Crea un mini-diccionario español-inglés que contenga, al menos, 20 palabras (con su correspondiente traducción). Utiliza un objeto de la clase HashMap para almacenar las parejas de palabras. El programa pedirá una palabra en español y dará la correspondiente traducción en inglés.</p>
13	<p>Realiza un programa que escoja al azar 5 palabras en español del mini- diccionario del ejercicio anterior. El programa irá pidiendo que el usuario teclee la traducción al inglés de cada una de las palabras y comprobará si son correctas. Al final, el programa deberá mostrar cuántas respuestas son válidas y cuántas erróneas.</p>
14	<p>Crear un programa para gestionar la plantilla de un equipo de futbol. El programa debe tener un HashMap con los jugadores actuales de la plantilla, de forma que la clave sea su número (que es único) y el valor su nombre. Será posible añadir nuevos jugadores (comprobar que no exista jugador con ese número para no machacarlo), eliminar jugadores, mostrar todos los jugadores y buscar si hay actualmente un jugador con el número indicado.</p>
15	<p>Crear un programa para gestionar el stock de productos en un almacén. Será necesario mantener información del nombre del producto, y las unidades del mismo. Se creará un menú que permita añadir productos, si el producto existe aumenta el stock, si no existe lo crea, eliminar productos, reducir el stock de un producto (teniendo en cuenta que no puede quedar stock negativo), buscar las unidades que hay de un determinado producto, listar todos los productos, y listar todos los productos junto con su stock.</p>
16	<p>Modificar el ejercicio 14, para que en vez de guardar de un jugador sólo el nombre, se almacene nombre y posición, que podrá ser (Delantero, Portero, Defensa o Medio_campo). Además, añadir funcionalidades para modificar la posición de un jugador, y modificar el nombre.</p>
17	<p>Crear un programa que inicialmente contenga una estructura de tipo HashMap, de forma que la clave sea un entero, y el valor otro entero. Se deberá mostrar el contenido completo del hashMap, posteriormente se modificarán los valores cuya clave sea par, multiplicando por dos el valor, se</p>

volverá a mostrar el contenido completo, y se modificarán los valores cuya clave sea impar, para sumar 10 al valor. Volviendo a mostrar el contenido completo.

Utilizando las clases creadas en el ejercicio 5 de la unidad 8 (Electrodomésticos) se deberá crear un hashMap para gestionar el almacén, teniendo en cuenta las siguientes características:

- Todos los electrodomésticos serán almacenados en un único HashMap
- La clave se asignará de la siguiente forma dependiendo del tipo de electrodoméstico:
 - Lavadora: Lvx, siendo x un número consecutivo.
 - Televisión: TVx
 - Cualquier otro: Elx
- Se tendrá un stock inicial de electrodomésticos
- Se podrán añadir cualquier electrodoméstico, la clave deberá generarse automáticamente dependiendo del tipo. El resto de datos se pedirán al usuario.
- Se podrán listar todos los electrodomésticos
- Se podrán listar todas las lavadoras
- Se podrán listar todos los televisores
- Se podrán listar todos los electrodomésticos que no sea televisor ni lavadora.
- Se podrán eliminar electrodomésticos, pero esa clave no será reutilizada en el futuro, simplemente se ignora.
- Obtener el precio final de un electrodoméstico a partir de su clave.

Todas las funcionalidades anteriores se realizarán mediante un menú que llamará a métodos siempre que sea posible. La clase Main debe quedar lo más clara posible.

Crea una clase Empleado con los siguientes atributos y constructor:

```
private String nombre;  
private double sueldo;  
public Empleado(String nombre, double sueldo)
```

Además tendrá los siguientes métodos:

```
public String getNombre()  
public double getSueldo()
```

Escribe un programa que utilizando una colección HashSet permita, a través de un menú, gestionar una lista de empleados de número ilimitado donde no se permitirán nombres de empleados repetidos y siendo el orden de los mismos totalmente irrelevante:

- 1.-Introducir empleado
- 2.-Listar empleados
- 3.-Eliminar empleado
- 4.-Borrar todos
- 5.-Mostrar número de empleados
- 6.-Buscar empleado
- 7.-Salir

- Para introducir un empleado hará falta también introducir su sueldo.
- Al listar los empleados se mostrará su nombre y sueldo.
- Para eliminar un empleado será necesario introducir su nombre.
- El programa no será sensible a mayúsculas