Tema 4: Utilización de objetos.

1.- Introducción.

Si nos paramos a observar el mundo que nos rodea, podemos apreciar que casi todo está formado por **objetos**. Existen coches, edificios, sillas, mesas, semáforos, ascensores e incluso personas o animales. **Todos ellos pueden ser considerados objetos, con una serie de características y comportamientos.** Por ejemplo, existen coches de diferentes marcas, colores, etc. y pueden acelerar, frenar, girar, etc., o las personas tenemos diferente color de pelo, ojos, altura y peso y podemos nacer, crecer, comer, dormir, etc.

Los programas son el resultado de la búsqueda y obtención de una solución para un problema del mundo real. Pero ¿en qué medida los programas están organizados de la misma manera que el problema que tratan de solucionar? La respuesta es que muchas veces los programas se ajustan más a los términos del sistema en el que se ejecutarán que a los del propio problema.

Si redactamos los programas utilizando los mismos términos de nuestro mundo real, es decir, utilizando objetos, y no los términos del sistema o computadora donde se vaya a ejecutar, conseguiremos que éstos sean más legibles y, por tanto, más fáciles de modificar.

Esto es precisamente lo que pretende la **Programación Orientada a Objetos (POO)**, en inglés **OOP (Object Oriented Programming),** establecer una serie de técnicas que permitan trasladar los problemas del mundo real a nuestro sistema informático. Ahora que ya conocemos la sintaxis básica de Java, es el momento de comenzar a utilizar las características orientadas a objetos de este lenguaje, y estudiar los conceptos fundamentales de este modelo de programación.

Citas para pensar

John Johnson: Primero resuelve el problema. Entonces, escribe el código.

2.- Características de POO.

Cuando hablamos de Programación Orientada a Objetos, existen una serie de características que se deben cumplir. Cualquier lenguaje de programación orientado a objetos las debe contemplar. Las características más importantes del paradigma de la programación orientada a objetos son:

- **Abstracción.** Es el proceso por el cual definimos las características más importantes de un objeto, sin preocuparnos de cómo se escribirán en el código del programa, simplemente lo definimos de forma general. En la Programación Orientada a Objetos la herramienta más importante para soportar la abstracción es la **clase**. Básicamente, una clase es un tipo de dato que agrupa las características comunes de un conjunto de objetos. Poder ver los objetos del mundo real que deseamos trasladar a nuestros programas, en términos abstractos, resulta de gran utilidad para un buen diseño del software, ya que nos ayuda a comprender mejor el problema y a tener una visión global de todo el conjunto. Por ejemplo, si pensamos en una clase **`Vehículo`** que agrupa las características comunes de todos ellos, a partir de dicha clase podríamos crear objetos como **`Coche`** y **`Camión`**. Entonces se dice que **`Vehículo`** es una abstracción de **`Coche`** y de **`Camión`**.

- **Modularidad.** Una vez que hemos representado el escenario del problema en nuestra aplicación, tenemos como resultado un conjunto de objetos software a utilizar. Este conjunto de objetos se crean a partir de una o varias clases. Cada clase se encuentra en un archivo diferente, por lo que la modularidad nos permite modificar las características de la clase que define un objeto, sin que esto afecte al resto de clases de la aplicación.

- **Encapsulación.** También llamada **"ocultamiento de la información"**. La **encapsulación** o **encapsulamiento** es el mecanismo básico para ocultar la información de las partes internas de un objeto a los demás objetos de la aplicación. Con la encapsulación un objeto puede ocultar la información que contiene al mundo exterior, o bien restringir el acceso a la misma para evitar ser manipulado de forma inadecuada. Por ejemplo, pensemos en un programa con dos objetos, un objeto **`Persona`** y otro **`Coche`**. **`Persona`** se

comunica con el objeto **Coche** para llegar a su destino, utilizando para ello las acciones que Coche tenga definidas como por ejemplo conducir. Es decir, **Persona** utiliza **Coche** pero no sabe cómo funciona internamente, sólo sabe utilizar sus métodos o acciones.

- **Jerarquía**. Mediante esta propiedad podemos definir relaciones de jerarquías entre clases y objetos. Las dos jerarquías más importantes son la jerarquía **"es un"** llamada **generalización** o **especialización** y la jerarquía **"es parte de"**, llamada **agregación**. Conviene detallar algunos aspectos:
  - La generalización o especialización, también conocida como **herencia**, permite crear una clase nueva en términos de una clase ya existente (herencia simple) o de varias clases ya existentes (herencia múltiple). Por ejemplo, podemos crear la clase**CochedeCarreras** a partir de la clase **Coche**, y así sólo tendremos que definir las nuevas características que tenga.
  - La agregación, también conocida como **inclusión**, permite agrupar objetos relacionados entre sí dentro de una clase. Así, un**Coche** está formado por **Motor**, **Ruedas**, **Frenos** y **Ventanas**. Se dice que **Coche** es una agregación y **Motor**, **Ruedas**, **Frenos** y **Ventanas** son agregados de **Coche**.

- **Polimorfismo**. Esta propiedad indica la capacidad de que varias clases creadas a partir de una antecesora realicen una misma acción de forma diferente. Por ejemplo, pensemos en la clase **Animal** y la acción de expresarse. Nos encontramos que cada tipo de **Animal** puede hacerlo de manera distinta, los **Perros** ladran, los **Gatos** maullan, las**Personas** hablamos, etc. Dicho de otra manera, el polimorfismo indica la posibilidad de tomar un objeto (de tipo **Animal**, por ejemplo), e indicarle que realice la acción de expresarse, esta acción será diferente según el tipo de mamífero del que se trate.

3.- Clases y Objetos. Características de los objetos.

Un **objeto es un conjunto de datos con las operaciones definidas para ellos.** Los objetos tienen un **estado** y un **comportamiento**.

Por tanto, estudiando los objetos que están presentes en un problema podemos dar con la solución a dicho problema. Los objetos tienen unas características fundamentales que los distinguen:

- **Identidad**. Es la característica que permite diferenciar un objeto de otro. De esta manera, aunque dos objetos sean exactamente iguales en sus atributos, son distintos entre sí. Puede ser una dirección de memoria, el nombre del objeto o cualquier otro elemento que utilice el lenguaje para distinguirlos. Por ejemplo, dos vehículos que hayan salido de la misma cadena de fabricación y sean iguales aparentemente, son distintos porque tienen un código que los identifica.
- **Estado**. El estado de un objeto viene determinado por una serie de parámetros o atributos que lo describen, y los valores de éstos. Por ejemplo, si tenemos un objeto `Coche`, el estado estaría definido por atributos como `Marca, Modelo, Color, Cilindrada,` etc.
- **Comportamiento**. Son las acciones que se pueden realizar sobre el objeto. En otras palabras, son los métodos o procedimientos que realiza el objeto. Siguiendo con el ejemplo del objeto **Coche**, el el comportamiento serían acciones como: `arrancar(), parar(), acelerar(), frenar(),` etc.

3.1.- Propiedades y métodos de los objetos.

Como acabamos de ver todo objeto tiene un estado y un comportamiento. Concretando un poco más, las partes de un objeto son:

- **Campos, Atributos o Propiedades:** Parte del objeto que almacena los datos. También se les denomina V**ariables** M**iembro**. Estos datos pueden ser de cualquier tipo primitivo (`boolean`, `char`, `int`, `double`, etc) o ser su vez ser otro objeto. Por ejemplo, un objeto de la clase `Coche` puede tener un objeto de la clase `Ruedas`.
- **Métodos o Funciones Miembro:** Parte del objeto que lleva a cabo las operaciones sobre los atributos definidos para ese objeto.

La idea principal es que el objeto reúne en una sola entidad los datos y las operaciones, y para acceder a los datos privados del objeto debemos utilizar los métodos que hay definidos para ese objeto.

La única forma de manipular la información del objeto es a través de sus métodos. Es decir, si queremos saber el valor de algún atributo, tenemos que utilizar el método que nos muestre el valor de ese atributo. De esta forma, evitamos que métodos externos puedan alterar los datos del objeto de manera inadecuada. **Se dice que los datos y los métodos están encapsulados dentro del objeto.**

3.2.- Clases.

En otras palabras, **una clase es una plantilla o prototipo donde se especifican**:

- Los **atributos** comunes a todos los objetos de la clase.
- Los **métodos** que pueden utilizarse para manejar esos objetos.

Para declarar una clase en Java se utiliza la palabra reservada **class**. La declaración de una clase está compuesta por:

- **Cabecera de la clase.** La cabecera es un poco más compleja que como aquí definimos, pero por ahora sólo nos interesa saber que está compuesta por una serie de modificadores, en este caso hemos puesto `public` que indica que es una clase pública a la que pueden acceder otras clases del programa, la palabra reservada `class` y el nombre de la clase.
- **Cuerpo de la clase.** En él se especifican encerrados entre llaves los atributos y los métodos que va a tener la clase.

En la unidad anterior ya hemos utilizado clases, aunque aún no sabíamos su significado exacto. Por ejemplo, en los ejemplos de la unidad o en la tarea, estábamos utilizando clases, todas ellas eran clases principales, no tenían ningún atributo y el único método del que disponían era el método **main()**.

El método **main()** se utiliza para indicar que se trata de una clase principal, a partir de la cual va a empezar la ejecución del programa. Este método no aparece si la clase que estamos creando no va a ser la clase principal del programa.

4.- Utilización de objetos.

Si existe una clase, podemos crear objetos en nuestro programa a partir de esas clases.

Cuando creamos un objeto a partir de una clase se dice que hemos creado una **"instancia de la clase".** A efectos prácticos, "objeto" e "instancia de clase" son términos similares. Es decir, nos referimos a objetos como instancias cuando queremos hacer hincapié que son de una clase particular.

Los objetos se crean a partir de las clases, y representan casos individuales de éstas.

Para entender mejor el concepto entre un objeto y su clase, piensa en un **molde de galletas y las galletas**. El molde sería la clase, que define las características del objeto, por ejemplo su forma y tamaño. Las galletas creadas a partir de ese molde son los objetos o instancias.

Otro ejemplo, imagina una clase **Persona** que reúna las características comunes de las personas (color de pelo, ojos, peso, altura, etc.) y las acciones que pueden realizar (crecer, dormir, comer, etc.). Posteriormente dentro del programa podremos crear un objeto **Trabajador** que esté basado en esa clase **Persona**. Entonces se dice que el objeto **Trabajador** es una instancia de la clase **Persona**, o que la clase **Persona** es una abstracción del objeto**Trabajador**.

Cualquier objeto instanciado de una clase contiene una copia de todos los atributos definidos en la clase. En otras palabras, lo que estamos haciendo es reservar un espacio en la memoria del ordenador para guardar sus atributos y métodos. Por tanto, cada objeto tiene una **zona de almacenamiento propia** donde se guarda toda su información, que será distinta a la de cualquier otro objeto. A las variables miembro instanciadas también se les llama **variables instancia.** De igual forma, a los métodos que manipulan esas variables se les llama **métodos instancia.**

En el ejemplo del objeto **Trabajador**, las variables instancia serían **color_de_pelo**, **peso**, **altura**, etc. Y los métodos instancia serían **crecer()**, **dormir()**, **comer()**, etc.

4.1.- Ciclo de vida de los objetos.

Todo programa en Java parte de una única clase, que como hemos comentado se trata de la clase principal. Esta clase ejecutará el contenido de su método **main()**, el cual será el que utilice las demás clases del programa, cree objetos y lance mensajes a otros objetos.

Las instancias u objetos tienen un tiempo de vida determinado. Cuando un objeto no se va a utilizar más en el programa, es destruido por el recolector de basura para liberar recursos que pueden ser reutilizados por otros objetos.

A la vista de lo anterior, podemos concluir que los objetos tienen un ciclo de vida, en el cual podemos distinguir las siguientes fases:

- **Creación**, donde se hace la reserva de memoria e inicialización de atributos.
- **Manipulación**, que se lleva a cabo cuando se hace uso de los atributos y métodos del objeto.
- **Destrucción**, eliminación del objeto y liberación de recursos.

4.2.- Declaración.

Para la creación de un objeto hay que seguir los siguientes pasos:

- **Declaración**: Definir el tipo de objeto.
- **Instanciación**: Creación del objeto utilizando el operador `new`**.**

Pero ¿en qué consisten estos pasos a nivel de programación en Java? Veamos primero cómo declarar un objeto. Para la definición del tipo de objeto debemos emplear la siguiente instrucción:

```
<tipo> nombre_objeto;
```

Donde:

- `tipo` es la clase a partir de la cual se va a crear el objeto, y
- `nombre_objeto` es el nombre de la variable referencia con la cual nos referiremos al objeto.

Los tipos referenciados o referencias se utilizan para guardar la dirección de los datos en la memoria del ordenador.

Nada más crear una referencia, ésta se encuentra vacía. Cuando una referencia a un objeto no contiene ninguna instancia se dice que es una referencia nula, es decir, que contiene el valor **null.** Esto quiere decir que la referencia está creada pero que el objeto no está instanciado todavía, por eso la referencia apunta a un objeto inexistente llamado "nulo".

Para entender mejor la declaración de objetos veamos un ejemplo. Cuando veíamos los tipos de datos en la Unidad 2, decíamos que Java proporciona un tipo de dato especial para los textos o cadenas de caracteres que era el tipo de dato **String**. Veíamos que realmente este tipo de dato es un **tipo referenciado** y creábamos una variable mensaje de ese tipo de dato de la siguiente forma:

```
String mensaje;
```

> Los nombres de la clase empiezan con mayúscula, como **String**, y los nombres de los objetos con minúscula, como **mensaje**, así sabemos qué tipo de elemento utilizando.

Pues bien, **String** es realmente la clase a partir de la cual creamos nuestro objeto llamado **mensaje**.

Si observas poco se diferencia esta declaración de las declaraciones de variables que hacíamos para los tipos primitivos. Antes decíamos que **mensaje** era una variable del tipo de dato**String.** Ahora realmente vemos que **mensaje** es un objeto de la clase **String**. Pero **mensaje** aún no contiene el objeto porque no ha sido instanciado, veamos cómo hacerlo.

Por tanto, cuando creamos un objeto estamos haciendo uso de una variable que almacena la dirección de ese objeto en memoria. Esa variable es una **referencia** o un **tipo de datos referenciado**, porque no contiene el dato si no la posición del dato en la memoria del ordenador.

```
String saludo = new String ("Bienvenido a Java");
String s; //s vale null
s = saludo; //asignación de referencias
```

En las instrucciones anteriores, las variables **s** y **saludo** apuntan al mismo objeto de la clase **String**. Esto implica que cualquier modificación en el objeto saludo modifica también el objeto al que hace referencia la variable **s**, ya que realmente son el mismo.

4.3.- Instanciación.

Una vez creada la referencia al objeto, debemos crear la instancia u objeto que se va a guardar en esa referencia. Para ello utilizamos la orden **new**con la siguiente sintaxis:

```
nombre_objeto = new <Constructor_de_la_Clase>([<par1>, <par2>, ..., <parN>]);
```

Donde:

- `nombre_objeto` es el nombre de la variable referencia con la cual nos referiremos al objeto,
- `new` es el operador para crear el objeto,
- `Constructor_de_la_Clase` es un método especial de la clase, que se llama igual que ella, y se encarga de inicializar el objeto, es decir, de dar unos valores iniciales a sus atributos, y
- `par1-parN`, son parámetros que puede o no necesitar el constructor para dar los valores iniciales a los atributos del objeto.

Durante la instanciación del objeto, se reserva memoria suficiente para el objeto. De esta tarea se encarga Java y juega un papel muy importante el **recolector de basura,** que se encarga de eliminar de la memoria los objetos no utilizados para que ésta pueda volver a ser utilizada.

De este modo, para instanciar un objeto **String**, haríamos lo siguiente:

```
mensaje = new String;
```

Así estaríamos instanciando el objeto mensaje. Para ello utilizaríamos el operador **new** y el constructor de la clase **String** a la que pertenece el objeto según la declaración que hemos hecho en el apartado anterior. A continuación utilizamos el constructor, que se llama igual que la clase, **String**.

En el ejemplo anterior el objeto se crearía con la cadena vacía (""), si queremos que tenga un contenido debemos utilizar parámetros en el constructor, así:

```
mensaje = new String ("El primer programa");
```

Java permite utilizar la clase **String** como si de un tipo de dato primitivo se tratara, por eso no hace falta utilizar el operador new para instanciar un objeto de la clase **String**.

La declaración e instanciación de un objeto puede realizarse en la misma instrucción, así:

```
String mensaje = new String ("El primer programa");
```

4.4.- Manipulación.

Una vez creado e instanciado el objeto ¿cómo accedemos a su contenido? Para acceder a los atributos y métodos del objeto utilizaremos el nombre del objeto seguido del **operador punto (.)** y el nombre del atributo o método que queremos utilizar. Cuando utilizamos el operador punto se dice que estamos enviando un mensaje al objeto. La forma general de enviar un mensaje a un objeto es:

```
nombre_objeto.mensaje
```

Por ejemplo, para acceder a las variables instancia o atributos se utiliza la siguiente sintaxis:

```
nombre_objeto.atributo
```

Y para acceder a los métodos o funciones miembro del objeto se utiliza la sintaxis es:

```
nombre_objeto.método( [par1, par2, ..., parN] )
```

En la sentencia anterior **par1**, **par2**, etc. son los parámetros que utiliza el método. Aparece entre corchetes para indicar son opcionales.

Para entender mejor cómo se manipulan objetos vamos a utilizar un ejemplo. Para ello necesitamos la Biblioteca de Clases Java o API (Application Programming Interface - Interfaz de programación de aplicaciones). Uno de los paquetes de librerías o bibliotecas es **java.awt.** Este paquete contiene clases destinadas a la creación de objetos gráficos e imágenes. Vemos por ejemplo cómo crear un rectángulo.

En primer lugar instanciamos el objeto utilizando el método constructor, que se llama igual que el objeto, e indicando los parámetros correspondientes a la posición y a las dimensiones del rectángulo:

```
Rectangle rect = new Rectangle(50, 50, 150, 150);
```

Una vez instanciado el objeto rectángulo si queremos cambiar el valor de los atributos utilizamos el operador punto. Por ejemplo, para cambiar la dimensión del rectángulo:

```
rect.height=100;
rect.width=100;
```

O bien podemos utilizar un método para hacer lo anterior:

```
rect.setSize(200, 200);
```

4.5.- Destrucción de objetos y liberación de memoria.

Cuando un objeto deja de ser utilizado, es necesario liberar el espacio de memoria y otros recursos que poseía para poder ser reutilizados por el programa. A esta acción se le denomina **destrucción del objeto.**

Tema 4: Utilización de objetos.

En Java la destrucción de objetos corre a cargo del **recolector de basura (garbage collector)**. Es un sistema de destrucción automática de objetos que ya no son utilizados. Lo que se hace es liberar una zona de memoria que había sido reservada previamente mediante el operador **new**. Esto evita que los programadores tengan que preocuparse de realizar la liberación de memoria.

El recolector de basura se ejecuta en modo segundo plano y de manera muy eficiente para no afectar a la velocidad del programa que se está ejecutando. Lo que hace es que periódicamente va buscando objetos que ya no son referenciados, y cuando encuentra alguno lo marca para ser eliminado. Después los elimina en el momento que considera oportuno.

Justo antes de que un objeto sea eliminado por el recolector de basura, se ejecuta su método **finalize()**. Si queremos forzar que se ejecute el proceso de finalización de todos los objetos del programa podemos utilizar el método **runFinalization()** de la clase **System**. La clase **System** forma parte de la Biblioteca de Clases de Java y contiene diversas clases para la entrada y salida de información, acceso a variables de entorno del programa y otros métodos de diversa utilidad. Para forzar el proceso de finalización ejecutaríamos:

```
System.runFinalization();
```

5.- Utilización de métodos.

Los métodos, junto con los atributos, forman parte de la estructura interna de un objeto. Los métodos contienen la declaración de variables locales y las operaciones que se pueden realizar para el objeto, y que son ejecutadas cuando el método es invocado. Se definen en el cuerpo de la clase y posteriormente son instanciados para convertirse en **métodos instancia** de un objeto.

Para utilizar los métodos adecuadamente es conveniente conocer la estructura básica de que disponen.

Al igual que las clases, los métodos están compuestos por una **cabecera** y un **cuerpo**. La cabecera también tiene modificadores, en este caso hemos utilizado **public** para indicar que el método es público, lo cual quiere decir que le pueden enviar mensajes no sólo los métodos del objeto sino los métodos de cualquier otro objeto externo.

Dentro de un método nos encontramos el cuerpo del método que contiene el código de la acción a realizar. Las acciones que un método puede realizar son:

- **Inicializar** los atributos del objeto
- **Consultar** los valores de los atributos
- **Modificar** los valores de los atributos
- **Llamar** a otros métodos, del mismo del objeto o de objetos externos

Citas para pensar

Franklin Delano Roosevelt: En cuestión de sentido común tomar un método y probarlo. Pero lo importante es probar algo.

5.1.- Parámetros y valores devueltos.

Los métodos se pueden utilizar tanto para consultar información sobre el objeto como para modificar su estado. La información consultada del objeto se devuelve a través de lo que se conoce como **valor de retorno**, y la modificación del estado del objeto, o sea, de sus atributos, se hace mediante la **lista de parámetros**.

En general, la lista de parámetros de un método se puede declarar de dos formas diferentes:

- **Por valor.** El valor de los parámetros no se devuelve al finalizar el método, es decir, cualquier modificación que se haga en los parámetros no tendrá efecto una vez se salga del método. Esto es así porque cuando se llama al método desde cualquier parte del programa, dicho método recibe una copia de los argumentos, por tanto cualquier modificación que haga será sobre la copia, no sobre las variables originales.
- **Por referencia.** La modificación en los valores de los parámetros sí tienen efecto tras la finalización del método. Cuando pasamos una variable a un método por referencia lo que estamos haciendo es pasar la dirección del dato en memoria, por tanto cualquier cambio en el dato seguirá modificado una vez que salgamos del método.

En el lenguaje Java, todas las variables se pasan por valor, excepto los objetos que se pasan por referencia. En Java, la declaración de un método tiene dos restricciones:

- **Un método siempre tiene que devolver un valor** (no hay valor por defecto). Este **valor de retorno** es el valor que devuelve el método cuando termina de ejecutarse, al método o programa que lo llamó. Puede ser un tipo primitivo, un tipo referenciado o bien el tipo void, que indica que el método no devuelve ningún valor.
- **Un método tiene un número fijo de argumentos.** Los argumentos son variables a través de las cuales se pasa información al método desde el lugar del que se llame, para que éste pueda utilizar dichos valores durante su ejecución. Los argumentos reciben el nombre de **parámetros** cuando aparecen en la declaración del método.

El **valor de retorno** es la información que devuelve un método tras su ejecución.

Según hemos visto en el apartado anterior, la cabecera de un método se declara como sigue:

```
public tipo_de_dato_devuelto nombre_metodo (lista_de_parametros);
```

Como vemos, el tipo de dato devuelto aparece después del modificador public y se corresponde con el valor de retorno.

La lista de parámetros aparece al final de la cabecera del método, justo después del nombre, encerrados entre signos de paréntesis y separados por comas. Se debe indicar el tipo de dato de cada parámetro así:

```
(tipo_parámetro1 nombre_parámetro1, ..., tipo_parámetroN nombre_parámetroN)
```

Cuando se llame al método, se deberá utilizar el nombre del método, seguido de los argumentos que deben coincidir con la lista de parámetros.

La **lista de argumentos** en la llamada a un método debe coincidir en número, tipo y orden con los **parámetros** del método, ya que de lo contrario se produciría un error de sintaxis.

5.2.- Constructores.

¿Recuerdas cuando hablábamos de la creación e instanciación de un objeto? Decíamos que utilizábamos el operador **new** seguido del nombre de la clase y una pareja de abrir-cerrar paréntesis. Además, el nombre de la clase era realmente el constructor de la misma, y lo definíamos como un método especial que sirve para inicializar valores. En este apartado vamos a ver un poco más sobre los constructores.

Un constructor es un método especial con el mismo nombre de la clase y que no devuelve ningún valor tras su ejecución.