Tema 4

Programación orientada a objetos.
Clases y Objetos.

Indice

1.Introducción

- Java es un lenguaje orientado a objetos.
- Toda aplicación Java está formada por un conjunto de clases
- Hay que tener claro los conceptos de:

Clase y Objeto

Clase y Objeto

- Una **clase** es una unidad de software que posee memoria y comportamiento
- Un **objeto** es una instancia (o caso concreto) de una clase
- Una clase sería el molde con el cual se generan los objetos: define sus propiedades y su comportamiento
- Ejemplo: En una aplicación bancaria existirá la clase Cuenta, la clase Cliente, la clase Oficina etc.

  La cuenta XXXX de Pepe Pérez será un objeto de la clase Cuenta
  La oficina 0919 de la calle Sol será un objeto de la clase Oficina

Componentes de una Clase

Una clase se compone de:

- Atributos: son las propiedades de los objetos de la clase.
- Métodos: Procedimientos u operaciones que comparten los objetos de la clase
- Constructores: Procedimientos que se ejecutan en el momento de crear un objeto de la clase. Tienen el mismo nombre de la clase y pueden existir varios.

Diagrama de clases:

Nombre de la clase

Propiedades o Atributos

Métodos

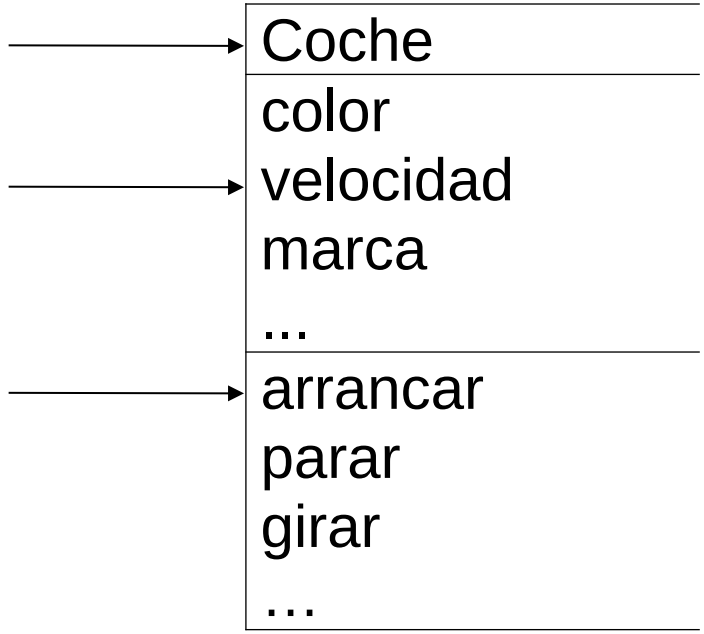
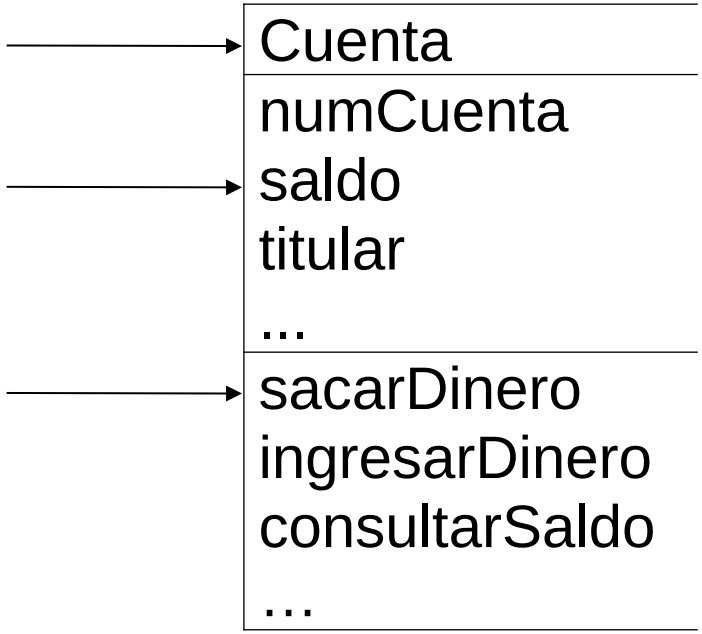


Diagrama de clases:

Nombre de la clase

Propiedades o Atributos

Métodos



2.Características de la POO

La POO se basa en cuatro conceptos

- Abstracción
- Encapsulación
- Herencia
- Polimorfismo

Abstracción:

- Un objeto es capaz de desempeñar una función independientemente del contexto en que éste es utilizado
- Es decir, en cualquier ámbito un objeto tiene las mismas propiedades y se comporta de la misma forma
- Se habla de abstracción en dos sentidos:
  - **Funcional:** Se refiere a que conociendo los métodos que tiene una clase podremos usarlo sin conocer como funcionan internamente. Por ejemplo Math.sqrt sabemos que hace la raíz cuadrada pero no como (ni nos importa).
  - **De Datos:** La manera en que se almacenan o definen los atributos, también es irrelevante para el diseño del objeto. Por ejemplo, el color puede definirse como la palabra rojo, o como un vector RGB (255,0,0).

Encapsulación:

La encapsulación está íntimamente relacionada con la ocultación de la información, definiendo qué partes de un objeto son visibles (interfaz público) y qué partes son ocultas (o privadas).

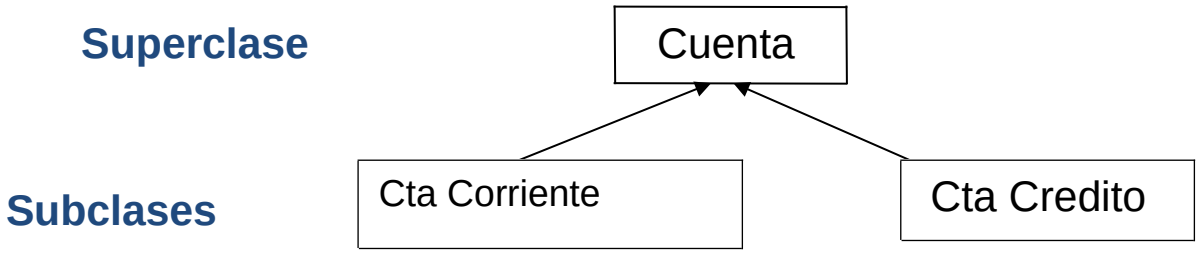
El programador que usa una clase ya creada no tiene por qué saber cómo funcionan internamente los métodos de la clase.

Herencia:

La herencia permite definir a partir de una clase, otras clases relacionadas. Esto supondrá:

- ***Especialización*** de la clase dada (Ej. la clase Cuenta Corriente es una especialización de la clase Cuenta)
- ***Generalización*** de la clase dada (La clase Cuenta es una generalización de la clase Cuenta Corriente).

Ventaja: No es necesario repetir el código, basta con decir que una clase **extiende** a la otra o **hereda** de ella.



La clase “Cuenta” es la clase padre (o superclase). Sus métodos y atributos son heredados por las clases derivadas (hijas o subclases).

Polimorfismo:

El polimorfismo permite que objetos un mismo super(tipo) puedan realizar una operación de forma distinta

Ejemplo: Clase Cuenta tiene dos subclases Cuenta Corriente y Cuenta de Crédito. El método sacar funciona de forma diferente para una cuenta corriente y para una de crédito

3. Declaración de una clase

La forma básica para declarar una clase en Java es:

```
[modifAcceso] class NombreClase
{
    //Atributos o propiedades de la clase
    //(color, velocidad,…etc)
    ...
    //Constructor o constructores
    ...
    //Métodos de la clase
    //(arrancar, parar,…etc)
    ...
}
```

Modificadores de acceso de clase

Son palabras reservadas que se anteponen a la declaración de la clase e indican la visibilidad de la clase.

public	Accesible para todas las clases
	Si no se indica nada la clase será accesible sólo desde clases que estén en el mismo paquete(friendly)

3.1. Atributos

Los atributos de una clase se definen según esta sintaxis:

```
[modifAcceso] [modifAtributo] tipo nombre [= valorInicial];
```

Donde "*nombre*" es el nombre que daremos al atributo, siendo un nombre válido según las normas del lenguaje: por convención, en Java, los nombres de las atributos empiezan con una letra minúscula.

Modificadores de acceso de métodos/atributos:

Indica desde que parte del código se puede acceder a la variable.

public	El método o atributo es siempre visible
private	El método o atributo es privado. Sólo es visible desde dentro de la propia clase
protected	El método o atributo es visible sólo por la propia clase y las que hereden de ella
	El método o atributo es visible pero sólo desde clases que se encuentren en el mismo paquete (friendly)

Modificadores de atributos:

Son características específicas del atributo. Los posibles valores son: static o final (ya veremos que significan)

3.2. Métodos

- Es un bloque de código que realiza una tarea específica.
- Un método pertenece a una clase, y podrá acceder a los atributos de la misma (salvo que sea estático).
- Un método tiene
  - Una lista de parámetros de entrada al método (datos que necesita el método para hacer la tarea específica)
  - Un valor devuelto
  - Puede tener también variables locales al método
- Un método puede no tener ningún parámetro y/o no devolver ningún valor.
- Pueden ser llamado o invocado desde cualquier punto de la aplicación.

Para definir los métodos se emplea la siguiente sintaxis:

```
[modifAcceso] [modifMétodo] tipoRetorno  nombreMétodo (listaParámetros)
{
    //variables locales al método
    //código del método
    return expresión;   //cuando el tipoRetorno es void no es necesario
}
```

Para **modifAcceso** se aplica las mismas normas que para atributos, es decir, public, private, protected, (friendly).

Para **modifMétodo** los posibles valores son: abstract, static o final (ya veremos que significan)

tipoRetorno: Es el tipo de valor devuelto por el método (void si no devuelve ningún dato)

listaParámetros:

El método o función puede tener una lista de argumentos o parámetros. Estos parámetros estarán separados por comas y definidos como:

```
tipo nombreParámetro
```

Los argumentos de los *tipos básicos* o primitivos se pasan siempre por ***valor***. El método recibe una copia del argumento actual; si se modifica esta copia, el argumento original que se incluyó en la llamada no queda modificado.
Los objetos se pasan por ***referencia***, es decir, a través de las referencias se pueden modificar los objetos referenciados.

Valor devuelto por un método:

Si el método tiene algún tipo de retorno, quiere decir que ha de devolver un valor de dicho tipo. Esto se hace mediante la palabra reservada ***return***.

Ejemplo: un método de la clase Círculo que calcula el área debería devolver el dato área (double) que ha calculado

Ejemplo inicial clase Circulo

```
public class Circulo {

     // Atributos de la clase Círculo
     private double radio;
     public final double  PI=3.14;
     // Aquí faltaría el método constructor
     // Métodos de la clase Círculo
     public double calcularArea(){
          double area;
          area=PI*Math.pow(radio,2);
          return area;
     }
     public void setRadio(double radionuevo) {
          if (radionuevo > 0)
               radionuevo = radio;
     }

     public double getRadio() {
          return radio;
     }
}
```

4. Método Constructor

Cuando se crea o instancia un objeto de una clase es necesario dar un valor inicial a sus atributos, es por ello que existe un método especial en cada clase, llamado ***constructor***.

El constructor es llamado automáticamente al crear un objeto de la clase

Para definir los constructores se emplea la siguiente sintaxis:

```
nombreConstructor (listaParámetros)
```

Donde **`nombreConstructor`** debe coincidir con el nombre de la clase.

El constructor, posee unas características especiales:

- Se tienen que llamar igual que la clase.
- No tienen valor de retorno (ni siquiera void).
- Pueden existir varios, que se distinguirán por los parámetros que aceptan (sobrecarga).
- Debe existir al menos 1 constructor.
- Si no existe, se crea un constructor por defecto, que no tiene argumentos.
- Si la clase tiene algún constructor, el constructor por defecto deja de existir.

Ejemplo Constructor Clase Cuenta

```
public class Cuenta {
     // Atributos de la clase
    private String numcta;
    private double saldo;

    /* Primer Método constructor.
    Si no se indica el saldo se considera saldo 0*/
    Cuenta (){
        saldo=0;
    }
    /* Otro Método constructor.
     El objeto Cuenta se crea con  un saldo inicial indicado*/
     Cuenta ( double saldoinicial){
        saldo= saldoinicial;
    }
    public double ingresarCantidad (){
        .....
     }

}
```

Ejemplo Constructor Clase Circulo

```
public class Circulo {

     // Atributos de la clase Círculo
     private double radio;
     public final double  PI=3.14;

     // Método constructor
     Circulo ( double radioinicial)
     {
          radio= radioinicial;
     }
     // Métodos de la clase Círculo
     public double calcularArea(){
          double area;
          area=PI*Math.pow(radio,2);
          return area;
     }

     ....
}
```

5. El Método Principal (main)

- Toda aplicación Java tiene uno y sólo un método denominado main.
- Es el método que busca el interprete para ejecutar en ***primer*** lugar, es decir el punto de entrada de la aplicación
- Es un método público y estático.
- ***void*** indica que no devuelve ningún tipo de datos.
- Tiene como parámetros de entrada un array de String (que veremos más adelante que contiene y como se usa)

6. Creación y manipulación de objetos

- Un objeto es una instancia de una clase. En nuestros programas tendremos objetos que ejecutan métodos.
- Para poder utilizar un objeto hay que
  - Declarar una referencia a la clase.
  - Crear un objeto mediante el operador new, invocando al constructor adecuado.
- Para invocar un método de un objeto

```
nombreObjeto.nombreMetodo( ....);
```

Ejemplo Clase TrabajandoConCuentas

```
public class TrabajandoConCuentas {

    public static void main(String[] args) {

        Cuenta cuenta1;      // declarar la la referencia al objeto
cuenta1
        cuenta1= new Cuenta();
        // crear el objeto cuenta1. LLamará a constructor de la clase
cuenta

        Cuenta cuenta2=new Cuenta();
         // en un solo paso hace lo mismo con una nueva referencia cuenta2

        cuenta1.ingresarCantidad(100);
        cuenta2.ingresarCantidad(200);
        cuenta1.sacarCantidad(50);

    }

}
```

7. El objeto “this”

Java proporciona una referencia al objeto con el que se está trabajando. Esta referencia se denomina **this**, que es el objeto que está ejecutando el método. En los ejemplos que hemos visto hasta ahora, se obviaba esta referencia, puesto que se sobreentiende que nos referimos al objeto está invocando al método.

En algunas ocasiones nos servirá para resolver ambigüedades o para devolver referencias al propio objeto.

Ejemplo:

```
public class Circulo {

    // Atributos de la clase Círculo
    private double radio;
    public final double  PI=3.14;

    // Método constructor
    Circulo ( double radio)
    {
        this.radio= radio;
    }
}
```

8.Los métodos setter y getter.

Los **sets** y **gets** son la forma de acceder a atributos de una clase.

Generalmente, se usan con los atributos privados, ya que a los públicos se puede acceder directamente sin tener que acudir a ellos.

Por uno de los principios de la POO, los atributos de los objetos deben ser siempre "privados" (concepto "encapsulación": no son accesibles desde fuera del objeto, solo el objeto tiene la potestad de usarlos directamente) y se deberán crear métodos públicos para obtener su valor.

- Método "**setter**" para asignar un valor a una variable. La sintaxis es:

```
public void setNombreAtributo(TipoPropiedad valor)

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

- Método "**getter**" para retornar el valor (devolver la información del atributo). La sintaxis es:

```
public TipoPropiedad getNombreAtributo()

public String getNombre(){
    return this.nombre;
}
```

Ejemplo:

```
public class Persona{
    private String nombre;
    private int edad;

    public Persona(){}                          //Constructor

        public void setNombre(String nombre){      //Asignar un valor
            this.nombre = nombre;
        }
        public String getNombre(){              //Devolver el valor
        return this.nombre;
    }
        public void setEdad(int edad){
          if (edad >= 0)
                this.edad = edad;
        }
        public int getEdad(){
            return this.edad;
        }
}
```

Estos métodos lo único que hacen es proporcionar acceso a unos atributos que son privados y que no serían accesibles desde otras clases. Además los set controlarán que no se den a los atributos valores no válidos.

Ejemplo:

```
Persona p = new Persona();

p.setNombre("Pepe");

p.setEdad(23);

System.out.println("Su nombre es: " + p.getNombre());
System.out.println("Su edad es: " + p.getEdad());
```

El método toString

- El método toString es un método que tienen todos los objetos Java (porque todos heredan de Object)
- Devuelve un String con la información de los atributos del objeto.
- Cuando se un objeto se imprime con

```
System.out.println ("El objeto contiene" + o);
```

se está llamando a o.toString() es decir, sería equivalente a

```
System.out.println("El objeto contiene" + o.toString() );
```

- Podemos redefinirlo para que devuelva la información del objeto que nosotros queramos.

9. Atributos y métodos estáticos

Atributos estáticos o Atributos de clase:

Una clase puede tener variables propias de la clase y no de cada objeto. A estas variables se les llama ***variables de clase*** o variables ***static*** y se suelen utilizar para definir constantes comunes para todos los objetos de la clase o variables que sólo tienen sentido para toda la clase.

Las variables ***static*** son compartidas por todos los objetos de la clase (no se crea una copia por objeto, como con las variables de instancia).

El acceso a las variables estáticas se puede realizar desde la clase o desde los objetos

Métodos estáticos o Métodos de clase:

- Para invocar a un método estático no se necesita crear un objeto de la clase en la que se define:
- Si se invoca desde la clase en la que se encuentra definido, basta con escribir su nombre.
- Si se le invoca desde una clase distinta, debe anteponerse a su nombre, el de la clase en la que se encuentra seguido del operador punto (.) <NombreClase>.metodoEstatico

Restricciones de métodos estáticos:

- Solo pueden acceder a variables y métodos estáticos de la misma clase.
- No pueden acceder a this.
- No puede acceder a variables y métodos no estáticos directamente, tiene que crear primero un objeto.
- Los métodos de instancia sí pueden acceder a variables y métodos estáticos.
- Un constructor no puede ser estático.

Variables y Métodos estáticos conocidos:

- Variables estáticas:
  Math.PI
  Integer.MAX_VALUE …etc.

- Métodos estáticos: en la clase Math, todos los métodos son static:

```
public static double pow(double x, double y)
```

y se invocan usando el nombre de la clase:

```
Math.pow(2, 5);
```

10. Atributos final

Una variable declarada como ***final*** no puede cambiar su valor a lo largo de la ejecución del programa, puede ser considerada como una constante.

- Una variable ***final*** tiene que ser inicializada en su declaración.
- Por convenio, el identificador de una variable ***final***, debe escribirse en mayúsculas.
- Es lógico definir las constantes como ***static***, de forma que sean compartidas por todos los métodos y objetos que se creen.

La sintaxis es:

```
final [static] tipo NOMBRE = valor;
final static double PI=3.141592;
```

ACLARACION: Una método también puede ser declarado como ***final.*** Esto significa que no se podrá sobreescribir (lo veremos en el tema de herencia)

11. Normas utilizadas en Java

Por convenio:

- Los nombres de las clases comienzan por mayúscula.
- Los nombres de variables, métodos y objetos comienzan por minúscula.
- Si contienen varias palabras se unen evitando subrayado y separando palabras con mayúsculas.
- Los nombres de las constantes son en mayúscula

12. La clase Object

- En Java existe una clase especial llamada Object. Todas las clases que definamos y las ya definidas heredan de Object.

- Object proporciona una serie de métodos públicos, no static. Aunque tiene más métodos, nos vamos a centrar solamente en tres de esos métodos: equals, hashCode y toString (ya lo hemos visto)

```
boolean equals(Object o);
int hashCode();
String toString();
```

El método equals(Object o) se utiliza para decidir si el objeto es igual al que se le pasa como parámetro.

El método hashCode() devuelve un entero , que es el código hash del objeto. Todo objeto tiene, por lo tanto, un código hash asociado.

¿Qué es un hash y cómo funciona?

https://latam.kaspersky.com/blog/que-es-un-hash-y-como-funciona/2806/

Es importante definir el método equal y el hashcode de cualquier clase. Lo utilizaremos cuando veamos las Colecciones.