Programación

Práctica 1

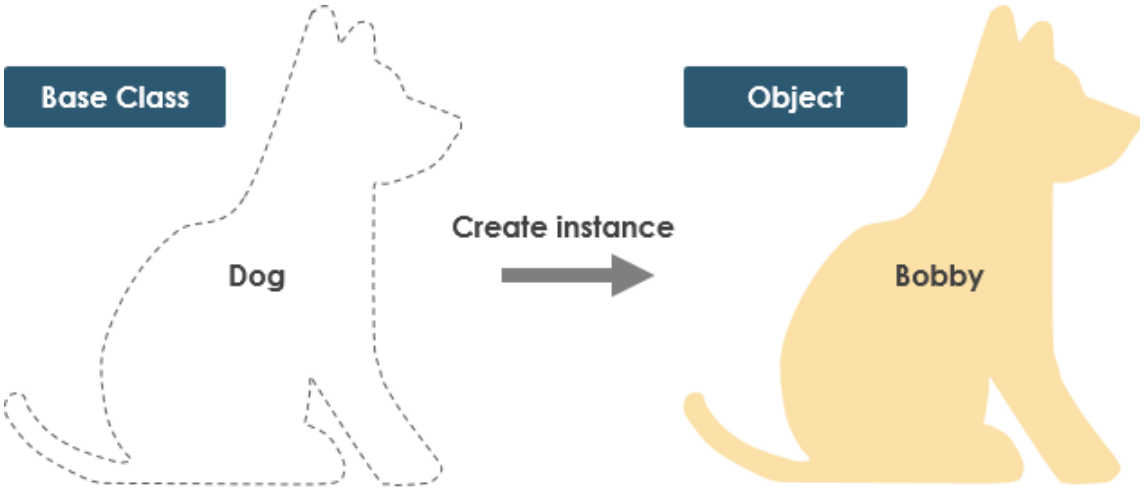
Programación Orientada a Objetos

Contenido

Objetivos

Los objetivos de esta práctica son que el alumno/a:

- Ser capaz de generar sus propios tipos, basándose en las características de la programación orientada a objetos proporcionadas por Java
- Ser capaz de crear diferentes instancias de tipos propios
- Ser capaz de diferenciar entre los conceptos de Clase, Instancia y Encapsulación, elementos básicos de la programación orientada a objetos
- Ser capaz de diferenciar entre miembro de instancia y miembro de clase
- Ser capaz de desarrollar una aplicación completa, organizada en diferentes paquetes y ficheros



Properties	Methods
Color	Sit
Eye Color	Lay Down
Height	Shake
Length	Come
Weight	

Property Values	Methods
Color: Yellow	Sit
Eye Color: Brown	Lay Down
Height: 17 in	Shake
Length: 35 in	Come
Weight: 24 pounds	

Source: https://medium.com/@singhamritpal49/object-oriented-programming-9beb509d4fcf

Programación orientada a objetos

Los lenguajes de programación orientada a objetos expanden el concepto de estado de la máquina en la forma de estado distribuido entre objetos.

Estos lenguajes, como Java, C++, Python, C# o Dart, ofrecen características como las siguientes:

- **Ocultación**: La capacidad de un tipo de ocultar al usuario parte de su funcionamiento, de modo que el usuario solo conoce *qué* puede hacer con el objeto, pero no *cómo* funciona internamente (para más información, ver interfaces)
- **Herencia**: La posibilidad de reaprovechar código entre clases mediante relaciones de tipo *is-a*
- **Polimorfismo**: Según el tipo de una instancia, frente a una misma operación, el objeto se puede comportar de una forma u otra

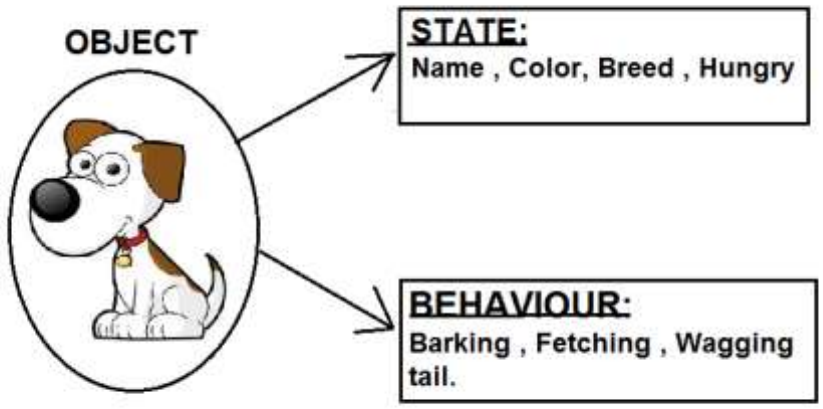
En esta práctica solo trabajaremos sobre el concepto de Ocultación, dado que Herencia y Polimorfismo son conceptos más avanzados que se impartirán en otras asignaturas (e.g., LTPP)

Este apartado no pretende ser un reemplazo del contenido visto en teoría, sino un repaso rápido de aquellos conceptos clave para la realización de las prácticas.

Clases e instancias

Una **clase** permite que el usuario pueda definir un tipo propio en base a un estado (variables) y comportamiento (métodos).

Una **clase** podemos verla como una plantilla que recoge las características del tipo declarado.



Source: https://programmingforengineering.blogspot.com/2016/07/introduction-to-oop-in-c-plus-plus.html

Dog.java

```
1 public class Dog {
2
3    // State
4    public String name;
5    public String breed;
6    public int age;
7
8    // Behaviour
9    public void bark() {...}
10   public void wagTail() {...}
11 }
```

Instancias y acceso a miembros

Brevemente, una **clase** es como un plano para construir una casa, mientras que un objeto/**instancia** es como la casa construida a partir de ese plano. Cada instancia creada a partir de una clase tiene su propia copia de los atributos y comportamientos definidos en la clase.

Main.java

```
1 public class Demo {
2
3    public static void main(String[] args) {
4
5       Dog rufus = new Dog();
6       rufus.name = "rufus";
7       rufus.breed = "Dalmatian";
8       rufus.age = 1;
9
10      rufus.bark();
11   }
12 }
```

A través de una **instancia**, podemos acceder a los miembros visibles (ver ocultación) a través del operador.

Tipos primitivos y referencias

Como se estudió en cursos anteriores, las clases, como los arrays, son tipos referencia, por lo tanto, en el ejemplo anterior, la variable *rufus* reside en el stack, y su contenido es una referencia al lugar donde realmente se almacenan los datos, el heap.

Main.java

```
1 public class Demo {
2
3    public static void main(String[] args) {
4
5        Dog rufus = new Dog();
6        rufus.name = "rufus";
7        rufus.breed = "Dalmatian";
8        rufus.age = 1;
9        setDogName(rufus, "rufián");
10       rufus.bark();
11    }
12
13    public static void setDogName(Dog dog, String newName) {
14        dog.name = newName;
15    }
16 }
```

Enlace pythontutor.com

Ocultación => Getters & Setters

El principio de ocultación trata de ocultar el funcionamiento de un tipo al usuario. Así, éste se puede focalizar en las acciones que puede hacer con el objeto, y no en cómo funciona.

Usando los modificadores de acceso, podemos elegir a qué puede acceder el usuario de la clase y a qué no.

private	Accesible solo desde dentro la clase
Sin modificador	Accesible desde dentro de la clase, y otras clases en el mismo paquete
public	Accesible desde cualquier parte

En esta asignatura, el principio de ocultación normalmente afecta a las variables de instancia/clase. La idea detrás de este principio es que el usuario no pueda acceder y modificar estas variables, que afectan al funcionamiento de la clase, de manera directa.

Para que el usuario acceda a esta información, añadimos a la clase lo que conocemos como *getters* y *setters*.

Dog.java

```
1 public class Dog {
2
3     // State
4     private String name;
5     private String breed;
6     private int age;
7
8     // Behaviour
9
```

```
10      // Getter & setter for "name" member
11      public String getName() {
12          return this.name;
13      }
14      public void setName(String name) {
15          this.name = name;
16      }
17
18      public void bark() {...}
19      public void wagTail() {...}
20  }
```

El uso de `this` dentro de un MÉTODO DE INSTANCIA (e.g., `getName`) permite acceder al objeto implícito, es decir, a la instancia que invoca el método.

Main.java

```
1  public class Demo {
2
3      public static void main(String[] args) {
4
5          Dog rufus = new Dog();
6          // rufus.name = "rufus";     // ERROR
7
8          rufus.setName("rufián");
9          System.out.println(rufus.getName());
10
11          rufus.bark();
12      }
13  }
```

Ciclo de vida

Un objeto, como cualquier variable en Java, dispone de un ciclo de vida, desde que se reserva memoria para el mismo, hasta que se libera:

- **Instanciación**: Reserva de memoria
- **Inicialización**: Preparación del estado del objeto para su uso
- **Uso**
- **Finalización**: Liberar posibles dependencias (e.g., cerrar un fichero)
- **Destrucción**: Liberación de memoria del propio objeto

En este curso, nos vamos a centrar en la fase 1 y 2.

Dog.java

```
1 public class Dog {
2
3     // State
4     private String name;
5     private String breed;
6     private int age;
7
10     // Constructor
11     public Dog(String name, String breed, int age) {
12         this.name = name;
13         this.breed = breed;
14         this.age = age;
15     }
16
17     // Getter & setter for "name" member
```

```
18      public String getName() {
19          return this.name;
20      }
21      public void setName(String name) {
22          this.name = name;
23      }
24      // Behaviour
25      public void bark() {...}
26      public void wagTail() {...}
27  }
```

Main.java

```
1  public class Demo {
2
3      public static void main(String[] args) {
4
5          // Fase 1 => new; Fase 2 => Constructor
6          Dog rufus = new Dog("Rufus", "Dalmatian", 2);
7
8          // Fase 3 => usamos el objeto
9          rufus.setName("rufián");
10         System.out.println(rufus.getName());
11
12         rufus.bark();
13     }    // Fase 4 y 5 => El objeto sale de su ámbito y no hay
referencias al mimso
14          // Se libera la memoria del mismo
15 }
```

Cuando se invoca el constructor, el objeto se encuentra en un estado inicial consistente para empezar a trabajar con él (i.e., invocar métodos, pasarlo a diferentes métodos, etc.)

Variables de clase vs. instancia

Cada una de las instancias que hemos creado a partir de la definición de la clase Dog tienen una copia propia para cada variable que modela el estado:

Dog.java

```java
public class Dog {

    // State
    private String name;
    private String breed;
    private int age;

     // Constructor
     public Dog(String name, String breed, int age) {
         this.name = name;
         this.breed = breed;
         this.age = age;
     }

     // // Getter & setter for "name" member
     public String getName() {
         return this.name;
     }
     public void setName(String name) {
         this.name = name;
     }

     public void bark() {...}
     public void wagTail() {...}
}
```

Main.java

```java
public class Demo {

    public static void main(String[] args) {

        Dog rufus = new Dog("Rufus", "Dalmatian", 2);
        Dog gaby = new Dog("Gaby", "Golden Retriever", 4);
        Dog linda = new Dog("Linda", "Bulldog", 1);
}
```

Sin embargo, también es posible declarar variables que serán compartidas por todas las instancias, las variables de clase:

Dog.java

```
1 public class Dog {
2
3      // State
4     private String name;
5     private String breed;
6     private int age;
7     private static int numDogs = 0;    // Class variable
8
9     // Behaviour
10
11     // Constructor
12     public Dog(String name, String breed, int age) {
13         this.name = name;
14         this.breed = breed;
15         this.age = age;
16         numDogs ++;
17     }
18
19     public static int getNumDogs() {    // Class method
20         return numDogs;
21     }
22
23     // Getter & setter for "name" member
24     public String getName() {
25         return this.name;
26     }
27     public void setName(String name) {
```

```
28          this.name = name;
29      }
30
31      public void bark() {...}
32      public void wagTail() {...}
33  }
```

Main.java

```
1  public class Demo {
2
3      public static void main(String[] args) {
4
5          Dog rufus = new Dog("Rufus", "Dalmatian", 2);
6          System.out.println(Dog.getNumDogs());    // 1
7
8          Dog gaby = new Dog("Gaby", "Golden Retriever", 4);
9          System.out.println(Dog.getNumDogs());    // 2
10
11          Dog linda = new Dog("Linda", "Bulldog", 1);
12          System.out.println(Dog.getNumDogs());    // 3
13 }
```

No es necesaria ninguna instancia para acceder ni a las variables ni a los métodos de clase.

Como se puede observar, se accede a estos miembros a través de la clase:

e.g., `Dog.getNumDogs();`

¿Podemos utilizar `this` dentro de un método de clase?

Otro ejemplo

Dog.java

```
1 public class Dog {
2
3     // State
4     private String name;
5     private String breed;
6     private int age;
7     private static int numDogs = 0;    // Class variable
8
9     // Behaviour
10
11     // Constructor
12     public Dog(String name, String breed, int age) {
13         this.name = name;
14         this.breed = breed;
15         this.age = age;
16         numDogs += 1;
17     }
18
19     public static int getNumDogs() {    // Class method
20         return numDogs;
21     }
```

```
22
23      public static String getBreedGroup(String breed) {    //
Class method
24          switch(breed) {
25              case "Labrador Retriever":
26                  return "Sporting Group";
27              case "Bulldog":
28                  return "Non-Sporting Group";
29              case "Poodle":
30                  return "Toy Group";
31              default:
32                  return "Unrecognized Breed";
33          }
34      }
35
36      // Getter & setter for "name" member
37      public String getName() {
38          return this.name;
39      }
40      public void setName(String name) {
41          this.name = name;
42      }
43
44      public void bark() {...}
45      public void wagTail() {...}
46  }
```

Main.java

```
1  public class Demo {
2
3      public static void main(String[] args) {
4
```

```
5         String group1 = Dog.getBreedGroup("Bulldog");   //
"Non-Sporting Group"
6         String group2 = Dog.getBreedGroup("Dalmatian");  //
"Unrecognized Breed"
7 }
```

Tareas a realizar

Esta práctica está organizada en base al desarrollo guiado de una serie de pasos, de tal forma que el paso N+1 no se pueda realizar hasta que el paso N no se haya finalizado y testeado.

En esta práctica, se propone que el alumno/a desarrolle un software de gestión de una tienda de ropa ficticia. Este software será utilizado por un empleado de la tienda, y con él podrá realizar diferentes acciones sobre un inventario de prendas y un registro de ventas. Estas acciones son:

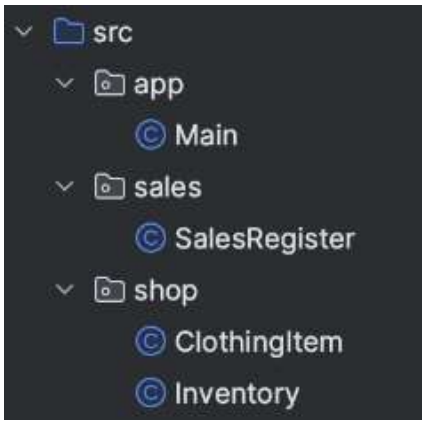
1. Añadir una prenda al inventario
2. Consultar las prendas que hay en el inventario
3. Vender una prenda
4. Consultar la cantidad de ventas realizadas
5. Salir del programa

Objetivo final y entregable

El programa en ejecución deberá mostrar un menú interactivo como el que se puede observar en el vídeo a continuación.

El alumno deberá tener sincronizado en Github el proyecto desde IntelliJ o Eclipse, como se explicó en la práctica 0.

Estructura de paquetes



Resultado

https://www.youtube.com/watch?v=INoFJVWqWK0

Tarea 1 - ClothingItem

Esta clase representa una prenda de ropa de una tienda (e.g., una camiseta, unos vaqueros... la clase contempla de manera genérica todas las posibles prendas)

1. Crea una clase `ClothingItem` en el paquete `shop`
2. La clase contendrá las siguientes variables de instancia:
    - `name` de tipo `String`
    - `price` de tipo `double`
    - `size` de tipo `char`
3. Declara el constructor de la clase para inicializar las variables de instancia con el mismo orden (i.e., estado)
4. Aplica el principio de ocultación para evitar el acceso a las variables de instancia de manera directa, y declara getters & setters para las tres
5. Sobrescribe el método `toString`, que únicamente devolverá la representación en `String` de la prenda, es decir, el nombre, precio y tamaño en una única `String`

La variable `size` solo podrá tener los valores `'S'`, `'M'` o `'L'`, por lo que deberás comprobarlo tanto en el constructor si son válidos. En caso erróneo y asignar el valor `'S'` por defecto.

En el setter encargado de modificar la variable `size`, `setSize`, si nos pasan un valor inválido, no modificar la variable.

Cuando tengas el código preparado puedes comprobar que es correcto con la herramienta de test. Crea una clase TestClothingItem en el

paquete test. Copia los recursos facilitados de la clase. En la clase main añade el siguiente código para realizar el test.

```
ClothingItem item1 = new ClothingItem("Camisa", 25.99, 'M');
if (item1 != null) {
      TestClothingItem.checkClass(item1.getClass());
}
```

Observa las indicaciones por consola y corrige el código de ClothingItem hasta que esté todo correcto.

Tarea 2 - Inventory

Esta clase representa una colección de prendas de ropa en lo que denominamos con inventario. Aplica el concepto de stock para simbolizar cuantas prendas de un tipo específico tenemos en el inventario.

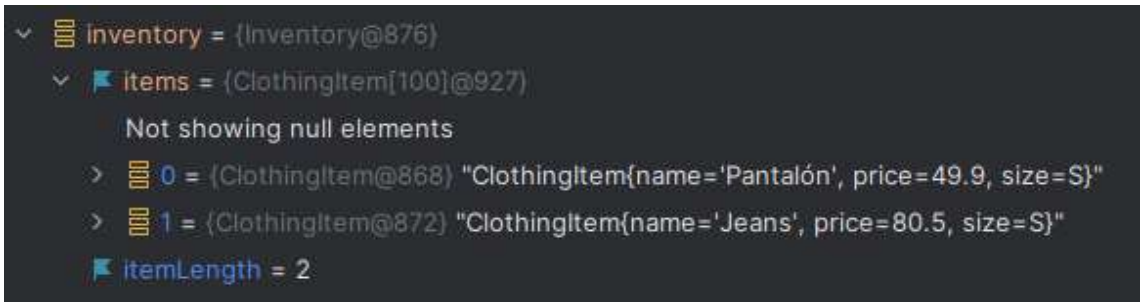
1. Crea una clase `Inventory` en el paquete `shop`. Esta clase almacenará las prendas de la tienda en un array de `ClothingItem`, en la cual implementaremos los siguientes métodos, detallados más abajo:

    - Comprobar la cantidad (numérica) de prendas no supera el stock máximo.
    - Añadir una prenda
    - Consultar stock de una prenda específica (dado su `name` y `size`)
    - Eliminar una prenda (dado su `name` y `size`)
    - Encontrar una prenda (dado su `name` y `size`)
    - Representar el inventario, en forma de String, de manera tabular
2. Declara las siguientes variables:
    a) Variable de instancia `items`, de tipo `ClothingItem`, donde almacenaremos las prendas como array (arreglo)
    b) Variable de instancia `itemLength`, para saber el total del stock
    c) Variable de instancia, constante, `MAX_SIZE`, donde indicará la capacidad máxima que permite el almacén
3. Declara el constructor de la clase, que aceptará la capacidad del inventario (i.e., `MAX_SIZE`) e inicializará el array a vacío. Tras la creación del inventario, tendremos un array de tamaño máximo `MAX_SIZE`, pero en él habrá 0 elementos (`itemLength`)
4. Declara un método `getItemCount`, que devuelva cuantos items hay en el inventario

5. Declara un método `addItem` que permita añadir prendas al inventario. Dado un `ClothingItem`, lo deberá añadir en la última posición libre del array
    a) Si no hay prendas en el inventario, será colocado en la posición 0; si hay una prenda ya, pues estará en la posición 1; y así sucesivamente
    b) Comprobar que el inventario no se encuentra a su máxima capacidad
6. Declara un método para `checkStock` comprobar si hay stock para una prenda en concreto. Dado el `name` y el `size` de la prenda, deberás realizar una búsqueda y devolver cuantas prendas hay de ese producto en el inventario (Para ello deberás recorrer el vector)
7. Declara un método `removeItem` para eliminar una prenda del inventario. Dado el `name` y el `size` de la prenda, la eliminará de su hueco.
    a) Primero, comprueba si está en stock (paso 6)
    b) Si está en stock, busca su posición (No debes iterar hasta `MAX_SIZE`)
    c) Una vez encontrada la prenda en su posición, deberás desplazar las prendas que haya en posiciones posteriores, una posición adelante, evitando que la última prenda aparezca duplicada
    d) El hueco de la última prenda deberá ser puesto a valores nulos.

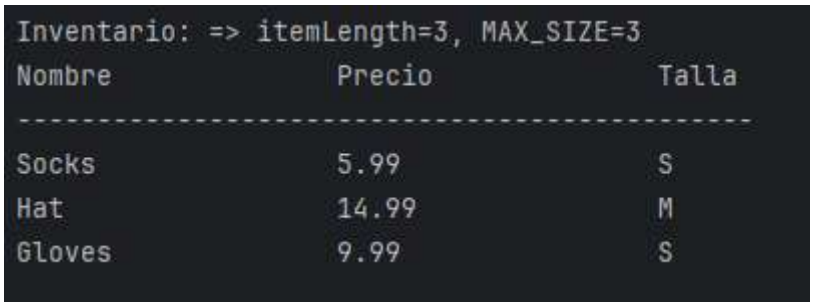
Ejemplo memoria en modo debug después de añadir tres ClothingItem.

Después de eliminar primer elemento inventory.removeItem("Camisa", 'S');



8. Declara un método `extractItem` para encontrar una prenda, dados su `name` y `size`. Si la encuentra, **devolverá** el objeto `ClothingItem`
    a) Primero, comprueba si está en stock (paso 6)
    b) Si está en stock, búscala y que devuelva el objeto
    c) Deberá eliminarse del stock (paso 7)
    d) Si no existe devuelve null
9. Sobrescribe el método `toString` para representar, de forma tabular, el inventario (mediante saltos de línea y tabuladores)
    a) La representación en String contendrá la cantidad de prendas en stock, su capacidad, y los items en formato tabular

Representación del inventario en forma de `String` tras imprimirlo por consola:



Tarea 3 - Main, parte 1

Esta clase representa el punto de entrada de la aplicación. Más adelante, implementará un menú interactivo para un gestor del inventario; sin embargo, en este momento solo la utilizaremos para testear que lo implementado en el inventario funciona.

1. Crea una clase `Main` en el paquete `app`, con el método `main`
2. En esta clase, declara una **variable de clase** de tipo `Inventory` e inicialízalo al inicio del punto de entrada del programa
3. COMPRUEBA que los diferentes métodos que hay añadido a la clase `Inventory` funcionan correctamente ANTES DE CONTINUAR
    a) Comprueba que puedes añadir varias prendas al inventario
    b) Comprueba que puedes comprobar el stock de una prenda
    c) Comprueba que puedes eliminar una prenda y el inventario se encuentra en un estado consistente (usa el debugger)
    d) Comprueba que, sí imprime el inventario de manera directa por consola, se invoca su método `toString` de manera *implícita* y se muestra el inventario en formato tabular
    e) Comprueba que extractIem obtiene el artículo y es elimina del stock
    f) Fija un MAX_ITEM y comprobar que no supera el límite del stock

Cuando lo tengas comprobado verifica con la herramienta de test que todo es correcto. Crea una clase TestInventory en el paquete test. Copia los recursos facilitados de la clase. En la clase main añade el siguiente código para realizar el test.

```
Inventory inventory = new Inventory(100);
if (inventory != null) {
       TestInventory.checkClass(inventory.getClass());
}
```

Verificar por consola que no da ningún error y corrige si los hubiera.

Tarea 4 - SalesRegister

Esta clase se encarga de mantener un registro (numérico) de las ventas procesadas, de manera genérica, independientemente del inventario. **Todos sus miembros serán de clase**, no existirán instancias de esta clase.

1. Declara la clase `SalesRegister` en el paquete `sales`
2. Declara una **variable de clase** de tipo `long` para contar cuantos artículos se han vendido `totalSalesCount`
3. Declara una **variable de clase** de tipo `double` para sumar las ventas totales que se han realizado, `totalSalesAmount`
4. Declara un **método de clase** `processSale` que procese una venta. Dado un inventario, el `name` y el `size` de un producto:
    a) Buscará el producto en el inventario (ya hemos implementado ese método de instancia)
        - Si no existe, imprimirá un mensaje por consola y finalizará su ejecución
    b) Si el producto existe, lo extraerá del inventario (ya hemos implementado este método de instancia)
    c) Incrementa el número de artículos vendidos
    d) Incrementará las ventas totales
    e) Devuelve el item
5. Declara un **método de clase** `getTotalSalesCount()` para conocer el recuento de ventas que se han realizado
6. Declara un **método de clase** `getTotalSalesAmount()` para conocer cuantas ventas se han procesado
7. Añade la función `getBalance()` que nos muestre un balance usado (5 y 6)
8. Añade un método que ponga a cero la cuenta de productos vendidos resetTotalSalesCount() y otro las ventas resetTotalSalesAmount()

Crea una clase TestSalesRegister en el paquete test. Copia los recursos facilitados de la clase. En la clase main añade el siguiente código para realizar el test.

```
TestSalesRegister.checkSalesRegister();
```

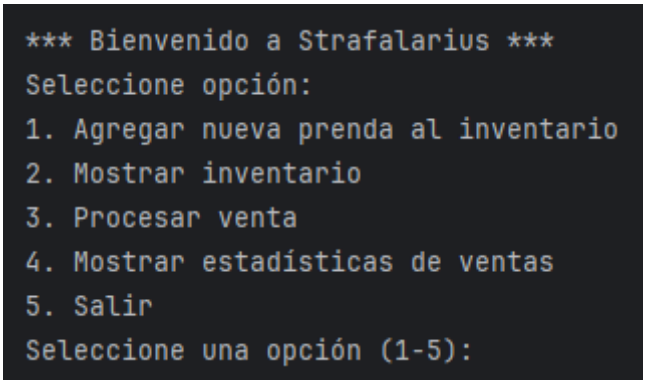
Verifica por consola que no da ningún error, y corrige si los hubiera.

Tarea 5 - Main, parte 2

El último paso de la práctica e implementar la interacción del usuario con el sistema, a través de un menú interactivo.

Es aconsejable realizar una llamada a `nextLine()` del `Scanner` cada vez que se utiliza para obtener un valor numérico.

1. Declara, a su vez, un `Scanner` como **variable de clase** e inicialízalo (`System.in`) justo después del inventario, para utilizarlo más adelante `scanner = new Scanner(System.in);`
2. Declara un método que imprima las posibles opciones del menú y, al mismo tiempo, devuelva la opción elegida por el usuario (teclado)
    a) Si la opción no es válida, devolverá -1



5 opciones posibles

3. Desde el método `main`, tras haber instanciado el inventario y el `Scanner`, muestra el menú en la estructura de iteración *do-while*
    a) Este bucle deberá mostrar el menú, capturar la opción seleccionada por el usuario (método ya implementado) e invocar un método diferente para cada una de las posibles opciones (vacíos por ahora, todos void)
    b) Utiliza una estructura *switch-case* para manejar la opción seleccionada

c) El menú siempre se volverá a mostrar tras la selección de una opción, excepto si la opción seleccionada es el 5. En tal caso, deberás salir del bucle y finalizar el programa

4. Declara el método para la opción 1:
    a) El usuario introducirá el name, price y size de una prenda
    b) Crea un ClothingItem con eso datos
    c) Añádelo al inventario (ya has implementado este método)
5. Declara el método para la opción 2:
    a) Imprime el inventario por consola (invocación implícita a su método toString)
6. Declara el método para la opción 3:
    a) El usuario introducirá el name y el size de una prenda
    b) Invoca el **método de clase** para procesar una venta de la clase SalesRegister (ya implementado)
    c) Imprime un mensaje por consola, bien si la venta se ha realizado (obtuviste la prenda al procesar la venta), bien si la venta no se ha realizado (no obtuviste la prenda al procesar la venta, tienes un valor null)
7. Declara el método para la opción 4:
    a) Muestra por consola cuantas ventas se han procesado (ya has implementado el **método de clase** en SalesRegister)

Ejemplo añadir item al inventario

```
Seleccione una opción (1-5): 1
Ingrese los detalles de la prenda:
Nombre: Camisa
Precio: 50.9
Talla (S/M/L): L
Prenda añadida al inventario correctamente.
```

Ejemplo listar inventario

```
Seleccione una opción (1-5): 2
Inventario: => itemLength=1, MAX_SIZE=100
Nombre              Precio              Talla
-----------------------------------------------
Camisa              50.9                L
```

Ejemplo vender prenda

```
Seleccione una opción (1-5): 3
Ingrese los detalles de la prenda a vender:
Nombre: Camisa
Talla (S/M/L): L
Clothing item 'Camisa' (Size L) extracted from inventory.
Venta procesada. Artículo vendido: ClothingItem{name='Camisa', price=50.9, size=L}
```

Autotest / Autoevaluación

Junto con el material de las prácticas se encuentran tres ficheros uno para comprobar cada una de las tareas de creación de clases:

- Tarea 1 – ClothingItem
- Tarea 2 – Inventory
- Tarea 4 – SalesRegister

Esta práctica está organizada en base al desarrollo guiado de una serie de pasos, de tal forma que el paso N+1 no se pueda realizar hasta que el paso N no se haya finalizado.

El alumno/a deberá construir las tres clases y comprobar el funcionamiento a través de Main con las instrucciones indicadas para realizar un menú de usuario.

El alumno/a deberá crear un paquete denominado test donde se incluirán los recursos facilitados.