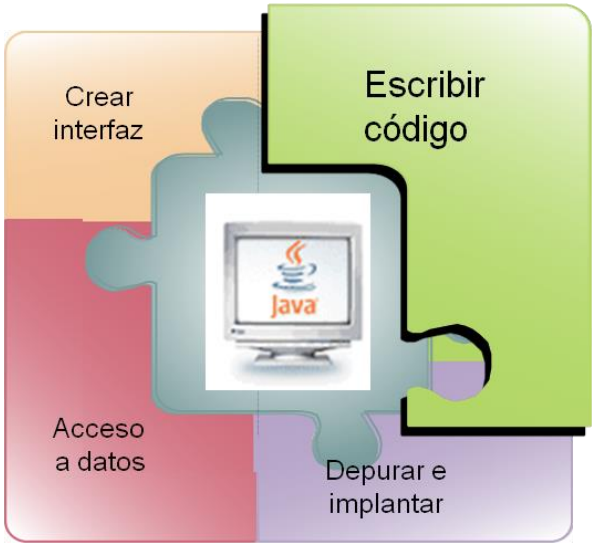
Unidad 6

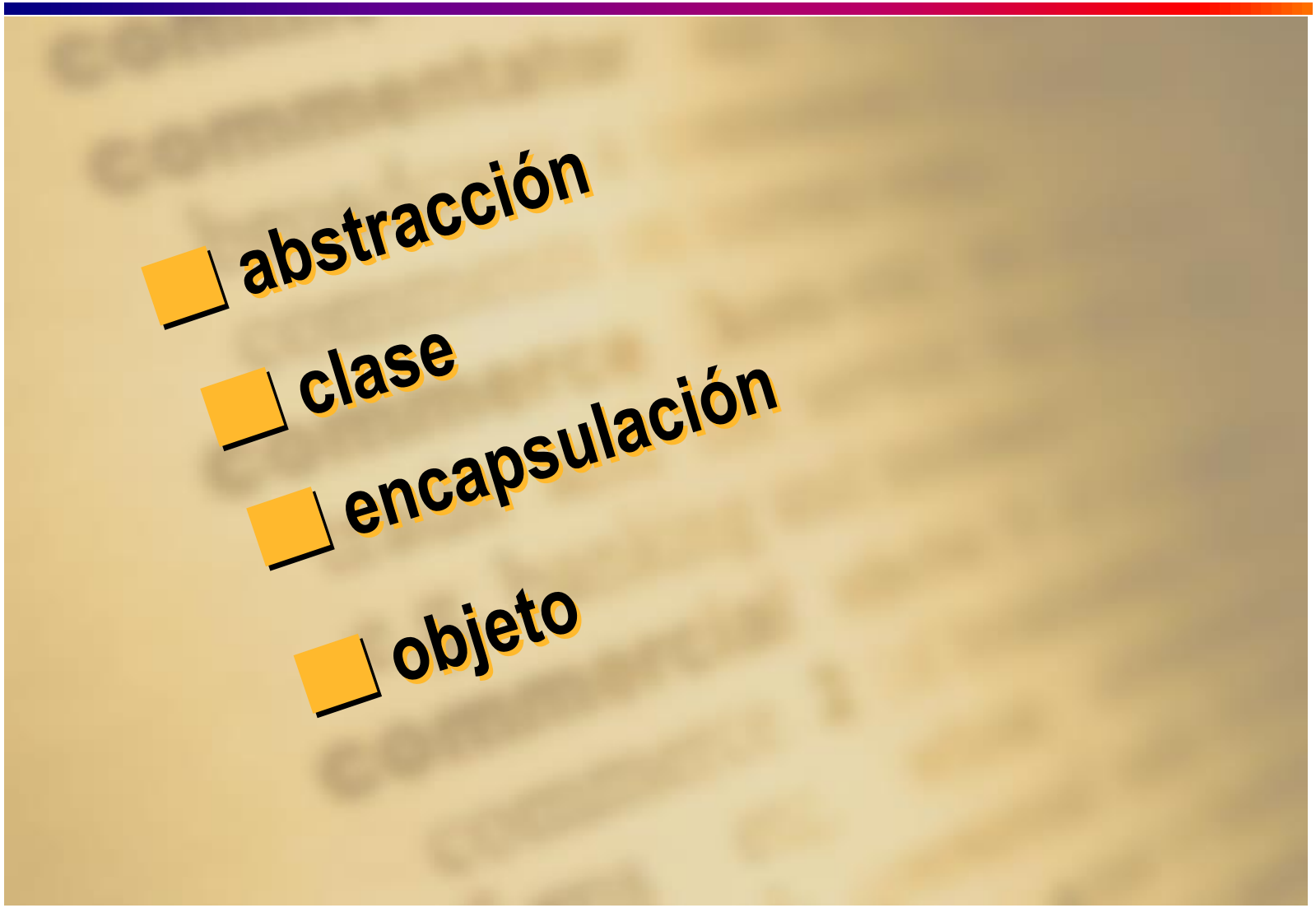
Programación Orientada a Objetos (I)

Introducción



- Entender el concepto de POO
- Entender el concepto de Clase y Objeto
- Trabajar con clases
- Uso de miembros genéricos (static)

Lección: Entender el concepto de POO



Programación Procedural <-> POO

- **Programación Estructurada Procedural**
  - Datos
  - Subprogramas (procedimientos y funciones)
- **Programación Orientada a Objetos (Object Oriented Programming)**
  - Objetos

    Agrupación de datos (atributos o campos) y subprogramas (métodos) con funcionalidad similar bajo un sistema unificado de manipulación y acceso

Fundamentos de la POO

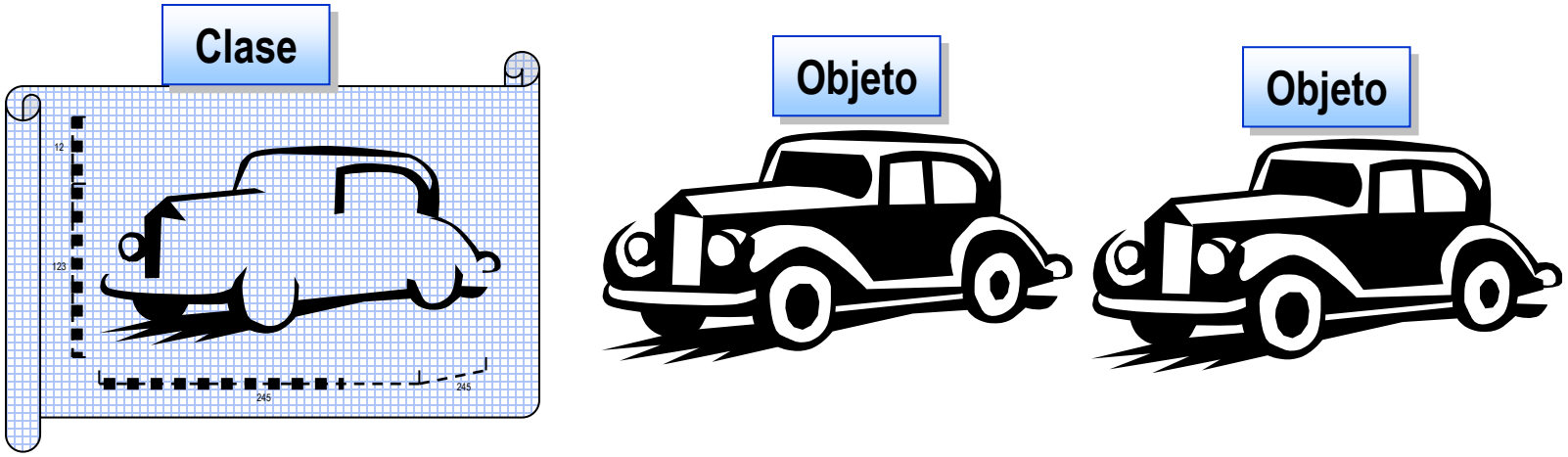
- Una objeto contiene un conjunto de datos (atributos o campos) y procedimientos (métodos) que ejecutan una serie de procesos destinados a resolver un grupo de tareas con un denominador común
- Una aplicación orientada a objetos tendrá tantos objetos como aspectos del programa sea necesario resolver
- Un procedimiento que esté situado dentro de un objeto no podrá llamar ni ser llamado por otro procedimiento situado en un objeto distinto si no es bajo una serie de reglas
- Igualmente los datos que contenga el objeto permanecerán aislados del exterior y sólo serán accesibles siguiendo ciertas normas

Lección: Entender el concepto de Clase

- **Qué es una clase**
- **Qué es un objeto**
- **Algunas propiedades:**
  - Abstracción
  - Encapsulamiento

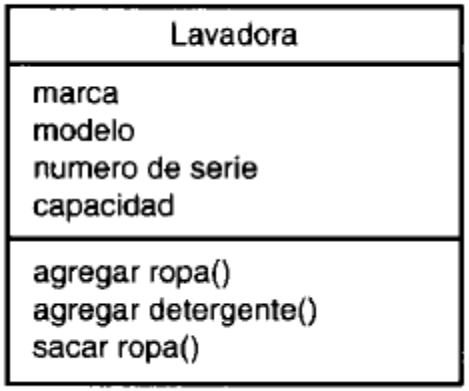
¿Qué es una clase y qué es un objeto?

- **Una *clase* es una plantilla o estructura preliminar que describe un objeto:**
  - define los atributos: propiedades, datos
  - Define los métodos: acciones, operaciones
- **Un *objeto* es una instancia de una clase**



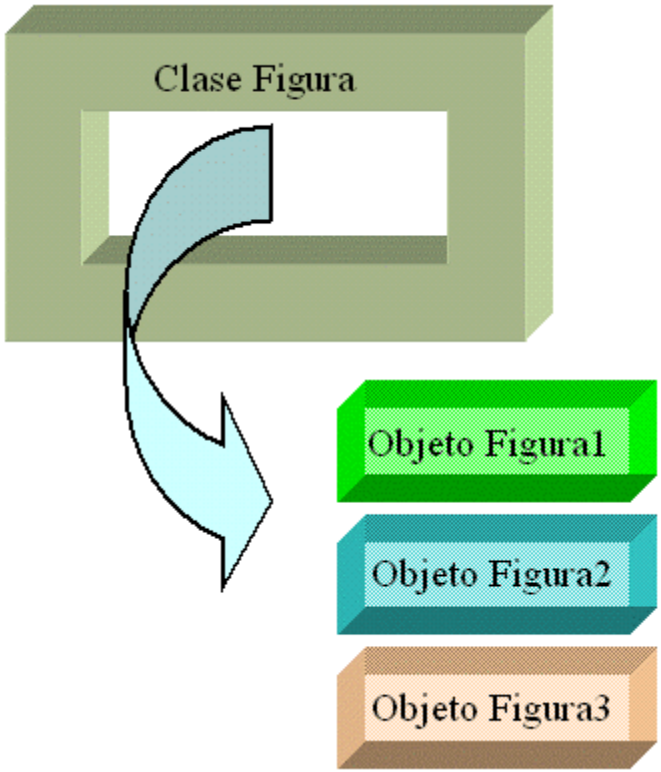
¿Qué es una clase y qué es un objeto?

- **Definimos la clase Lavadora:**



- **Instanciamos (o creamos) los objetos:**
  - miLavadora
  - tuLavadora
  - futuraLavadora

¿Qué es una clase y qué es un objeto?



Analogía

Tipo Primitivo ⟺ Clase

Variable ⟺ Objeto

Algunas propiedades de las Clases

- **Abstracción:**
  - Poner a disposición únicamente las propiedades y acciones que sean necesarias

- **Diferentes tipos de problemas requieren distintas abstracciones**

Ejemplo de abstracción

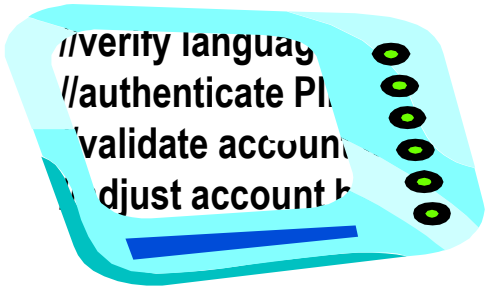
- Dos objetos coche, uno deportivo y otro familiar: Su aspecto exterior es muy diferente, sin embargo sabemos que ambos pertenecen a la clase Coche porque realizamos una abstracción o identificación mental de los elementos comunes que ambos tienen (ruedas, volante, motor, puertas, etc.).
- En el desarrollo de un programa de gestión orientado a objetos realizamos una abstracción de los objetos que necesitaríamos para resolver los procesos:
  - objeto Empleado, para gestionar al personal
  - objeto Factura, para gestionar las ventas
  - objeto Usuario, para verificar las personas que utilizan la aplicación, etc…

Algunas propiedades de las Clases

- **Encapsulación:**
  - separar el aspecto externo del objeto (accesible por otros objetos) del aspecto interno del mismo (que será inaccesible para los demás)
  - Establece la separación entre el interfaz del objeto y su implementación. La estructura interna permanece privada

Lo que ve el usuario:

Lo que está encapsulado:



Ejemplo de encapsulación

- **Un terminal de autoservicio**
  - La interfaz de la terminal es simple para el cliente y el funcionamiento interno está oculto
- **Una clase Cuenta**
  - Encapsula los métodos, campos y propiedades que describen una cuenta bancaria
  - La encapsulación permite a los usuarios utilizar los datos y procedimientos de la clase como una unidad sin conocer el código concreto

Lección: Trabajar con clases

- Cómo crear una nueva clase
- Cómo agregar atributos
- Cómo agregar métodos
- Cómo crear una instancia de una clase
- La referencia this
- Cómo utilizar los constructores
- Cómo utilizar los destructores

Cómo crear una nueva clase

- **Sintaxis:**

```
[ModificadorAccesso] class NombreClasse {
//Atributos de la clase
//Métodos de l clase
}
```

- **Una clase public debe ser declarada en un fichero fuente con el nombre de esa clase: NombreClase.java.**
  - En un fichero fuente puede haber más de una clase, pero sólo una con el modificador public

Modificadores de acceso de clases

- Java define 4 modificadores fundamentales que califican a clases:

Palabra clave	Definición
public	La clase es accesible desde otros paquetes
(por defecto) de paquete	La clase será visible en todas las clases declaradas en el mismo paquete
abstract	La clase no pueden ser instanciadas. Sirve únicamente para declarar subclases. Ya lo veremos…
final	ninguna clase puede heredar de una clase final. Ya lo veremos…

Cómo agregar atributos

- Sintaxis general:

[modificadorDeÁmbito] [static] [final] [transient] [volatile]
tipo nombreAtributo

- De momento, versión reducida:

[modificadorDeÁmbito] [static] [final] tipo nombreAtributo

Cómo agregar atributos y métodos

- Java define 4 modificadores fundamentales que califican a métodos y atributos:

Palabra clave	Definición
public	el elemento es accesible desde cualquier sitio
protected	el elemento es accesible dentro del paquete en el que se define y, además, en las subclases
(por defecto) de paquete	el elemento sólo es accesible dentro del paquete en el que se define (clases en el mismo directorio)
private	el elemento sólo es accesible dentro de la clase en el que se define

Cómo agregar atributos y métodos

- **Modificadores de acceso de atributos y métodos**

	visibilidad			
modificador	fichero	paquete (directorio)	subclases (extends)	cuaquiera
private	SÍ	NO	NO	NO
de paquete (package local)	SÍ	SÍ	NO	NO
protected	SÍ	SÍ	SÍ	NO
public	SÍ	SÍ	SÍ	SÍ

Cómo agregar atributos

- Los campos o atributos se recomienda que sean siempre "private"

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

```
public class Triangulo {
   private int lado1,lado2,lado3;
   …
}
```

Cómo agregar atributos

- **Si un atributo o variable de clase no ha sido inicializada tiene un valor asignado por defecto:**
  - Para las variables de tipo numérico, el valor por defecto es cero (0)
  - Las variables de tipo char, el valor ‘\u0000’
  - Las variables de tipo boolean, el valor false
  - Para las variables de tipo referencial (objetos), el valor null

Cómo agregar métodos

- Sintaxis general:

```
[ModificadorDeÁmbito] [ static] [abstract] [ final] [ native] [synchronized]
    TipoDevuelto NombreMétodo ( [ ListaParámetros] )
[throws   ListaExcepciones]
```

- De momento, versión reducida:

```
[ModificadorDeÁmbito] [ static]
   TipoDevuelto NombreMétodo ( [ ListaParámetros] )
```

- **Los métodos serán generalmente "public"**

Cómo agregar métodos

- **Agregar un método denominado** esEquilatero

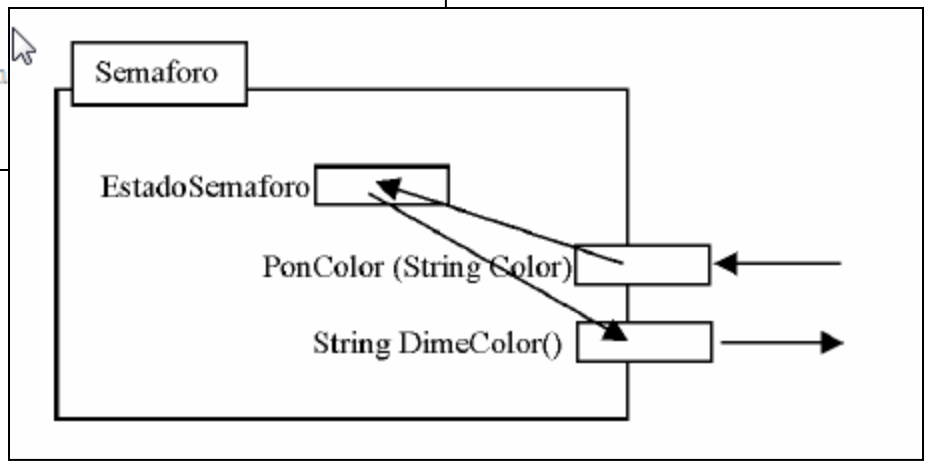
```
public class Triangulo {
   private int lado1,lado2,lado3;
   …
   public void esEquilatero() {
    if (lado1==lado2 && lado1==lado3) {
      System.out.println("Es equilátero");
    } else {
      System.out.println("No es equilátero");
    }
  }
…
}
```

Cómo agregar métodos

- **Los métodos pueden estar sobrecargados**
  - Dos o más métodos con el mismo nombre pero con una lista de parámetros distinta:
    - O bien, distinto número de parámetros
    - O bien, al menos un parámetro de tipo diferente
  - Estas sobrecargas hacen que el método sea más flexible para los usuarios del método

Ejemplo. Clase Semaforo (apuntes A1)

```
/*jbobi Cap. 6 modificado*/
package Semaforo;

public class Semaforo {
   private String EstadoSemaforo =  "Rojo";

  public void PonColor (String Color) {
     EstadoSemaforo = Color;
  }

  public String DimeColor() {
     return EstadoSemaforo;
  }

} // Fin de la clase Sem
```



Cómo crear una instancia de una clase I

- Cuando definimos una clase estamos creando una plantilla y definiendo un tipo. Hecho esto, podemos **crear objetos** de esa clase (o también decimos **instanciar** la clase)
- Para declarar una variable que pueda referenciar a un objeto de una clase:

  Clase variable;

  - Con ello tenemos un apuntador capaz de direccionar un objeto (pero no tenemos el objeto!!)
  - De momento la variable no apunta a ningún objeto… Se dice que contiene la referencia **null**
- Para crear el objeto, empleamos la palabra reservada new seguida de método que se llama igual que la clase (el constructor)

  variable= new Clase( );

  - Así conseguimos una variable que apunta (o que referencia) al objeto creado

Cómo crear una instancia de una clase II

- Usamos la palabra reservada new para crear una instancia de la clase
  - Podemos definir el objeto y luego crearlo:

    Persona cliente1, cliente2;

    cliente1 = new Persona( );
  - También **podemos definir y crear el objeto al mismo tiempo**:

    Perona cliente1 = new Persona( );

Cómo acceder a un miembro del objeto

- **Para acceder a un atributo del objeto:**
  - objeto.atributo

    cliente1.edad=23;
- **Para acceder a un método del objeto:**
  - objeto.metodo ( )

    cliente1.esMayorDeEdad( );

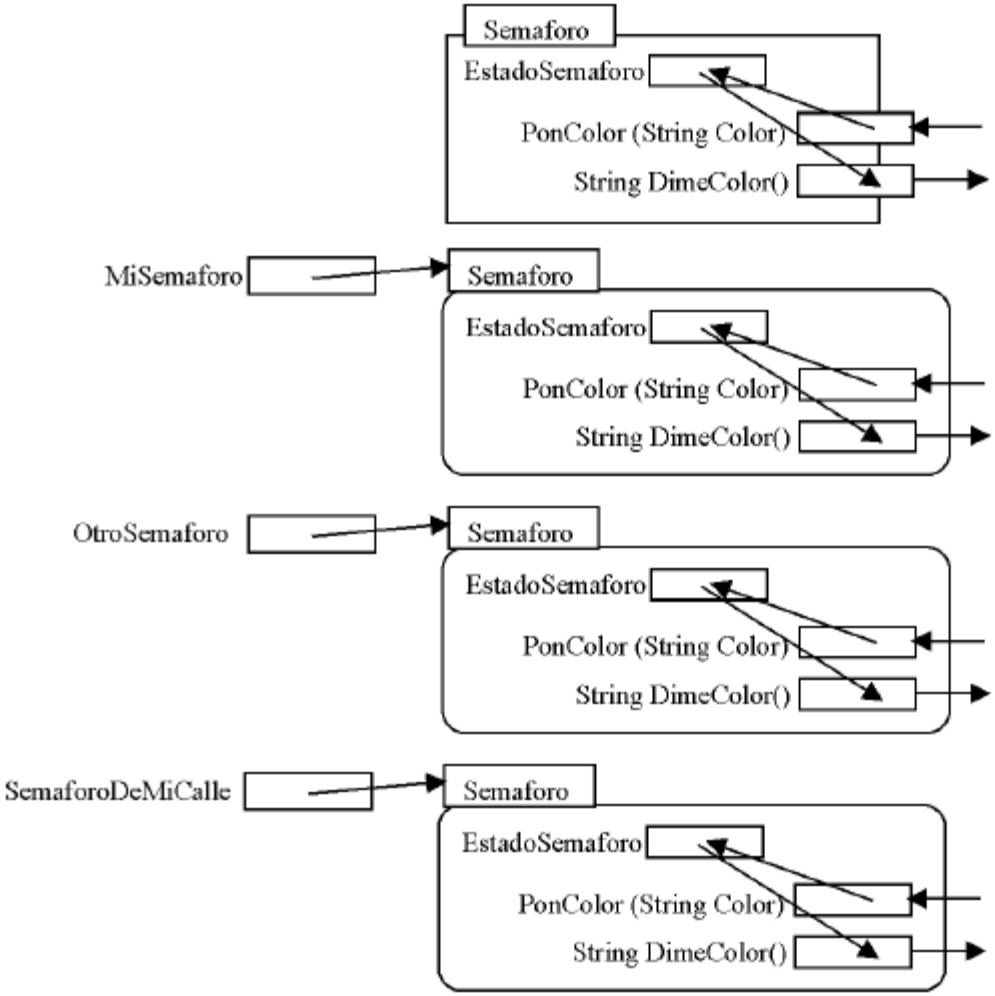
- //Ojo! si el atributo es private nos dará error el acceso desde otra clase

Ejemplo. Objetos de la clase Semaforo

```java
/*jbobi Cap. 6 modificado*/
package Semaforo;

public class Semaforo {
    private String EstadoSemaforo =  "Rojo";

  public void PonColor (String Color) {
     EstadoSemaforo = Color;
  }

  public String DimeColor() {
     return EstadoSemaforo;
  }

} // Fin de la clase Semaforo
```

```java
public class PruebaSemaforo {
   public static void main (String[] args) {
        Semaforo MiSemaforo = new Semaforo();
        Semaforo SemaforoDeMiCalle = new Semaforo();
        Semaforo OtroSemaforo = new Semaforo();

        MiSemaforo.PonColor("Rojo");
        OtroSemaforo.PonColor("Verde");

        System.out.println( OtroSemaforo.DimeColor() );
        System.out.println( SemaforoDeMiCalle.DimeColor() );

        if (MiSemaforo.DimeColor().equals("Rojo"))
         System.out.println ("No Pasar");

  }
}
```

Ejemplo. Objetos de la clase Semaforo



Métodos Setters (modificadores) y Getters (consultores)

- **Es buena práctica:**
  - Crear los atributos con el modificador private
  - Crear métodos públicos para acceder a los atributos:
    - consultar (get)
    - modificar (set)
  - Desde otras clases externas no podrán modificar ni acceder a los atributos de la clase, obligatoriamente deberán pasar por el Set y Get
- **Beneficios del encapsulamiento:**
  - Que nadie asigne valores por equivocación o sobrescriba funcionalidades cuando no debe
  - Programación en "**Caja Negra**", un programador que use esa clase no necesitará conocer **como lo hace**, sino solamente **que hace**

Ejemplo

```java
public class Punto {

    private int x, y;

    public void setCoordenadas(int a, int b) {
        x = a; // o bien this.x=a;
        y = b;  // o bien this.y=b;
    }
    public void setCoordenadaX(int a) {
        x = a; // o bien this.x=a;
    }
    public void setCoordenadaY(int a) {
        y = a;  // o bien this.y=a;
    }
    public int getCoordenadaX() {
        return x;
    }
    public int getCoordenadaY() {
        return y;
    }
```

```java
package punto2;
import java.util.Scanner;
public class PuntoApp {
    public static void main(String[] args) {
        int coorX, coorY;
        Punto punto1;
        punto1=new Punto();
        Scanner  teclado=new Scanner(System.in);
        System.out.print("Ingrese coordenada x :");
        coorX=teclado.nextInt();
        // punto1.x=coordX; dona error per ser private!!!!!!
        System.out.print("Ingrese coordenada y :");
        coorY=teclado.nextInt();
        punto1.setCoordenadas(coorX,coorY);

        System.out.println("Hablamos del punto ( "
            +punto1.getCoordenadaX()+" , "+punto1.getCoordenadaY()+" )");
        punto1.imprimirCuadrante();

    }
}
```

La referencia this

- **La palabra *this* es una referencia al propio objeto en el que estamos**
- **Ejemplo:**

```
//   ejemplo A

public class Punto {
  private int x, y;

  public void setCoordenadas (int a, int b){
     this.x=a;
     this.y=b;
  }
  …
}
```

```
//    ejemplo B

public class Punto {
  private int x, y;

  public void setCoordenadas (int x, int y){
     this.x=x;
     this.y=y;
  }
  …
}
```

- En el ejemplo A el uso de this es opcional
- En el ejemplo B hace falta la referencia this para clarificar cuando se usan los atributos y cuando los argumentos con el mismo nombre

Cómo utilizar los constructores

- **Un constructor es un método especial de una clase que es llamado automáticamente siempre que se crea un objeto de esa clase**
  - Es decir al usar la instrucción new
- **Su función es iniciar el objeto**
  - Se recomienda que los constructores inicialicen todos los campos del objeto

```
public class Rectangulo {
…
public Rectangulo(int x1, int y1, int w, int h) {
         x=x1;
         y=y1;
         ancho=w;
         alto=h;
    }
…
}
```

Cómo utilizar los constructores

- **Para declarar un constructor basta con declarar un método con el mismo nombre que la clase**
  - No se declara tipo devuelto por el constructor
    (ni siquiera void)
- **Si no se escribe ningún constructor java se inventa uno que no tiene ningún argumento e inicializa todos los campos a "cero“**
  - Java sólo inventa constructores si el programador no escribe ninguno
  - En cuanto se escribe uno, java se limita a lo escrito

Cómo utilizar los constructores

- **Es posible declarar distintos constructores (sobrecarga de métodos) al igual que los demás métodos de una clase**

```
public class Rectangulo {
    private int x;
    private int y;
    private int ancho;
    private int alto;

    public Rectangulo() {
        x=0;
        y=0;
        ancho=0;
        alto=0;
    }
    public Rectangulo(int x1, int y1, int w, int h) {
        x=x1;
        y=y1;
        ancho=w;
        alto=h;
    }
    public Rectangulo(int w, int h) {
        x=0;
        y=0;
        ancho=w;
        alto=h;
    }…
…}
```

Cómo utilizar los constructores

- **Un constructor puede llamar a otro constructor**

```
class Rectangulo {
    private double ancho, alto;

    public Rectangulo (double al, double an) {
        alto= al;
        ancho= an;
    }
    // construye un cuadrado
    public Rectangulo(double lado) {
        this(lado, lado);
    }
    // construye un cuadrado de lado 1
    public Rectangulo() {
        this(1);
    }
```

Ejemplo de Constructor

```
class Ficha {
  private int casilla;
   public Ficha() { //constructor
        casilla = 1;
  }
  public void avanzar(int n) {
        casilla += n;
  }
  public int casillaActual(){
        return casilla;
  }
}

public class app {
  public static void main(String[] args) {
        Ficha ficha1 = new Ficha();
        ficha1.avanzar(3);
        System.out.println(ficha1.casillaActual());//Da 4
```

¿Destructores?

- En Java hay un recolector de basura (*garbage collector) que se encarga de* gestionar los objetos que se dejan de usar y de eliminarles de memoria
- Este proceso es automático e impredecible y trabaja en un hilo (*thread) de baja prioridad*
- Por lo general ese proceso de recolección de basura, trabaja cuando detecta que un objeto hace demasiado tiempo que no se utiliza en un programa
- Esta eliminación depende de la máquina virtual. En casi todas, la recolección se realiza periódicamente

Lección: Uso de miembros estáticos (static)

- Cómo utilizar atributos static
- Cómo utilizar métodos static

Atributos static

Existen dos tipos de atributos:

- **Atributos de objeto (o de instancia)**
  - Los atributos de objetos son variables u objetos que almacenan valores distintos para instancias distintas de la clase (para objetos distintos)
  - Si no se especifica los atributos son de objeto
- **Atributos de clase o Atributos estáticos (o genéricos)**
  - Los atributos de clase son variables u objetos que almacenan un solo valor para todos los objetos instanciados a partir de esa clase
  - Se declaran con la palabra reservada **static**

Métodos static

- **Los métodos de objeto (o de instancia)**
  - Implementan la funcionalidad asociada al objeto
  - Solo pueden ser usados a través de una instancia de la clase
  - Ejemplos:
    - triangulo1.esEquilatero() punto2.setCoordenadas(3,5)
    - nombre.equals("Pepe") lector.nextInt()
- **Los métodos estáticos (o de clase)**
  - Únicamente pueden acceder a los atributos estáticos de la clase. Nunca a los atributos de objeto
  - No se necesita crear un objeto de esa clase (instanciar la clase) para poder llamar a ese método
  - Ejemplos:
    - Math.sqrt(4) ClaseU04.factorial(7)
    - public static void main(){…}

Atributos y métodos static

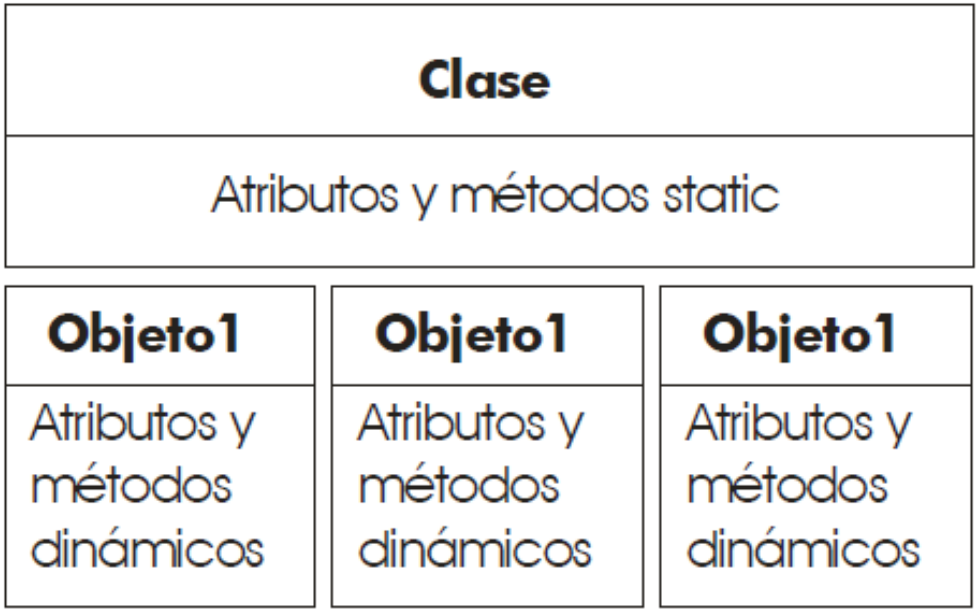


Diagrama de funcionamiento de los métodos y atributos static

Atributos y métodos static

- **Así pues, Crear objetos…**
- **Es necesario crear objetos para poder utilizar los métodos y atributos de objeto de una clase**
  - Los utilizamos:
    - objeto.método()
    - objeto.atributo
- **No es necesario crear objetos para poder utilizar los métodos y atributos estáticos**
  - Los utilizamos:
    - Clase.método()
    - Clase.atributo

Atributos y métodos static

- **Hay que crear métodos y atributos estáticos cuando ese método o propiedad vale o da el mismo resultado para todos los objetos**
- **Hay que utilizar métodos de objeto cuando ese método da resultados distintos según el objeto**
- **Por ejemplo (aunque en última instancia depende del problema a resolver) , en una clase que represente aviones:**
  - la capacidad o plazas reservadas serían atributos de objeto
  - el número total de aviones sería un atributo estático
  - mostarTotalAviones() sería un método static
  - reservarPlaza() sería un método de instancia