U7 – INTRODUCCIÓN A LA ORIENTACIÓN A OBJETOS.

¿Qué es un OBJETO?

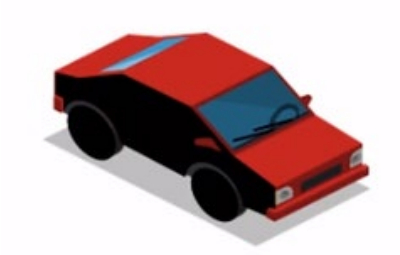
Un objeto es cualquier cosa sobre la que se puede emitir un concepto.

- Un pato
- Un camión
- Una bombilla

No tiene que ser algo tangible sino algo sobre lo que podemos emitir un concepto.

A nivel de programación vamos a poder construir representaciones de los objetos en nuestros programas:

- **Coche de juguete**
  - Plástico / Madera
  - Rojo / Verde / Azul
  - 4 ruedas
  - 1 volante
  - Se mueve hacia adelante y hace atrás



- Se diferencian **dos partes elementales**:
  - ESTRUCTURA / ATRIBUTOS: Material, Color … (¿Cómo está conformado?)
  - COMPORTAMIENTOS/MÉTODOS: Se mueve hacia adelante, hacia atrás… (¿Qué acciones realiza?)

¿Qué es un CLASE?

Si vemos un conjunto de objetos de la imagen todos tienen algo en común, todos son **GLOBOS**.

Así pues, aunque son objetos diferentes los llamamos de una manera común.

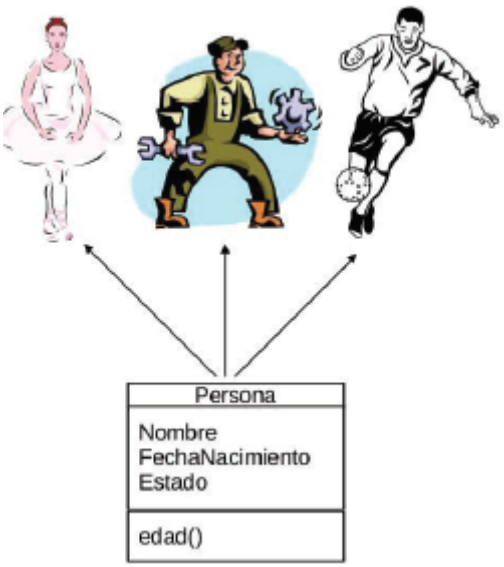
CLASE: Es un molde o plantilla con el que construir objetos de un tipo.

Este molde determinará: CARACTERÍSTICAS o ATRIBUTOS + COMPORTAMIENTOS.

Con este molde podremos construir nuevos objetos.

7.1. INTRODUCCIÓN A POO.

La unidad fundamental de programación en Java es la clase, donde un programa está formado por un conjunto de clases.



Pero, ¿Qué es una clase?

Una clase es una "plantilla" que describe un conjunto de objetos con atributos y comportamientos similares o comunes a un cierto tipo de instancias. En sí es una entidad autónoma y autocontenida.

Una vez tenemos definida una clase podremos "instanciar" objetos de dicha clase en nuestro programa.

Pero, ¿Qué es un objeto?

En Java, un objeto es básicamente una instancia de una clase.

Es un ejemplar de una clase que utilizaremos en nuestro algoritmo y será independiente de otro objeto, aunque sea de la misma clase y tenga un comportamiento similar, representa otro objeto.

Ahora bien, **¿Qué entendemos programación orientada a objetos?**

La programación orientada a objetos o **POO**, trata de descomponer un problema en un pequeño número de entidades o "clases" relacionadas pero independientes de forma que cada entidad puede ser utilizada por separado.

Así pues, si una entidad es demasiado compleja se podría aplicar nuevamente el principio de descomposición creando entidades más simples, fáciles de manejar y que agrupen/definan un tipo de instancias u objetos.

En definitiva: Estructuramos nuestro programa mediante el uso de clases y objetos.

Los objetos serán entidades o elementos independientes construidos con el mismo "molde" o clase. Y estos objetos podrán evolucionar de forma independiente o intercambiar/interactuar entre ellos mediante "paso de mensajes".

7.2. BENEFICIOS DE POO.

POO es una nueva forma de pensar acerca del software que se basa en abstracciones del mundo real.

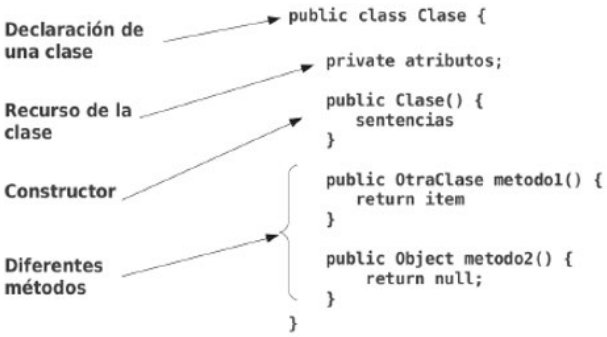
La POO ofrece rendimiento, flexibilidad y funcionalidad para implementaciones prácticas y que permite la reutilización de componentes de software. Beneficios:

- Beneficios por la reutilización.
    - Reduce el tiempo de desarrollo = mejora productividad.
    - Aumenta fiabilidad y eficiencia, minimizando el esfuerzo requerido en el mantenimiento.
- Desarrollo rápido de aplicaciones
- Aplicaciones efectividad y eficiencia, pudiendo aislar el funcionamiento o puntos de fallo.
- Portabilidad. Clase utilizable en diversos proyectos.
- Reduce costes y facilita el mantenimiento.

7.3. ¿CÓMO CREAR UNA CLASE?

Las clases son un tipo de datos definido mediante una estructura o atributos concretos y unas operaciones que se pueden utilizar sobre esos atributos.

La clase es la que nos dice los componentes del ejemplar que vamos a crear, es decir, una clase contiene los atributos y los métodos que conformarán al ejemplar o instancias, de este modo al momento de crear una clase en Java, debemos especificar el tipo y el nombre (como mínimo) de los atributos y adicionalmente debemos especificar (si existen) los métodos o funciones, el tipo de dato que retornan, el nombre y los parámetros que reciben dichos métodos.



7.3.1. PARTES DE UN CLASE:

- Cabecera.
- Campos o atributos:
    - Variables
    - Contantes.
- Métodos:
    - Funciones. Acciones que permite.
    - Constructores. Se utiliza para inicializar nuevos objetos.

```java
public class Animal {
    private String raza;
    private String nombre;
    private int edad;
    private boolean tieneChip;


    public Animal() {
        System.out.println("Nuevo animal ingresado sin información ingresada. Usamos datos por defecto.");
        this.raza="None";
        this.nombre="None";
        this.edad = 0;
        this.tieneChip = false;
    }

    public Animal(String nombre, String raza, int edad, boolean tieneChip) {
        System.out.println("Nuevo animal ingresado con toda la información necesaria");
        this.raza = raza;
        this.nombre = nombre;
        this.edad = edad;
        this.tieneChip = tieneChip;
    }

    public String getRaza() {
        return raza;
    }

    public void setRaza(String raza) {
        this.raza = raza;
    }

    public String getNombre() {
        return nombre;
    }

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

    public int getEdad() {
        return edad;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }

    public boolean isTieneChip() {
        return tieneChip;
    }

    public void setTieneChip(boolean tieneChip) {
        this.tieneChip = tieneChip;
    }

}
```

1. **CABECERA:**

MODIFICADOR DE ACCESO — NOMBRE DE CLASE

```java
public class Animal {


}
```

2. **ATRIBUTOS:**

```java
private String raza;
private String nombre;
private int edad;
private boolean tieneChip;
```

3. MÉTODOS:

3.1. CONSTRUCTOR / ES

```
public Animal() {
    System.out.println("Nuevo animal ingresado sin información ingresada. Usamos datos por defecto.");
    this.raza="None";
    this.nombre="None";
    this.edad = 0;
    this.tieneChip = false;
}

public Animal(String nombre, String raza, int edad, boolean tieneChip) {
    System.out.println("Nuevo animal ingresado con toda la información necesaria");
    this.raza = raza;
    this.nombre = nombre;
    this.edad = edad;
    this.tieneChip = tieneChip;
}
```

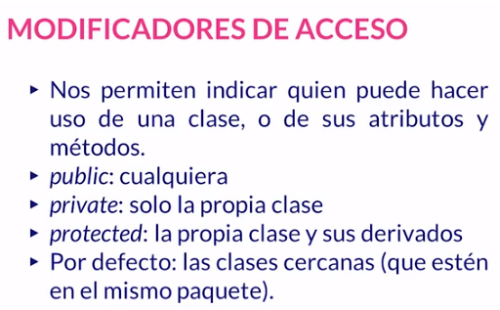
3.2. FUNCIONES.

```
public String getRaza() {
    return raza;
}

public void setRaza(String raza) {
    this.raza = raza;
}

public String getNombre() {
    return nombre;
}

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

public int getEdad() {
    return edad;
}

public void setEdad(int edad) {
    this.edad = edad;
}

public boolean isTieneChip() {
    return tieneChip;
}

public void setTieneChip(boolean tieneChip) {
    this.tieneChip = tieneChip;
}
```

7.4. ENCAPSULADO. VISIBILIDAD DE UNA CLASE.

Aunque creemos un objeto de una clase no todos los elementos de dicha clase son visibles y accesibles.

7.4.1. ENCAPSULADO:

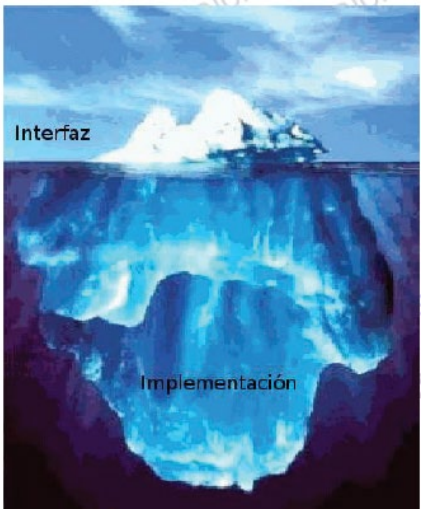


Encapsulado se define como: "proceso de almacenar en un mismo comportamiento los elementos de una abstracción que constituyen su estructura y su comportamiento".

Consiste en ocultar atributos de un objeto al exterior, de forma que los datos y su funcionamiento están protegidos de acciones de otros objetos.

7.4.2. VISIBILIDAD:

Se distingue una parte de implementación y una parte de interfaz.



7.4.2.1. PARTE VISIBLE:

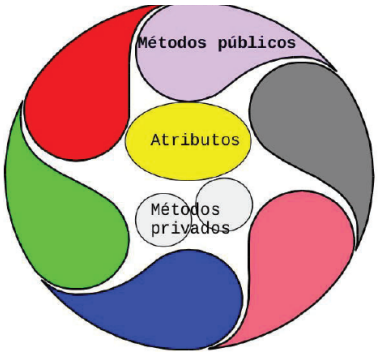
Las clases ofrecen una parte que es visible a otras entidades que se denomina "interfaz". La interfaz se compone de las operaciones que definen el comportamiento e interacción con el objeto de dicha clase.

7.4.2.2. PARTE INVISIBLE:

Las clases poseen una parte privada que solamente es visible dentro de la propia clase. Por lo tanto, no es accesible desde el objeto que se crea sino para el funcionamiento interno de dicho objeto. Por lo tanto, los detalles de implementación quedan ocultos.

7.4.3. ¿COMO AFECTA AL USO DE LOS OBJETOS?

Utilizaremos los modificadores de acceso para definir la accesibilidad o visibilidad de cada uno de los atributos y métodos:



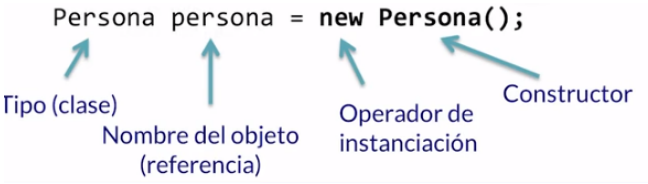
- **VISIBLE** Y ACCESIBLE: **public.**
- **INVISIBLE** Y NO ACCESIBLE DIRECTAMENTE: **private.**

Por lo general todos los atributos se deberán definir como **privados (invisibles)**, es decir, no son accesibles directamente.

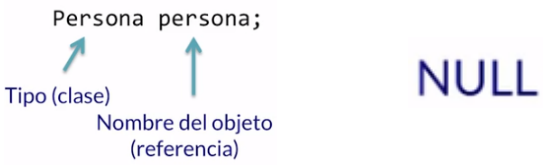
Entonces, ¿cómo podría acceder a un atributo o modificarlo? Para ello en toda clase debemos definir una serie de métodos denominados **"Getters & Setters":** Estos métodos serán públicos (visibles) y accesibles y permitirán modificar o acceder a cada uno de los atributos dado un objeto.

7.5. ¿CÓMO INSTANCIAR UN OBJETO DE UNA CLASE?

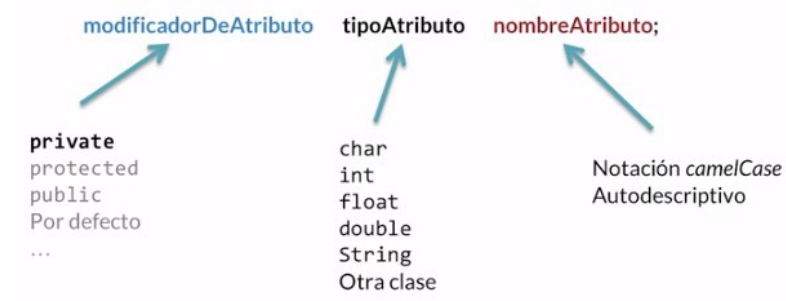
Un objeto es una instancia de una clase creada en el tiempo de ejecución. Mantendrá una estructura de datos formada por tantos atributos como tiene la clase y tantas funciones como métodos públicos tenga definido.



Si se crea un objeto y no se asocia a ningún constructor



7.6. ATRIBUTOS.



<**private**><**static**><**final**><**tipo**> nombre_variable

- **MODIFICADORES:** <static><final>

  final: Convierte la variable en una **constante** que no se puede ser modificado.

  static: **Atributo de clase.** Al contrario que un atributo de objeto:

  - Todas las instancias comparten la misma variable.
  - Se puede usar, aunque no exista objeto de la clase.
  - Accesible mediante: nombre_Clase.nombre_variable

- **MODIFICADORES DE ACCESIBLE:** private, public, protected.

  Mediante estos modificadores se puede modificar los permisos de acceso a dichas variables. Para respetar la “ocultación” y “encapsulación” las variables en las clase se deben definir como “privadas” (siendo accesibles únicamente mediante métodos).

7.7. CONSTRUCTOR.

Método especial invocado con el operador **new**. Se ejecuta exclusivamente en el momento de creación de un objeto y puede servir pera inicializar valores (disponer o no de argumentos).

El constructor es un método especial, es el método que hay que invocar para disponer de un objeto de una clase. Mediante este método se crea un objeto y es donde se pueden pasar atributos iniciales que tendrá el objeto al ser creado.

7.7.1 CREACIÓN DE CONSTRUCTORES DENTRO DE LA CLASE.

```
public Persona(String nombre, String apellidos) {
    this.nombre = nombre;
    this.apellidos = apellidos;
}
```

// OJO1: No devuelve nada por lo que no tiene ni void ni valores de retorno.

// OJO2: Tiene siempre el mismo nombre que la clase.

// OJO3: Puede haber varios en la misma clase (sobrecarga).

NOTA: Si una clase no se define ningún constructor se asume por defecto un constructor sin parámetros de entrada.

// **OJO4:** palabra reservada **this. THIS:**

Nos permite referenciar a los atributos dentro de la clase. De este modo, lo diferenciaremos de forma más clara de un argumento que le pasemos al método.

7.7.2. INSTANCIACIÓN DE UN OBJETO. DECLARACIÓN Y CREACIÓN.

INSTANCIAR: Invocar un constructor y crear un objeto de una clase. **USO** del operador **NEW**.

Ejemplo tenemos una clase de “Persona” e instanciamos dos objetos de dicha clase creando dos objetos persona1 y persona2.

En persona1. Utilizamos el constructor por defecto.

En persona2. Utilizamos el constructor definiendo los atributos del objeto.

```java
public class Persona {

    private String nombre;
    private int edad;

    public Persona() {
    }
    public Persona(String nombre, int edad) {
        super();
        this.nombre = nombre;
        this.edad = edad;
    }

    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public int getEdad() {
        return edad;
    }
    public void setEdad(int edad) {
        this.edad = edad;
    }

}
```

```java
public static void main(String[] args) {

    Persona persona1 = new Persona();      // Constructor por defecto. El objeto tendrá los valores por defecto en sus atributos.
    Persona persona2 = new Persona("Manuel", 11);  // Constructor donde se definen los valores de cada unos de los atributos.

}
```

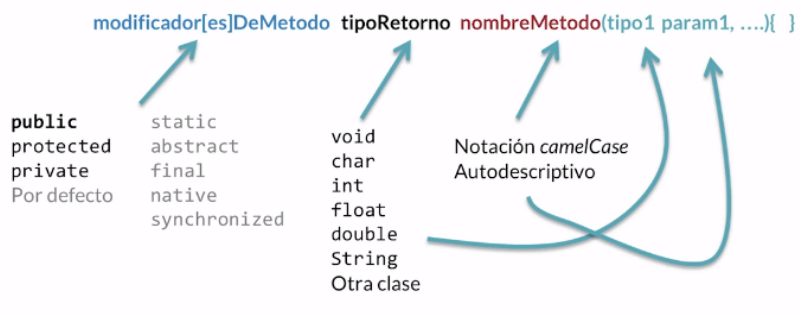
7.7.3. TIPOS DE CONSTRUCTORES.

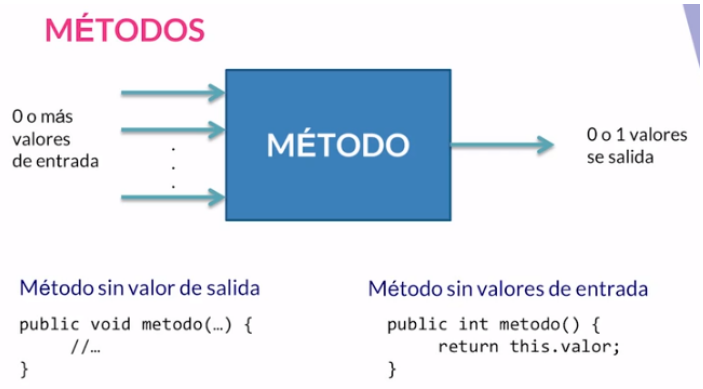
1. **CONSTRUCTOR POR DEFECTO.**
2. **CONSTUCTOR CON PARÁMETROS.**
3. **CONSTRUCTOR DE COPIA.**

```java
public class Cuenta {

    private String nombre;
    private String numeroCuenta;
    private double tipoInteres;
    private double saldo;

    //Constructor por defecto
    public Cuenta() {
    }

    //Constructor con parámetros
    public Cuenta(String nombre, String numeroCuenta, double tipoInteres, double saldo) {
        this.nombre = nombre;
        this.numeroCuenta = numeroCuenta;
        this.tipoInteres = tipoInteres;
        this.saldo = saldo;
    }

    //Constructor copia
    public Cuenta(final Cuenta c) {
        nombre = c.nombre;
        numeroCuenta = c.numeroCuenta;
        tipoInteres = c.tipoInteres;
        saldo = c.saldo;
    }
```

7.8. CREACIÓN DE MÉTODOS.

7.8.1. DECLARACIÓN DE MÉTODOS.






```
<tipo_acceso><tipo_dinamico_o_no><tipo_dato>nombre_metodo(<tipo_parametro> parametro)
```

Un método generalmente usa toda esa estructura solo exceptuando la declaración de si es dinámico u estático.

La primera parte de creación de un método se refiere a el tipo de acceso que puede ser:

- **protected**, acceso protegido, accesible por clases dentro del paquete.
- **private**, acceso solo de modo interno de la clase
- **public**, acceso desde una instancia externa de la clase.

La segunda parte se refiere a el uso del método Java, si es estático lo cual significa que el método sería accesible desde fuera de la clase sin necesidad de instanciar la clase.

- **static**, el acceso al método es estático.

En tercer lugar, el tipo de dato es dependiente de lo que se desea como resultado del método como puede ser por ejemplo **void** si nuestro método no tiene salida alguna, o un tipo de dato específico como puede ser **double** o **int** si es una salida de tipo numérico.

Luego, el nombre de método de preferencia debe ser escrito en *notación camelCase*

Por último, la creación del método no en todos los casos es necesario argumentos pero si deseamos usar algún argumento, cada argumento deberá tener su tipo de dato y nombre de argumento.

8.5.3. UTILIZACIÓN DE MÉTODOS.

Un método se invoca haciendo referencia al nombre del objeto, el nombre del método y los argumentos requeridos.

```
Persona persona1 = new Persona();
persona1.mostrarInformacion();
```

7.9. MÉTODO ESPECIALES:

7.9.1. UTILIZACIÓN DE PROPIEDADES: GETTERS & SETTERS.

Se llama **propiedades** a un tipo especial de **métodos** que **permita acceder y modificar** a los **atributos** de un objeto, que por defecto deberían ser privados.

<< CLAVES PARA IMPLEMENTAR LA ENCAPSULACIÓN >>

Getters & Setters:

- getVariable

  Devuelve el valor actual de la variable del objeto.

- setVariable

  Modifica el valor de un variable o parámetro del objeto.

OJO: En caso de retorna un boolean se suele utilizar **is**Variable()

```
Persona persona1 = new Persona();
persona1.setEdad(11);
System.out.println(persona1.getEdad());
```

7.9.2. UTILIZACIÓN DE toString()

Sirve para representar todo el objeto como una cadena (se auto genera con el IDE)

```
@Override
  public String toString() {
    return "Factura [numero=" + numero + ", concepto=" + concepto + ", importe=" + importe + "]";
  }
```

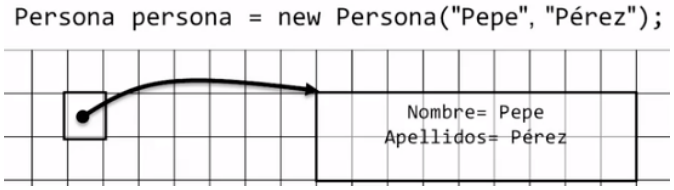
7.10. ESTADO DE UN OBJETO.

El estado de un objeto viene dado por el valor de los atributos en un momento dado. El cual puede variar/evolucionar con el paso del tiempo.

Cuando se crean varios objetos de la misma clase, todos ellos aunque se compongan de la misma clase o “plantilla” son independientes entre sí, es decir su estado es independiente del resto de objetos.

7.11. CICLO DE VIDA DE UN OBJETO.

Al crear un objeto de una clase estamos creando una entidad donde el objeto es la referencia a un espacio de memoria donde está la información de la clase.



Un objeto va a vivir durante todo el bloque de código en el que se ha instanciado.

Cuando se termina el ámbito, el objeto será enviado a la basura. (reciclaje) Por lo tanto en Java no tenemos que destruir un objeto creado y liberar le memoria, sino que Java lo gestiona todo por nosotros.

Uso "**barbage collector**" – recolector de basura.

7.12. CLASES ENVOLTORIO.

Java dispone de una clase por cada uno de los tipos de datos primitivos. Que "envuelven" cada uno de los tipos de datos primitivos vistos.

Tipo de dato primitivo	Clase envoltorio	Tipo de dato primitivo	Clase envoltorio
boolean	Boolean	int	Integer
char	Character	long	Long
byte	Byte	float	Float
short	Short	double	Double

¿Para qué utilizar las clases envoltorio? https://www.geeksforgeeks.org/java-lang-integer-class-java/

- Tiene una serie de métodos interesantes. Como la conversión de tipos de datos.
- Uso en colecciones u otros tipos de contenedores de objetos.

NÚMEROSAS CLASES EN JAVA: https://docs.oracle.com/javase/8/docs/api/