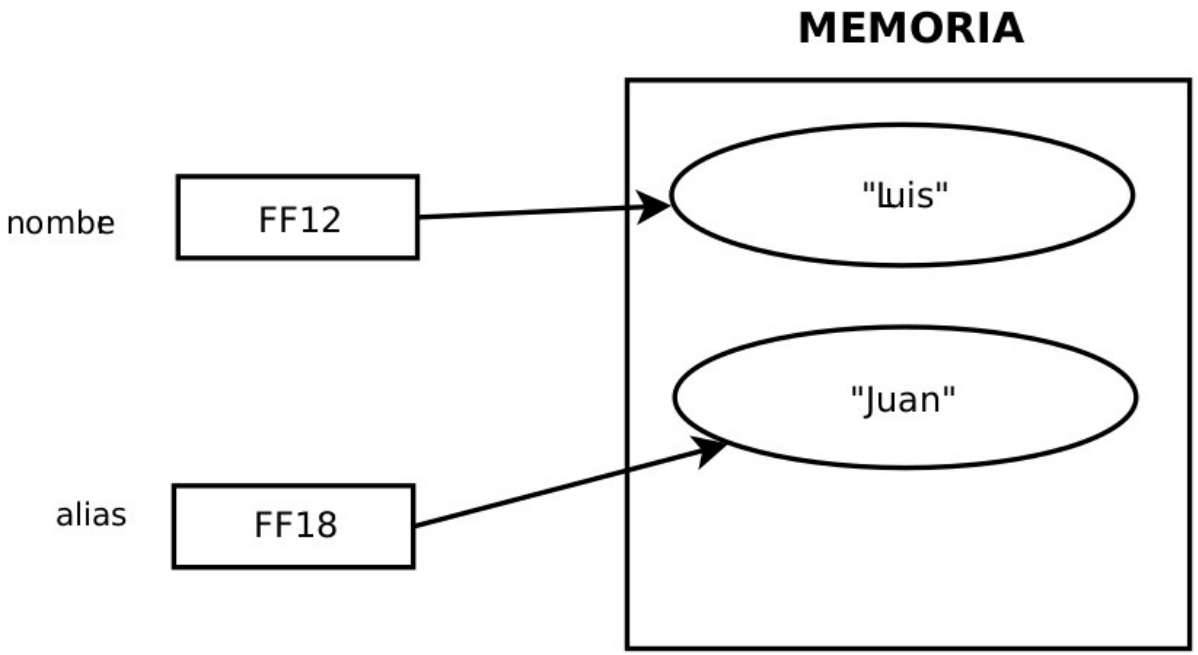
PROGRAMACIÓN

Unidad 5. Utilización de objetos



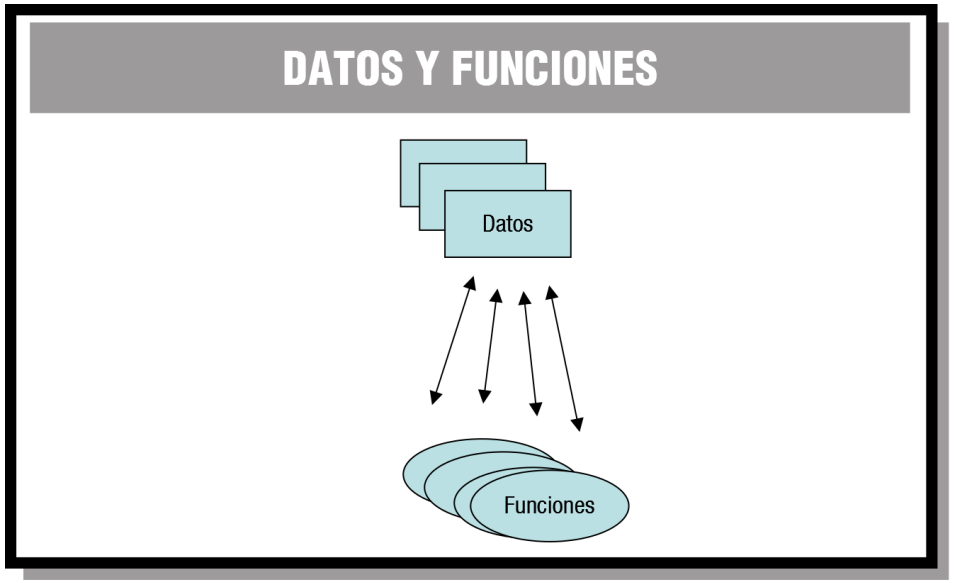
PROGRAMACIÓN ORIENTADA A OBJETOS CON JAVA

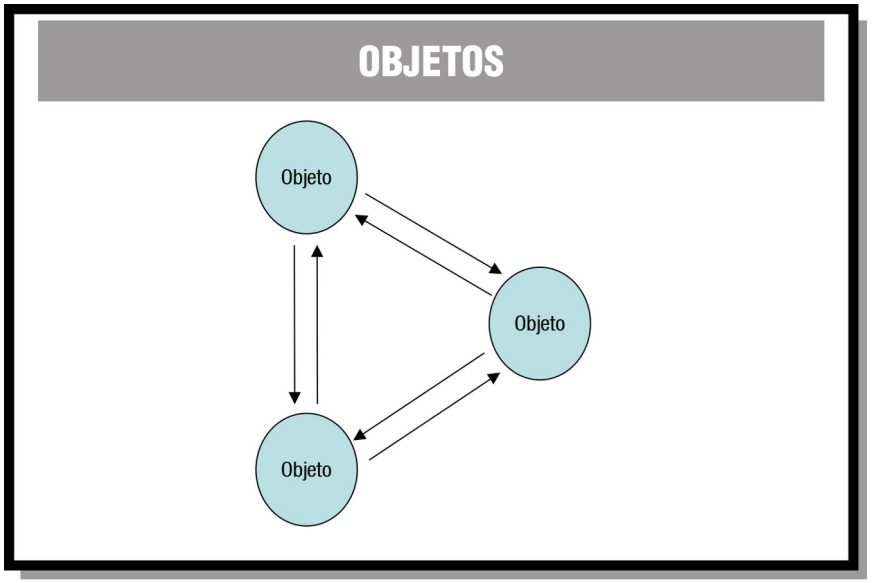
José L. Berenguel

Tabla de Contenidos

Introducción

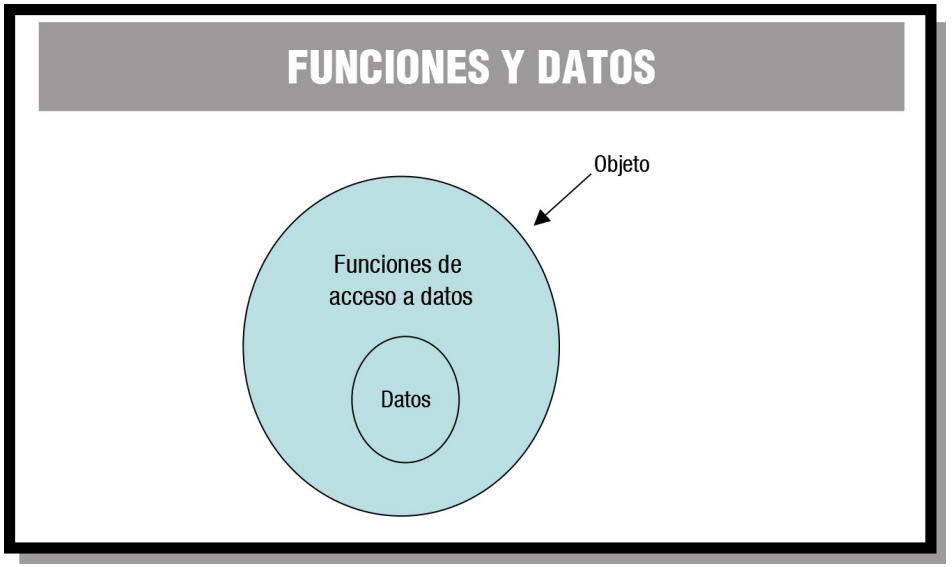
- **Programación Orientada a Objetos** (*Object Oriented Programming*): paradigma que intenta emular la representación y el funcionamiento de los objetos del mundo real.
- **Programación estructurada**: paradigma que se basa en el procesamiento de datos sin que exista una relación semántica entre ellos.





Introducción

- **Objeto**: entidad que agrupa propiedades o atributos (datos) y métodos (código).
  - El valor de las propiedades establece las características propias de un objeto (**estado**).
  - Los valores de las propiedades (estado) se modifica a través de sus operaciones o métodos (**comportamiento**).

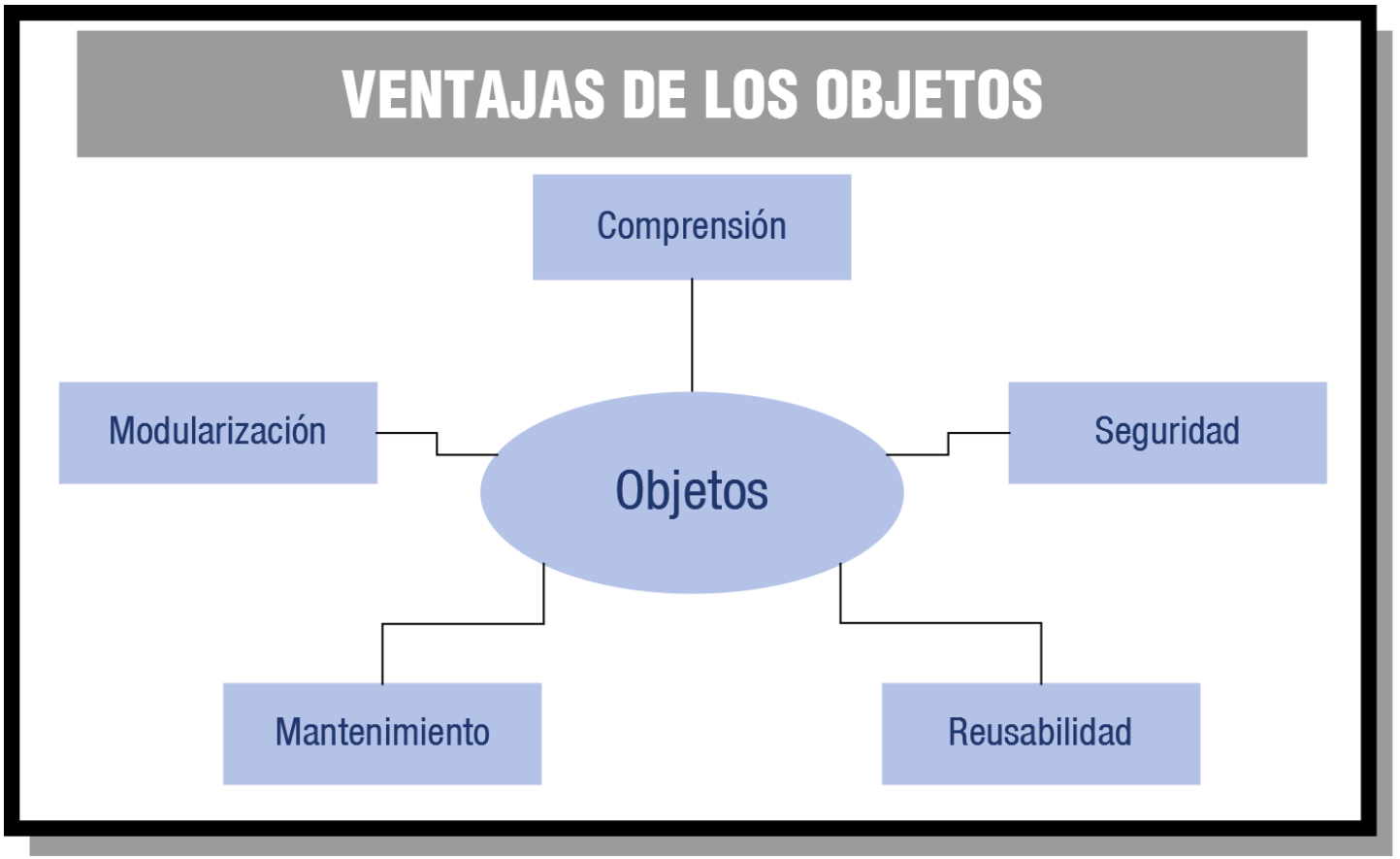


Introducción

- **Clase**: plantilla en la que se definen las características que tendrán los objetos de esta clase.
- **Conceptos** de POO:
  - **Abstracción**. Las características de los objetos se definen sin importar cómo se escribe el código.
  - **Encapsulación**. Se ocultan las partes internas que otras partes de la aplicación no necesitan conocer.
  - **Modularidad**. Las clases permiten la reutilización de código y su modificación de forma mucho más sencilla y con menos dependencias de código.
  - **Herencia**. Permite crear jerarquías de clases en las que estas pueden heredar funcionalidades de sus ancestros.
  - **Polimorfismo**. Permite que clases con un mismo ancestro tenga comportamientos distintos ante una misma funcionalidad.

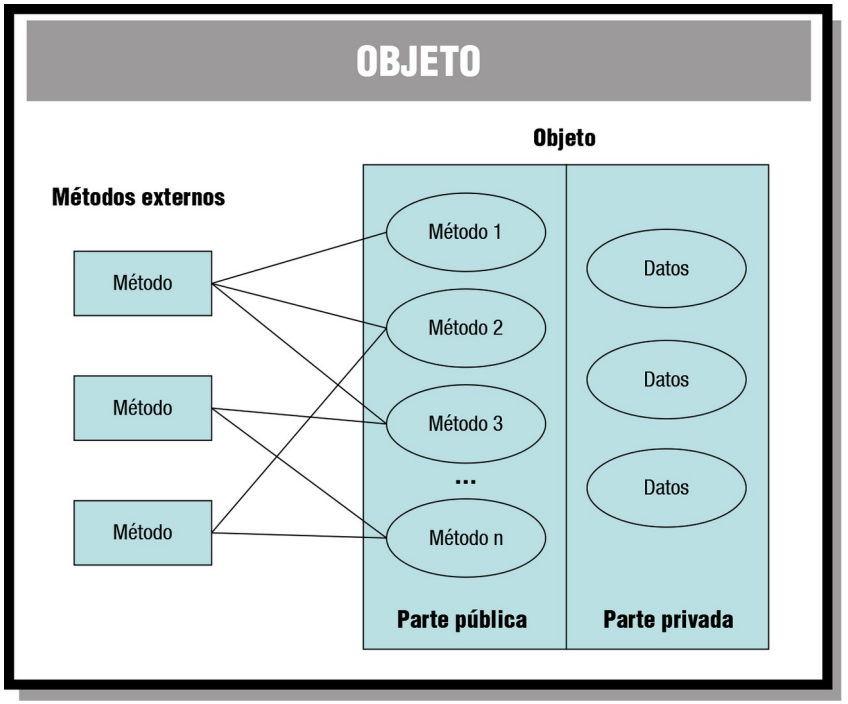
Introducción

- Beneficios de la POO.



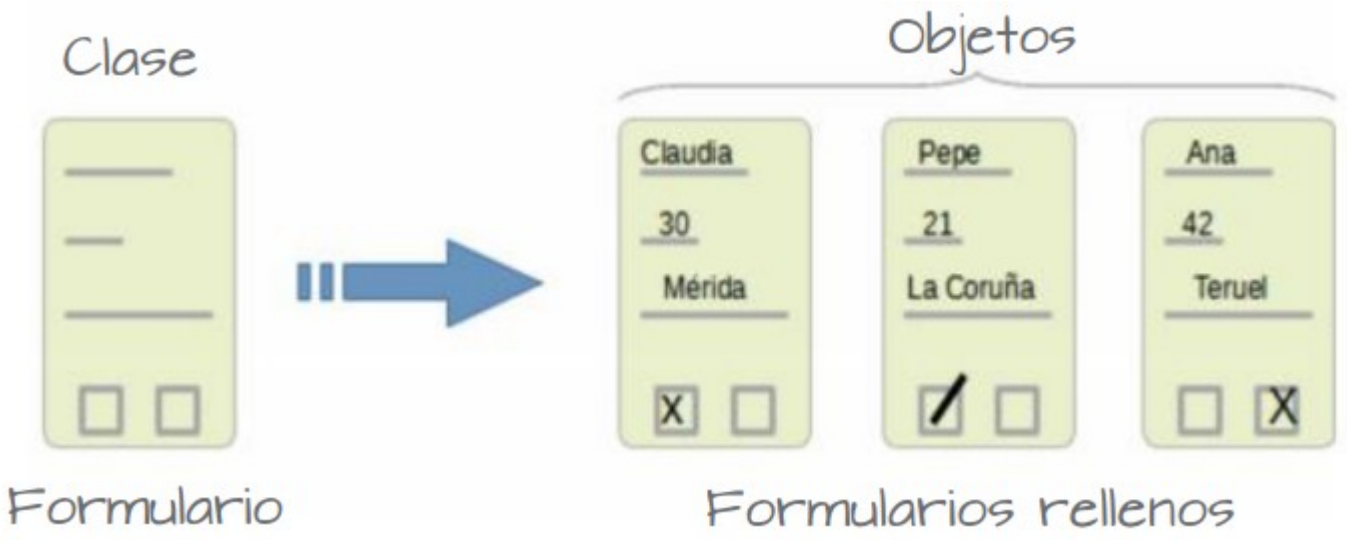
Clases y objetos

- Los objetos tienen unas características que los identifican:
  - **Identidad**. Permite diferenciar un objeto de otro (dirección de memoria, identificador, etc.).
  - **Estado**. Los valores de los atributos del objeto en un momento dado determinan su estado.
  - **Comportamiento**. Las acciones que se pueden realizar sobre el objeto para modificar su estado, a través de sus métodos.



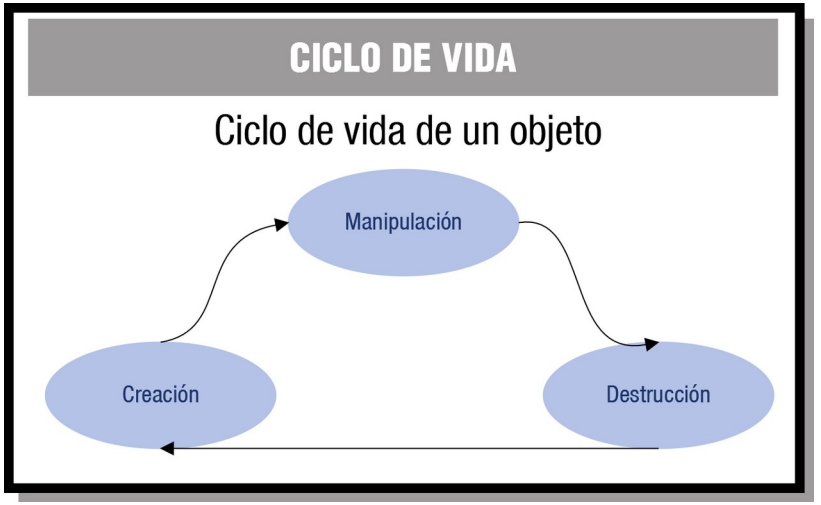
Clases y objetos

- Una clase es una plantilla en la que se definen las características (atributos y métodos) de los objetos de dicha clase.



Ciclo de vida de los objetos

- Los objetos pasan por las siguientes fases:
  - **Creación**. Se reserva la memoria y se inicializan los atributos.
  - **Manipulación**. Se hace uso de sus operaciones y se consulta su estado.
  - **Destrucción**. Se elimina la memoria reservada.



Ciclo de vida de los objetos

- Los pasos en el lenguaje de programación son:
  - **Declaración**. Se declara una variable de la clase del objeto.
  - **Instanciacion**. Se llama al operador **new** con el constructor de la clase (puede haber más de un constructor).
  - **Manipulación**. Se realizan operaciones a través de sus métodos.
  - **Destrucción**. El recolector de basura libera la memoria cuando el objeto ya no tiene ninguna referencia que lo apunte.

```
String saludo = new String("Bienvenido a Java");
String s; //s vale null
s = saludo; //asignación de referencias
s = saludo.toUpperCase(); //manipulación a través de un método
```

Definición de clases y creación de objetos

- La definición de una clase en Java consta de:
  - **Cabecera**: se compone de una serie de modificadores, la palabra reservada ***class*** y el identificador de la clase.
  - **Cuerpo**: se definen los atributos y métodos que tendrá la clase. Se definen también unos métodos especiales llamados constructores.

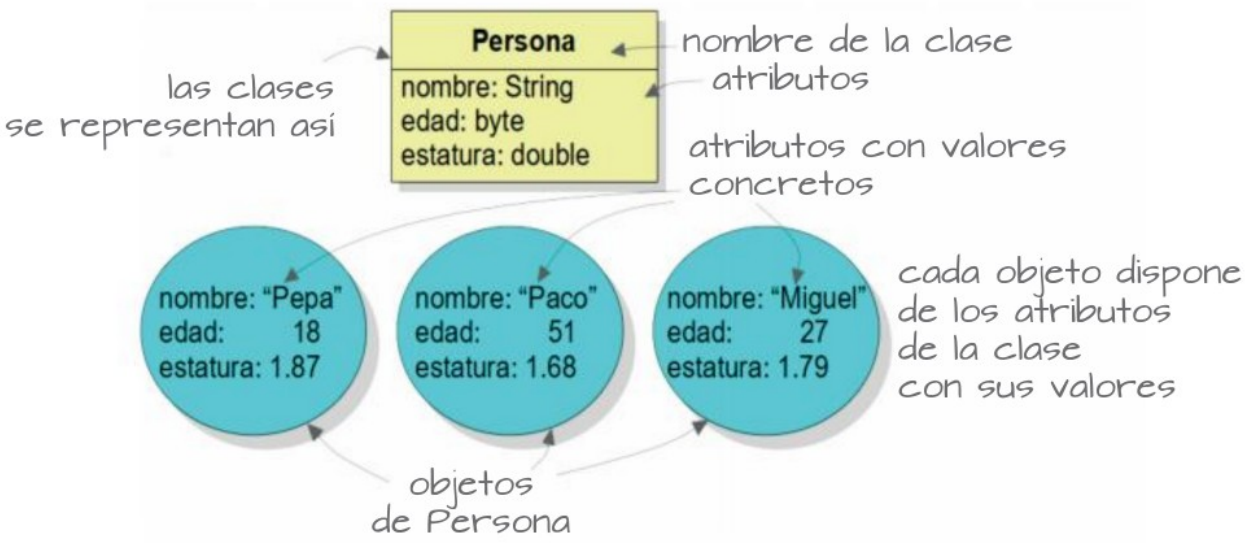
```
//Cabecera de la clase
[public] class MiClase{
    //Declaración de atributos

    //Declaración de métodos y constructores

}
```

Definición de clases y creación de objetos

- Ejemplo de clases y objetos




```
public class Persona{
     //Declaración de atributos
     private String nombre;
     private byte edad;
     private double estatura;

     //Declaración de métodos y constructores

}
```

Modificadores de acceso

- ***private***: restringido al interior de la clase
- (***ninguno***): acceso por defecto (***package-private***), visible para todas las clases del mismo paquete.
- ***protected***: clases del mismo paquete o subclases de ella (independientemente del paquete).
- ***public***: visible desde cualquier clase.

	private	(default)	protected	public
clase	NO	SI	SI	SI
método	SI	SI	SI	SI
atributo	SI	SI	SI	SI
variable local	NO	NO	NO	NO

Encapsulación de los atributos

- Permite proteger datos sensibles y facilita el mantenimiento de las aplicaciones.
- **Miembros *public***: aquellos que se van a exponer al exterior. Se denomina **interfaz del objeto**.
- Los atributos suelen tener acceso privado y son accedidos y modificados a través de métodos *get*/*set*.

```
public class Rectangulo{
    //atributos declarados públicos
    public int alto, ancho;
    //Métodos de la clase
}

// Problema al no encapsular los atributos
Rectangulo r = new Rectangulo();
r.alto=-5;
```

Encapsulación de los atributos

```
public class Rectangulo{
    private int alto, ancho;
    public void setAlto(int alto){
        if(alto>0)
            this.alto=alto;
    }
    public int getAlto(){
        return this.alto;
    }
    //resto de métodos de la clase
}
```

```
Rectangulo r=new Rectangulo();
r.setAlto(5);
r.setAlto(-3); //no tendría efecto la modificación
```

this es una referencia a la propia clase, y permite acceder a los miembros (atributos y métodos) de la misma, evitando la confusión con variables locales del mismo nombre.

Constructores

- Un **constructor** es un método especial que es ejecutado en el momento de la creación del objeto (llamada a ***new***).
- Se utilizan para inicializar los atributos del objeto.
- Características:
  - El **nombre del constructor es el mismo que el de la clase**.
  - **No puede tener tipo de retorno**, ni siquiera *void*.
  - Se puede sobrecargar.
  - El **constructor por defecto** es el que no tiene parámetros.
  - La clase debe tener **al menos un constructor**.
  - El constructor copia es aquel que recibe por parámetro un objeto de la misma clase y crea un objeto con los mismos valores.
  - Si no se especifica, Java provee uno por defecto.
  - Se puede llamar a otro constructor de la misma clase con ***this()***.

Constructores

```
public class Punto{
    private int x,y;
    public Punto(int x, int y){
        this.x=x;
        this.y=y;
    }
    public Punto(int v){
        this.x=v;
        this.y=v;
        //O también llamando a otro constructor de la clase:
        //this(v, v);
    }
    //resto de métodos de la clase
}
```

```
Punto p1 = new Punto(3,5);
Punto p2 = new Punto(6);
Punto p3 = new Punto(); //error de compilación
```

Constructores

Ejercicio: Completa el código de la siguiente clase

```
public class Persona{
     //Declaración de atributos
     private String nombre;
     private byte edad;
     private double estatura;

     //Declaración de constructores
     public Persona(){

     }

     public Persona(String nombre, byte edad, double estatura){

     }

     public Persona(Persona p){

     }
     //Declaración de métodos get/set para los atributos


}
```

Definición de métodos

- Un método encapsula una porción de código que va a ser reutilizado con frecuencia.
  - **[*modificadores*]**: opcionales. Se estudian más adelante.
  - ***tipo_retorno***: tipo de dato que devuelve (primitivo u objeto). Si no devuelve ningún valor se indica con ***void***.
  - ***parámetros***: datos que recibe el método como argumentos en la llamada. Similar a la declaración de variables.
  - ***return***: sentencia donde el método devuelve el dato que retorna. No aparece si el método devuelve ***void***.

```
//cabecera del método
[modificadores] tipo_retorno nombre_metodo(tipo parametro1, tipo parámetro2, … ){
     //cuerpo del método
     [return valor]
}
```

Sobrecarga de métodos

- **Varios métodos con el mismo nombre** en una misma clase.
- **Cada versión del método debe distinguirse en el número o tipo de argumentos** obligatoriamente. Puede cambiar el tipo de retorno (opcional).
- El compilador identifica la versión del método según los argumentos utilizados en la llamada.

```
//VÁLIDOS
public void calculo (int k){...}
public void calculo (String s){...}
public long calculo (int k, boolean b){...}

//NO VÁLIDOS
public int calculo(int k){...}
```

Variables *static*

- Son variables que pertenecen a la clase no al objeto.
- Una única copia compartida por todos los objetos.
- Se inicializan una única vez, al principio de la ejecución.
- **Se acceden directamente a través del nombre de la clase**. No necesitan ninguna instancia de objeto.
- Sintaxis:
  - **Declaración**: *<modificador> static <tipo> identificador;*
  - **Acceso**: *<Nombre_Clase>.<identificador>;*

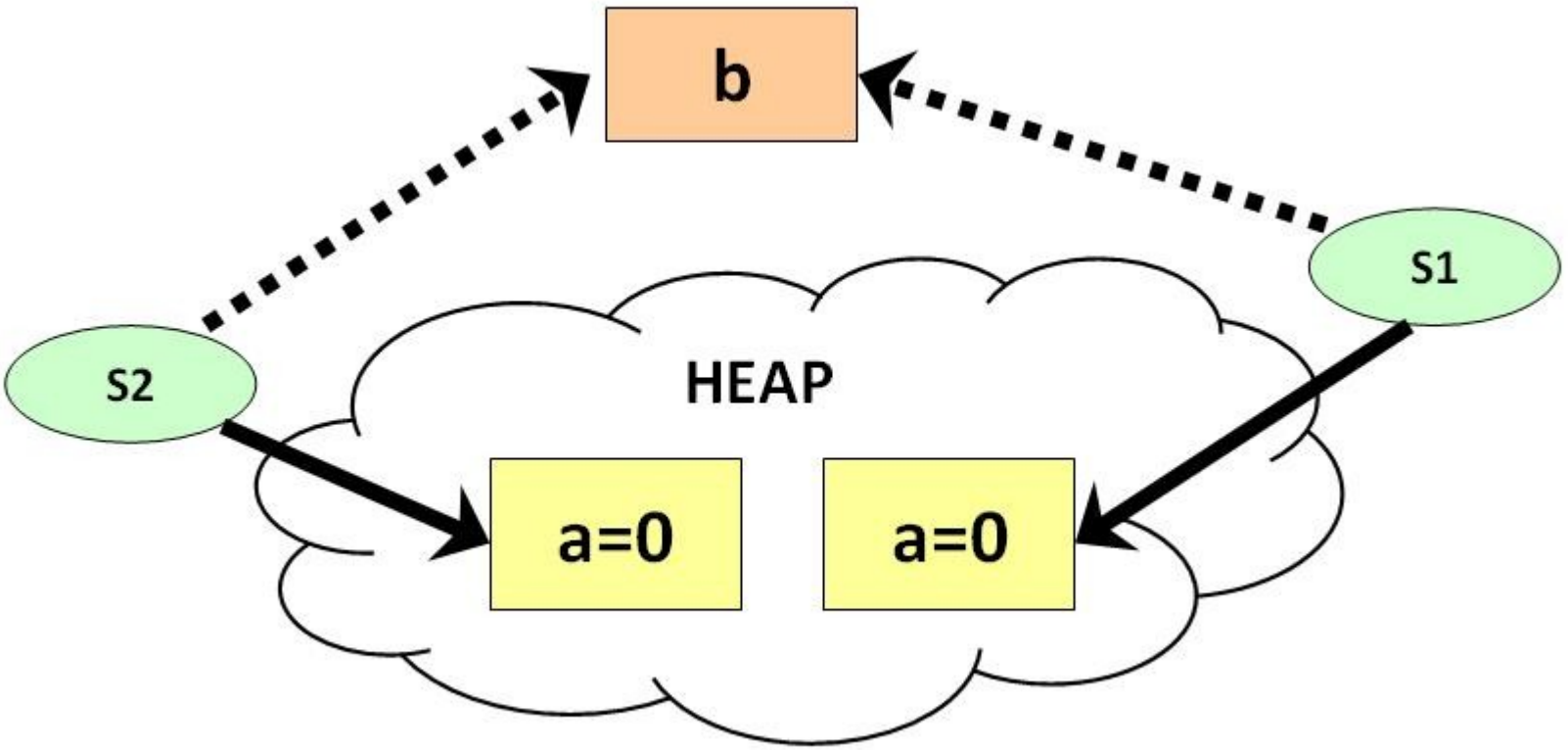
Métodos *static*

- Es un método que **pertenece a la clase no al objeto**.
- Un método *static* **solo puede acceder miembros *static***.
- Se accede a través del nombre de la clase. No es necesaria una instancia de un objeto.
- Desde un método *static* no se puede hacer referencia a ***this*** o ***super*** (se explica en temas más avanzados).
- Sintaxis:
  - **Declaración**: *<modificadores> static <tipo> identificador(..);*
  - **Acceso**: *<Nombre_Clase>.<identificador>(parámetros);*

Ejemplo atributos *static*

```
class Estudiante {
    private int a;
    private static int b; //inicializado a 0 al cargar la clase
  Estudiante(){
    this.a=0;
    b++; //Constructor incrementa la variable static b
    //Estudiante.b++; //Preferible esta notación
  }
  public void mostrarDatos(){
    System.out.println("Valor de a = "+a);
    System.out.println("Valor de b = "+b);
  }
  //Probar este método
  //public static void incrementar(){
    //a++;
  //}
}
class Demo{
   public static void main(String args[]){
     Estudiante s1 = new Estudiante();
     s1.mostrarDatos();
     Estudiante s2 = new Estudiante();
     s2.mostrarDatos();
  }
}
```

Ejemplo atributos *static*



Ejemplo métodos *static*

```
public class EjemploFunciones{
     public static void main(String args[]){
          double s = Math.sqrt(5); //acceso estático de un método en la clase Math
          String nombre=”José”;
          nombre.length(); //acceso a un método no-estático de un objeto String
          double r = cambiaSigno(-5); //también EjemploFunciones.cambiaSigno
          double mayor = Misfunciones.mayor(-5,5);
     }
     static double cambiaSigno(double d){
          return (-1)*d;
     }
}
```

```
public class MisFunciones{
     public static double mayor(double a, double b){
          return (double m=(a>b)?a:b);
     }
}
```

Bloque *static*

- Es un bloque en el interior de una clase Java que se ejecutará cuando la clase se cargue en la JVM.
- Sirve para inicializar miembros static, al igual que el constructor inicializa miembros de instancia.

```
class Test{
    static {
        //Código de inicialización de miembros static
 }
}
```

http://www.javatutorialhub.com/java-static-variable-methods.html

Unidad 5. Utilización de objetos

DUDAS Y PREGUNTAS