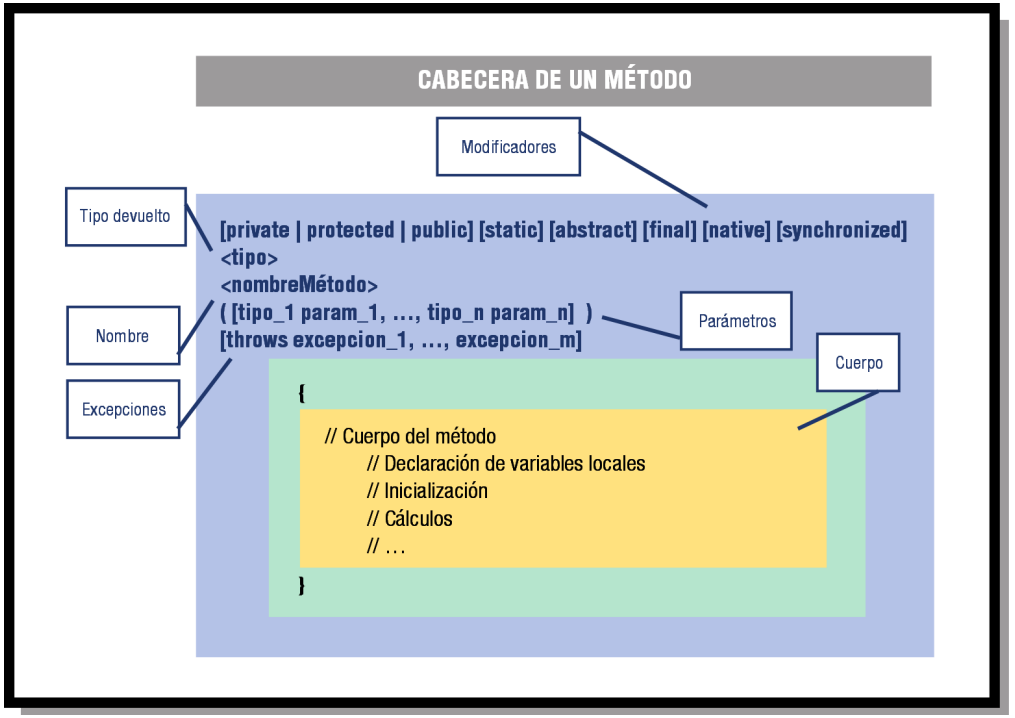
PROGRAMACIÓN

Unidad 5. Desarrollo de clases



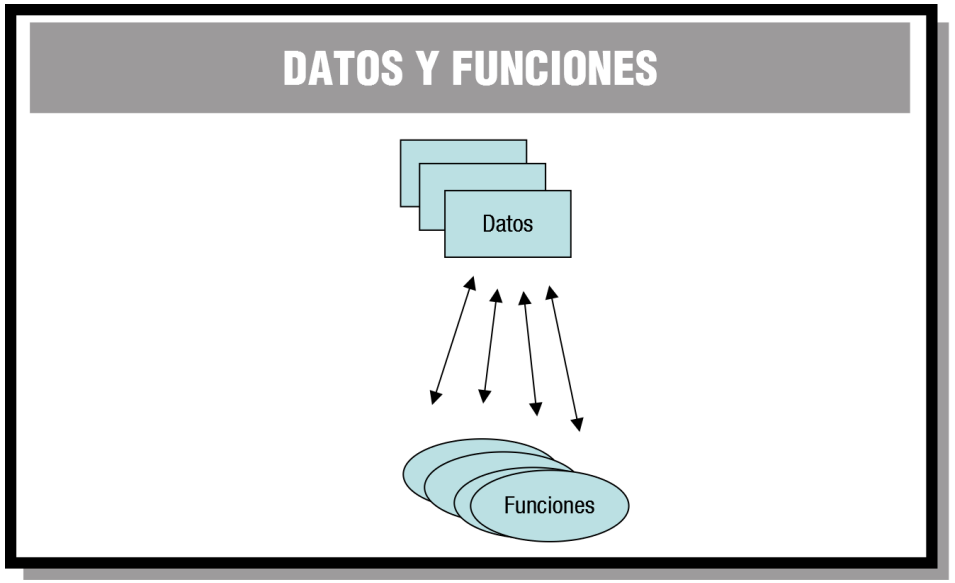
PROGRAMACIÓN ORIENTADA A OBJETOS CON JAVA

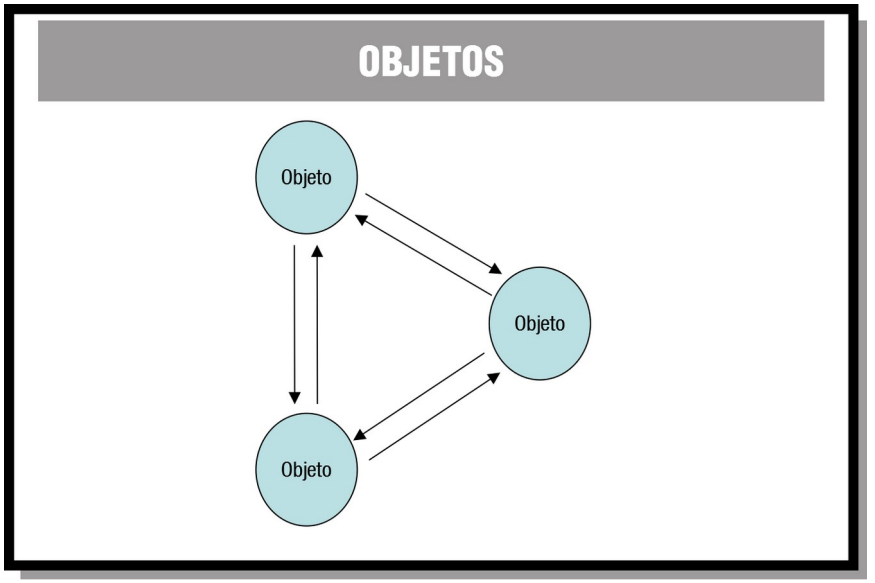
José L. Berenguel

Tabla de Contenidos

Introducción

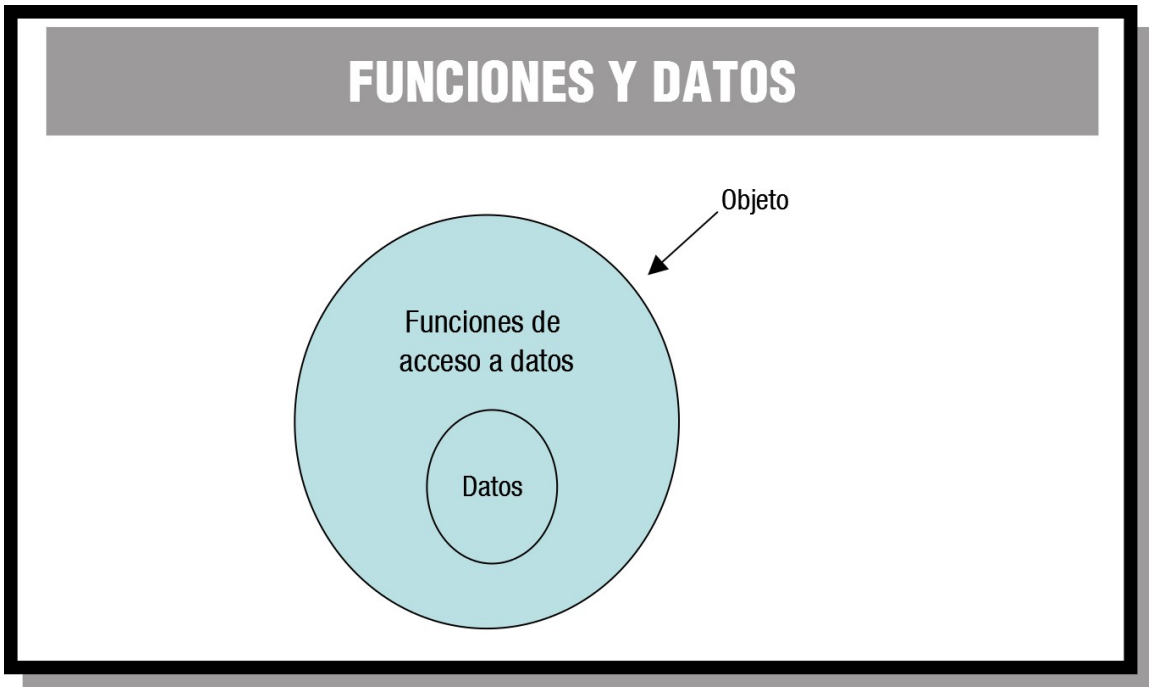
- **Programación Orientada a Objetos** (*Object Oriented Programming*): paradigma que intenta emular la representación y el funcionamiento de los objetos del mundo real.
- **Programación estructurada**: paradigma que se basa en el procesamiento de datos sin que exista una relación semántica entre ellos.





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**) que puede ser modificado a través de sus operaciones (**comportamiento**) .

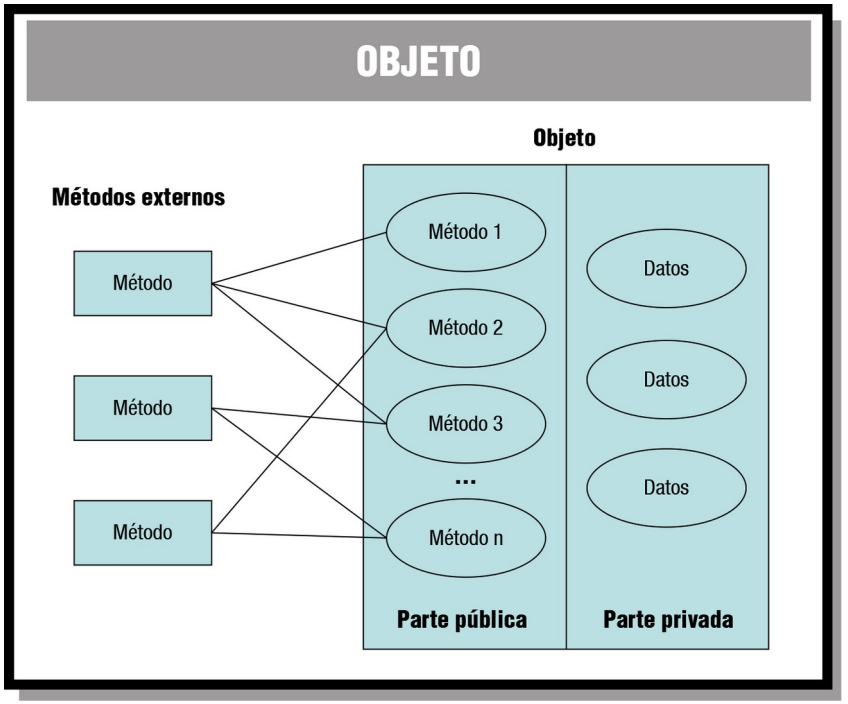


Introducción

- **Clase**: plantilla en la que se definen las características que tendrán los objetos de esta clase.
  - La información que contendrán → atributos.
  - La forma de interactuar con ellos o comportamiento del objeto → métodos.
- **Clase vs Objeto**.
  - La clase es una definición abstracta a partir de la cual se generan los objetos.
  - La clase establece los atributos mientras que un objeto tendrá valores concretos para cada uno de sus atributos.
  - Los objetos existen en memoria en tiempo de ejecución tras ser instanciados.

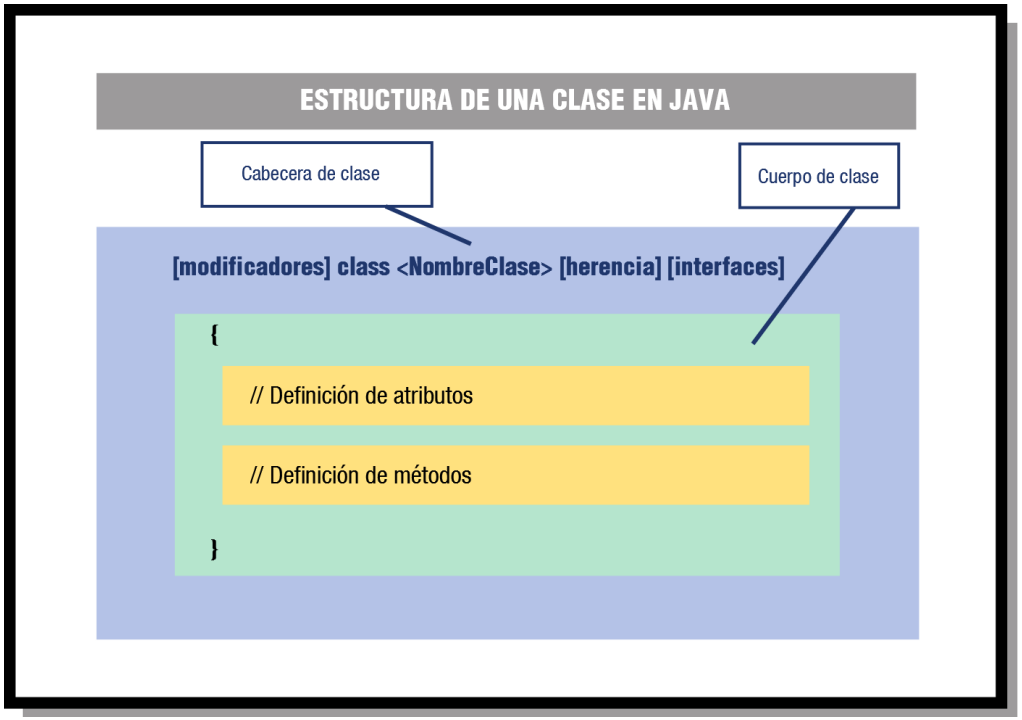
Clases y objetos

- 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.



Estructura y miembros de una clase

- 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. Posteriormente pueden aparecer tanto la herencia como las interfaces que se implementan.
  - **Cuerpo**: se definen los atributos y métodos que tendrá la clase.



Modificadores de una clase

- **Modificadores**: [public] [final | abstract]
  - ***public***. Indica que la clase es visible desde cualquier otra clase. Si no se especifica, la clase solo será visible desde las clases que estén en el mismo paquete (visibilidad de paquete).
  - ***final***. Indica que no se puede heredar de esta clase, es decir, finaliza la cadena de herencia.
  - ***abstract***. Indica que la clase es abstracta y no puede instanciarse (construir objetos con new). Se emplea en la herencia para crear jerarquías de clases donde las clases abstractas agrupan comportamiento común de las clases hijas.

Atributos

- Constituyen la estructura interna de los objetos o el conjunto de datos que almacenan.
- Se declaran como cualquier variable indicando el tipo y el identificador. El tipo de un atributo puede ser tanto un tipo primitivo como un tipo objeto.
- **Opcionalmente** puede contener algunos **modificadores**:

[public | protected | private] [static] [final] [transient] [volatile]

```
[modificadores] <tipo> <nombreAtributo>;
```

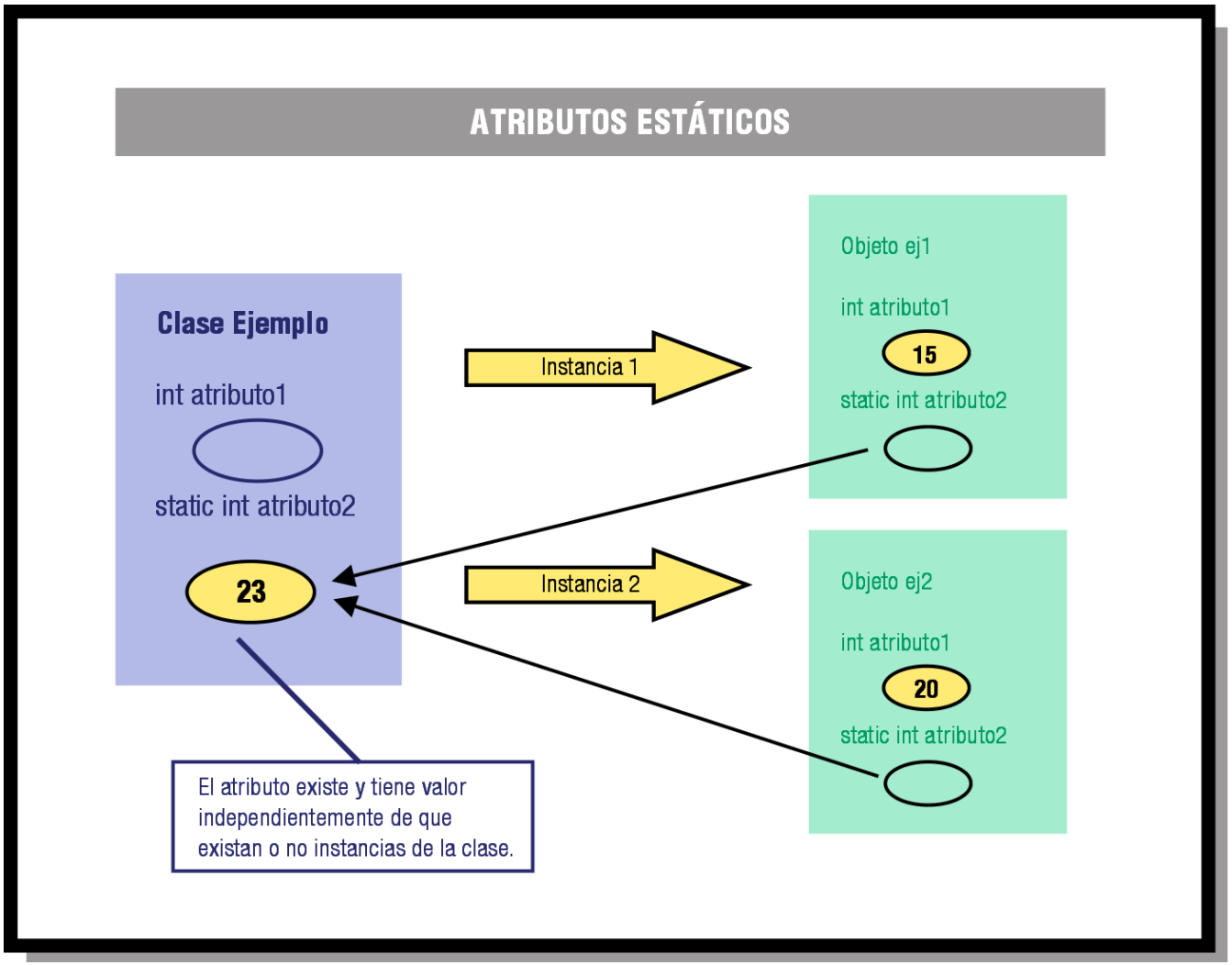
Atributos

- **Modificadores de acceso** (de menos a más restrictivo):
  - ***public***. Cualquier clase podrá acceder al atributo.
  - ***protected***. Tendrán acceso al atributo cualquier subclase y las clases que pertenezcan al mismo paquete.
  - Sin modificador (***package-private***). El acceso por defecto, en caso de no indicar ningún modificador el atributo será visible por las clases que pertenezcan al mismo paquete.
  - ***private***. Solo la propia clase puede acceder a este atributo.

Atributos

- **Modificadores de contenido:**
  - ***final***. Indica que el atributo tiene un valor constante y una vez inicializado no podrá ser modificado.
  - ***static***. Convierte el atributo en una variable de clase, común a todos los objetos, se denomina **atributo de clase**. Esta variable no se almacena en el espacio interno de cada objeto, sino que solo existe una copia común a todos los objetos de la clase.
  - ***transient***. Se utiliza en la serialización para indicar que el atributo no forma parte del estado del objeto y no se tendrá en cuenta.
  - ***volatile***. Se emplea en las aplicaciones multihilo para indicar que el valor del atributo se leerá y escribirá en memoria principal, por lo que no será cacheado.

Atributos

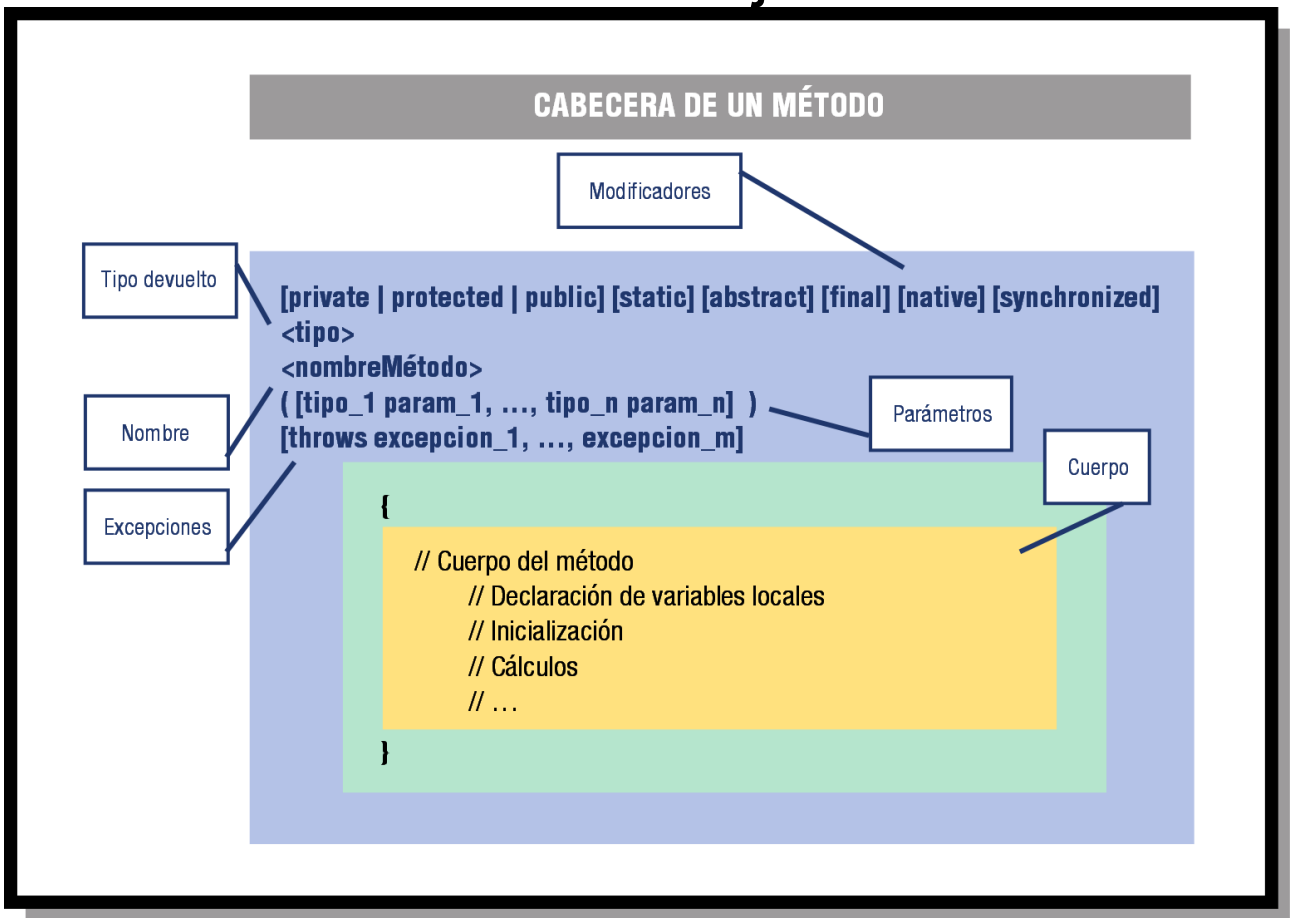

ATRIBUTOS ESTÁTICOS
Clase Ejemplo
int atributo1
static int atributo2
23
Instancia 1
Instancia 2
Objeto ej1
int atributo1
15
static int atributo2
Objeto ej2
int atributo1
20
static int atributo2
El atributo existe y tiene valor independientemente de que existan o no instancias de la clase.

Objetos inmutables

- Un objeto inmutable no puede ser modificado tras ser instanciado.
- Para hacer una clase inmutable:
  - Definir la clase como ***final*** para que no pueda ser modificada.
  - Definir todos los atributos ***private*** y ***final***.
  - Obligar a construir e inicializar el objeto completamente en la llamada al constructor.
  - No proveer ningún método que permita modificar el estado del objeto (métodos *setXXX* u otros).
  - Si la clase tuviera algún atributo mutable, este se debe copiar defensivamente.

Métodos

- Un **método** define el comportamiento de un objeto en sus interacciones con otros objetos.



Métodos

La cabecera de un método contiene los siguientes elementos.

- **[*modificadores*]**:
  - Modificadores de acceso: ***public***, ***protected*** o ***private***.
  - ***static***. Método de clase, por lo que para invocarlo no es necesario instanciar ningún objeto. Desde su interior tampoco se puede acceder a los atributos de objeto, pero sí a los atributos de clase (atributos static).
  - ***abstract***. El método no tiene implementación (no hay llaves) y solo tiene la cabecera que finaliza con un punto y coma. Solo se puede emplear si la clase también se ha declarado abstracta.
  - ***final***. El método no se puede sobreescribir por una clase descendiente.
  - ***native***. Permite que el método ejecute código compilado en otro lenguaje de programación.
  - ***synchronized***. Se emplea en programas multihilo para indicar que el método solo puede ser ejecutado por un hilo y el resto deberá esperar a que el hilo en ejecución termine.
- ***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***.
- ***throws***. Va seguido de la lista de excepciones que el método puede lanzar. Se estudiará en un tema posterior.

Métodos

- **Lista de parámetros variable**. Es posible construir un método que tenga un número de parámetros variable. A esta construcción se le denomina ***varargs***.
  - Se declara añadiendo **puntos suspensivos** (…) a continuación de la declaración del tipo de la variable.
  - Es posible mezclar *varargs* y parámetros fijos. Los parámetros variables deben aparecer al final.
  - La lista de parámetros variable se recorre como un array.

```
public double sumarElementos(double... elementos){
   double sumaTotal=0.0;
   for(double x:elementos){
      sumaTotal+=x;
   }
   return sumaTotal;
}
```

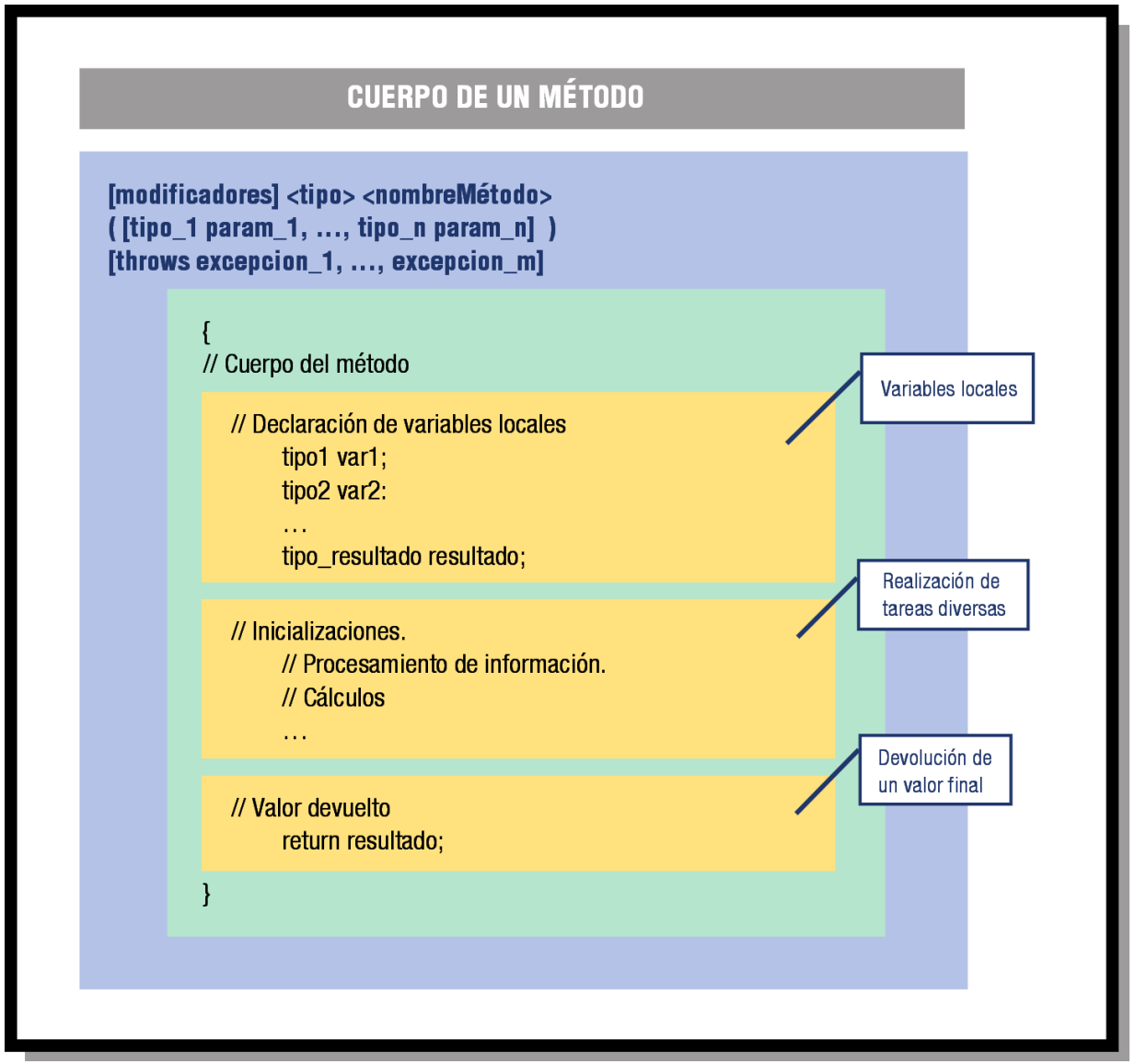
Métodos

- **Modificador *final* en los parámetros del método**. El valor del parámetro no se podrá modificar en el interior.
  - En caso de parámetros de tipo objeto, no se podrá modificar la referencia, pero sí el contenido del objeto.
  - En Java, **los parámetros siempre pasan por valor**, por lo que las modificaciones no tienen efecto fuera del método.
  - Dado que no es aconsejable modificar los valores de los parámetros de un método, señalarlos como *final* garantiza que el compilador mostrará un error siempre que esto ocurra.
  - Se aconseja su uso para evitar equivocaciones por parte del programador en caso de métodos muy largos.

```
public void metodo(final MiObjeto arg){
   arg = new OtroObjeto(); //Error de compilación

   arg.setXXX(20);         //Permitido
}
```

Métodos


CUERPO DE UN MÉTODO
[modificadores] <tipo> <nombreMétodo>
([tipo_1 param_1, ..., tipo_n param_n])
[throws excepcion_1, ..., excepcion_m]
{
// Cuerpo del método
// Declaración de variables locales
tipo1 var1;
tipo2 var2:
...
tipo_resultado resultado;
// Inicializaciones.
// Procesamiento de información.
// Cálculos
...
// Valor devuelto
return resultado;
}
Variables locales
Realización de tareas diversas
Devolución de un valor final

Métodos

- **Cuerpo del método**. Es todo lo que hay en el interior del método, encerrado entre llaves: { }
  - **Sentencia *return***: aparece si el método tiene tipo de retorno (si no es void). Puede haber más de una sentencia return pero se aconseja que solo haya una al final del método.
  - **Variables locales**: No se pueden declarar con el mismo nombre que los parámetros del método ya que se producirá un error de compilación. En cambio, sí pueden coincidir con el nombre de un atributo, que quedará oculto por la variable local. En este caso, para acceder al atributo habrá que utilizar la referencia ***this***.
  - **Operador de autoreferencia *this***: permite acceder a los atributos ocultados por variables locales o parámetros cuyos identificadores coinciden con el de los atributos. En cualquier caso, es aconsejable utilizar this siempre para acceder a miembros de la clase.

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, control de acceso y visibilidad

- 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, control de acceso y visibilidad

```
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.

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){...}
```

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**.
  - 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
```

Destrucción de objetos

- En Java, cuando un objeto deja de ser utilizado, se liberan todos sus recursos (memoria, manejadores de archivos, etc.).
- El proceso de destrucción de objetos lo lleva a cabo la máquina virtual de forma automática a través del **recolector de basura** (***Garbage Collector***). Se puede invocar con ***System.gc()*** pero es el sistema el que decide cuándo actuará.
- El método ***finalize()*** se puede sobreescribir para indicar al sistema las acciones que debe realizar antes de eliminar el objeto. Se emplea para labores de liberación de recursos complejas, como cerrar un fichero, la conexión a una base de datos, una conexión de red, etc.
- Se puede iniciar el proceso de destrucción de objetos invocando el método ***System.runFinalization()***. El objeto podría no ser destruido si el GC aún no lo ha marcado para su eliminación.

Métodos factoría

- En determinadas circunstancias, se emplean **métodos factoría** para controlar la creación de objetos, invisibilizando el acceso a los constructores con el modificador *private*.
- Los métodos factoría se declaran *static* y en su interior invocan a algún constructor de al clase mediante new.
- Ejemplo de métodos factoría: ***LocalTime.of***, ***LocalTime.parse***, ***LocalTime.now***.
- Los **métodos factoría** son un patrón de diseño conocido para encapsular la creación de objetos. Ofrece algunas ventajas como poder elegir qué tipo de subclase instanciar o la reutilización de objetos en casos donde la construcción de los mismos es muy costosa.

Atributos *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>;*

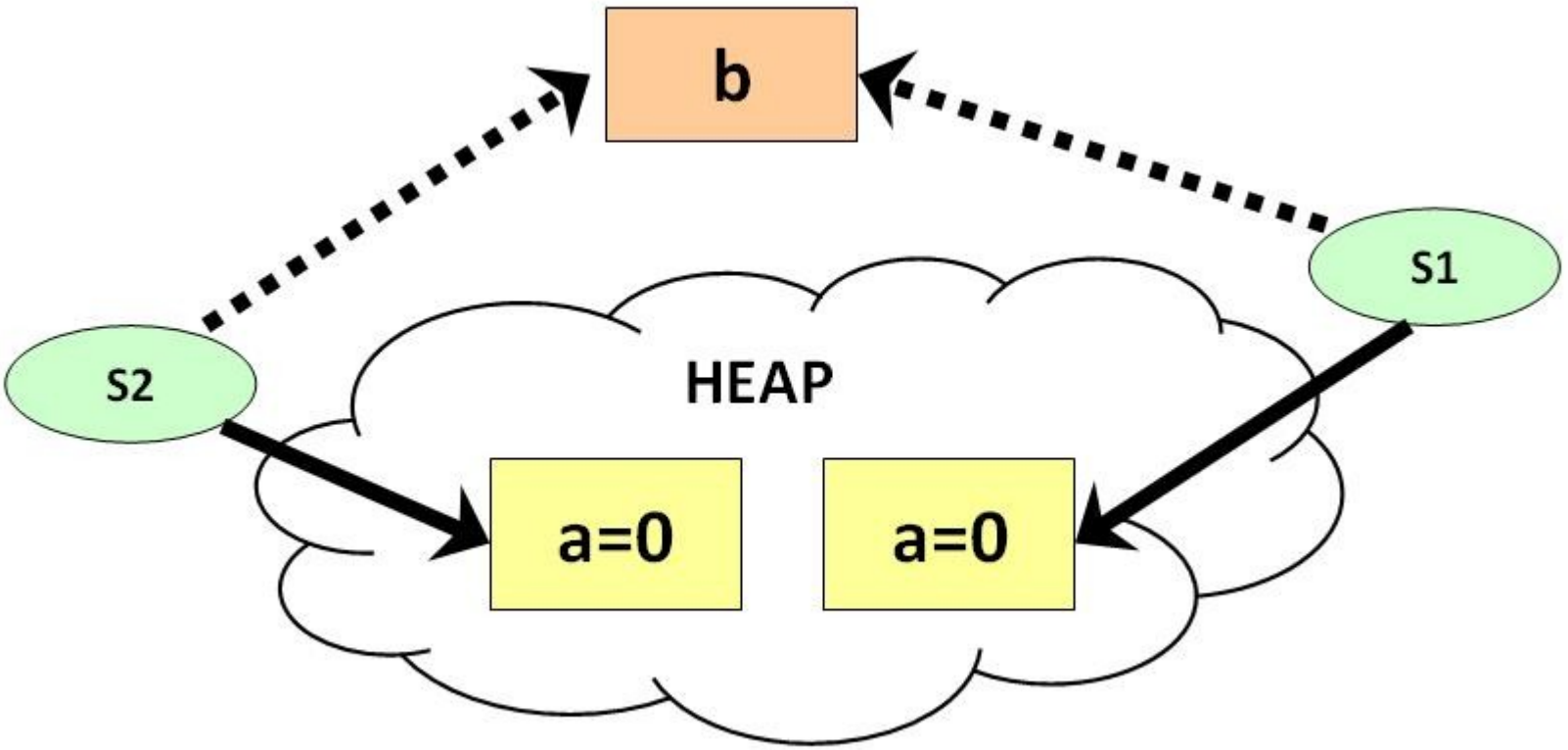
Métodos *static*

- 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);
     }
}
```

Bloques de inicialización

- **Bloque de inicialización *static*.**
  - Se ejecuta al cargar la clase en memoria una única vez.
  - Puede haber más de uno y se pueden colocar en cualquier lugar del cuerpo de la clase
  - Los bloques se ejecutan en el orden en el que aparecen.
  - Una alternativa es utilizar métodos estáticos privados.

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

https://docs.oracle.com/javase/tutorial/java/javaOO/initial.html

Bloques de inicialización

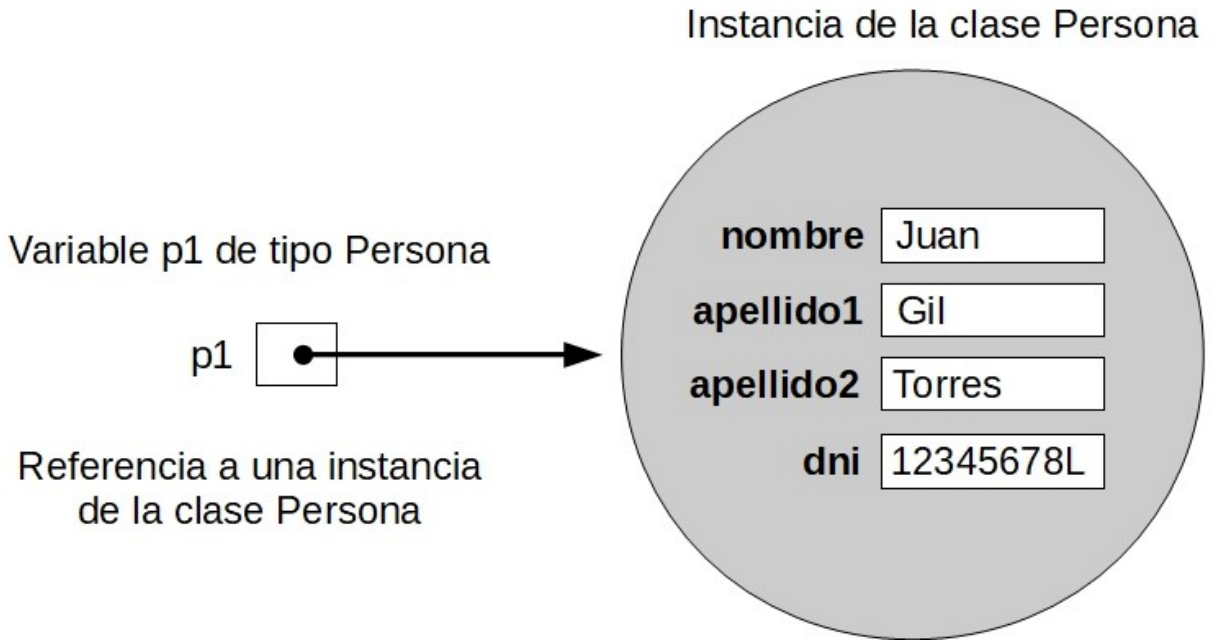
- **Bloque de inicialización de instancia**.
  - Lo habitual es utilizar constructores para iniciar los atributos de instancia.
  - Se ejecuta cada vez que se instancia un nuevo objeto.
  - Este código se copia en todos los constructores de la clase por lo que puede ser útil para compartir código idéntico.
  - Una alternativa es usar métodos finales.

```
class Test{
   {
      //Código del bloque de inicialización de instancia
   }
}
```

https://docs.oracle.com/javase/tutorial/java/javaOO/initial.html

Referencias a un objeto

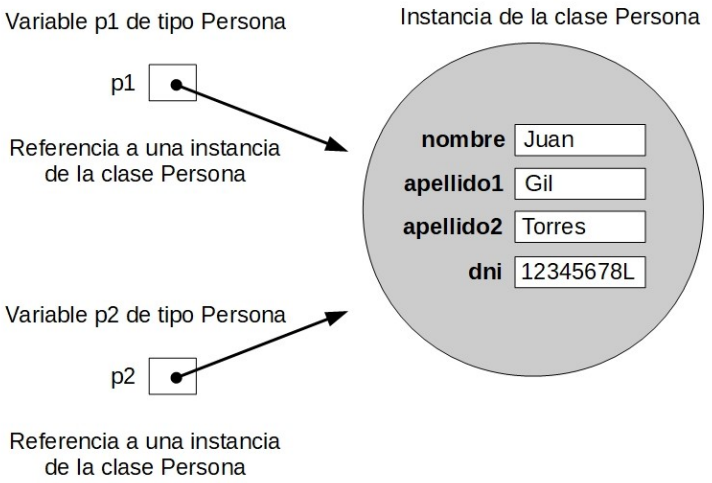
- **Las variables de tipo objeto almacenan referencias al objeto en memoria**.
- Modificar estas variables no modifican el contenido del objeto, sino el lugar a donde apuntan.




```
Persona p1 = new Persona("Juan", "Gil", "Torres", "12345678L");
```

Referencias a un objeto

- **Las variables de tipo objeto almacenan referencias al objeto en memoria**.
- Modificar estas variables no modifican el contenido del objeto, sino el lugar a donde apuntan.




```
Persona p2 = p1;
```

Unidad 5. Desarrollo de clases

DUDAS Y PREGUNTAS