U8. UTILIZACIÓN DE OBJETOS.

Como hemos visto en la unidad anterior la programación orientada a objetos tiene su base en la definición de clases.

Mediante el uso de clases, extendiendo el concepto de tipos de datos, se aumenta considerablemente la flexibilidad de los desarrollos, ya que permiten usar estructuras heterogéneas que agrupen argumentos de diferentes tipos.

Además, algo clave es que propicia la reutilización de software.

8.1. CLASE.

Unidad fundamental del desarrollo orientado a objetos. Además:

- Ocultan la implementación de las operaciones.
- Permiten ampliar los tipos de datos básicos.

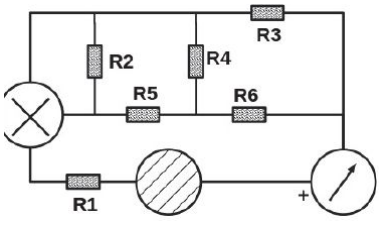
8.1.1. SIMIL CAJA DE LUCES.

- Se define una caja de luces con tres botones: rojo, azul y amarillo.
- Se define las especificaciones/requisitos.
- Se desarrolla el concepto de clase:

Constructor de la caja de luces: operación que instanciará una caja de luces a partir de la descripción de la misma.	caja()
Presionado de los botones: operaciones que realizan una acción en la caja de luces.	pulsarRojo() pulsarAmarillo() pulsarAzul()
Información del estado de las diferentes bombillas, es decir, si está encendida o no. Estas operaciones permiten ver el efecto de las operaciones del presionado en los botones.	boolean estaEnc1() boolean estaEnc2() boolean estaEnc3()

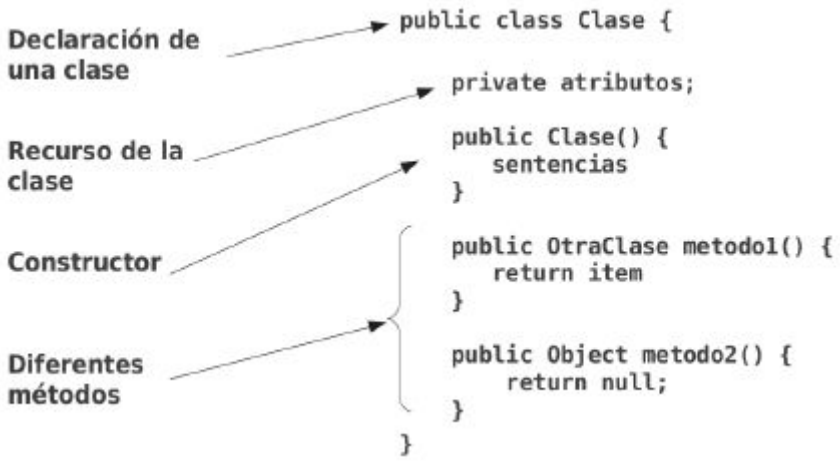
De este modo se define la interfaz con la que interactuar con nuestra caja de luces. Abstrayendo al usuario de la complejidad de su implementación, por ejemplo, el circuito electrónico requerido.

El programador no necesita saber cómo funciona o está implementado tan sólo como crear una instancia de la clase y como utilizarla.



8.2. ESTRUCTURA Y MIEMBROS DE UNA CLASE.

Los miembros de una clase son los **atributos**, los cuales definen la información que constituyen el r**ecurso de la clase**, y la **interfaz** que son los **métodos** mediante interactuar con dicha clase.



Definir una clase implica:

- Definir los **atributos** que se van a utilizar como recursos de la clase.
- Definir los **métodos** para operar sobre los atributos.

8.3. CREACIÓN DE 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).

8.4. CONSTRUCTOR.

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.

8.4.1 CREACIÓN DE CONSTRUCTOR EN LA CLASE.

El constructor es un método que tiene el mismo nombre que la clase y no puede devolver ningún valor.

Se puede usar más de un constructor, pero con diferentes tipos o números de parámetros. **"SOBRECARGA"**.

```
public Clase();
public Clase(int n);
public Clase(String s);
public Clase(int n, String s);
public Clase(String s, int n);
```

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

8.4.2. INSTANCIACIÓN DE UN OBJETO. DECLARACIÓN Y CREACIÓN.

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

```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.

}
```

Nota: ¿Que es "super()"? Se utiliza en el caso de las herencias para llamar a un método (en este caso el constructor) de la clase padre de la que hereda. Todos los constructores llaman por defecto al constructor de la clase superior a través de una llamada a super(), y se podría modificar en caso de que la clase hija tendrá que introducir algún valor por argumento para el constructor de la clase padre.

8.5. CREACIÓN DE MÉTODOS.

8.5.1. 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 de datos
- **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();
```

8.6. UTILIZACIÓN DE PROPIEDADES.

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

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());
```