

# Introducción a la Programación

**BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA  
FACULTAD DE CIENCIAS DE LA COMPUTACIÓN**

**Luz A. Sánchez Gálvez**

**[sanchez.galvez@correo.buap.mx](mailto:sanchez.galvez@correo.buap.mx)**

**[lgalvez@cs.buap.mx](mailto:lgalvez@cs.buap.mx)**

**[luzsg@hotmail.com](mailto:luzsg@hotmail.com)**

# Modularización

---

- ▶ Un programa puede ser muy grande y complejo. Para gestionar esta complejidad, es necesario realizar programas de forma modular.
- ▶ El programa debe estar estructurado en partes autónomas, llamadas módulos.
- ▶ Los diversos módulos estarán relacionados entre sí a través de relaciones precisas.



# Modularización

---

- ▶ Al proceder de este modo, se mejoran diversas cualidades de un programa.
- ▶ Específicamente:
  - ▶ **legibilidad** (o comprensibilidad) del programa
  - ▶ **extensibilidad** (la posibilidad de extender las funcionalidades del programa, si es necesario)
  - ▶ **reusabilidad** de partes del programa para diferentes propósitos.



# Métodos vistos como módulos

---

- ▶ Un método, visto como un módulo, se caracteriza por:
    - ▶ Servicios exportados: "qué" realiza el método.
    - ▶ Interfaz: la cabecera del método (especifica los tipos de parámetros de entrada y los resultados).
    - ▶ Los servicios importados: otros métodos o clases utilizadas para realizar el método.
    - ▶ Estructura interna: el código de Java describe "cómo" el método implementa lo "qué" realiza (esto no es de interés para los clientes del método, es decir, para quienes llaman al método).
- 



# Definición de métodos estáticos

---

- ▶ En Java, los métodos más simples son los métodos estáticos.
- ▶ Los métodos estáticos son métodos (definidos en una clase) que no tienen un objeto de invocación.
- ▶ Definición de un método estático



# Definición de métodos estáticos

- ▶ **public static tipodevuelto nombremétodo (parámetros)**
  - ▶ **public** indica que el método definido es accesible desde fuera de la clase.
  - ▶ **static** indica que el método es estático (es decir, no tiene objeto de invocación).
  - ▶ **tipodevuelto** es el tipo del resultado devuelto por el método o **void**, si el método no devuelve ningún resultado.
  - ▶ **nombremétodo** es el nombre del método.
  - ▶ **parámetros** es una lista de declaraciones de parámetros (tipo y nombre) separados por comas, cada parámetro es una variable, la lista de parámetros también puede estar vacía.
  - ▶ El **cuerpo del método** (bloque) contiene las instrucciones que se ejecutarán cuando el método es invocado.

# Definición de métodos estáticos

---

- ▶ El método `main()`, es un método estático.
  - ▶ `public static void main (String[] args) {`  
`...`  
`}`
  - ▶ es un método público, accesible desde fuera de la clase donde se define,
  - ▶ es un método estático,
  - ▶ no devuelve ningún resultado (tipo `void`),
  - ▶ y tiene un parámetro de tipo ***array*** de cadenas.



# Instrucción **return**

---

- ▶ Si un método debe devolver un resultado, entonces debe contener una instrucción **return**.
- ▶ Cuando se ejecuta **return** dentro de un método, causa la terminación del mismo y devuelve el resultado del método.
- ▶ **Sintaxis:**
  - ▶ **return expresión;**
    - ▶ expresión cuyo valor debe ser compatible con el tipo del resultado declarado en la cabecera del método.






# Métodos estáticos en la misma clase

```
import javax.swing.JOptionPane;
```

```
public class ImprimeSaludo {  
  
    public static void imprimeSaludo() {  
        System.out.println("¡Buenos días!");  
    }  
    public static void imprimeSaludoPersonal(String nombre, String apellido) {  
        System.out.print("¡Buenos días!");  
        System.out.print(nombre);  
        System.out.print(" ");  
        System.out.print(apellido);  
    }  
    public static void imprimeSaludoInformal(String nombre) {  
        System.out.println(" Hola" + nombre );  
    }  
    public static String saludoPersonal(String nombre, String apellido) {  
        return "¡Buenos días! " + nombre + " " + apellido;  
    }  
    public static void main(String[] args) {  
        imprimeSaludo();  
        String n = JOptionPane.showInputDialog("Nombre");  
        String a = JOptionPane.showInputDialog("Apellido");  
        imprimeSaludoPersonal(n,a);  
        imprimeSaludoInformal(n);  
        JOptionPane.showMessageDialog(null,saludoPersonal(n,a));  
        System.exit(0);  
    }  
}
```



# Métodos estáticos en otra clase (1)

```
public class Saludos {  
    public static void imprimeSaludo() {  
        System.out.println("Buenos días");  
    }  
    public static void imprimeSaludoPersonal(String nombre, String apellido) {  
        System.out.print("Buenos días ");  
        System.out.print(nombre);  
        System.out.print(" ");  
        System.out.print(apellido);  
    }  
    public static void imprimeSaludoInformal(String nombre) {  
        System.out.println(" Hola" + nombre );  
    }  
    public static String saludoPersonal(String nombre, String apellido) {  
        return "Buenos días " + nombre + " " + apellido;  
    }  
}
```



# Métodos estáticos en otra clase (2)

---

```
import javax.swing.JOptionPane;

public class SaludosCliente {
    public static void main(String[] args) {
        Saludos.imprimeSaludo();
        String n = JOptionPane.showInputDialog("Nombre");
        String a = JOptionPane.showInputDialog("Apellido");
        Saludos.imprimeSaludoPersonal(n,a);
        Saludos.imprimeSaludoInformal(n);
        JOptionPane.showMessageDialog(null, Saludos.saludoPersonal(n,a));
        System.exit(0);
    }
}
```



# Paso de parámetros (1)

- 
- ▶ El encabezado contiene una lista de parámetros, que se utilizan de la misma manera que las variables dentro del cuerpo del método
  - ▶ La llamada de un método contiene los parámetros (actuales) que tienen que ser usados como argumentos para el método y los parámetros formales aparecen en la cabecera de la definición del método.
  - ▶ Cuando un método es llamado y así activado, los parámetros actuales deben estar destinados a los parámetros formales.



# Paso de parámetros (2)

## ► Suponga:

---

- **pa** un parámetro actual en una llamada al método,
  - **pf** el parámetro formal en el encabezado de la definición del método:
  - Cuando el método se activa:
    - El parámetro actual **pa** se evalúa (**pa** es una expresión).
    - Una ubicación de memoria se asocia al parámetro formal **pf**.
    - El valor de **pf** (la posición de memoria correspondiente) se inicializa con el valor calculado para **pa**.
  - Es decir, el parámetro formal **pf** se comporta como una variable local creada en el momento en que se llama al método, y se inicializa con el valor del parámetro actual correspondiente **pa**.
- 



# Paso de parámetros (3)

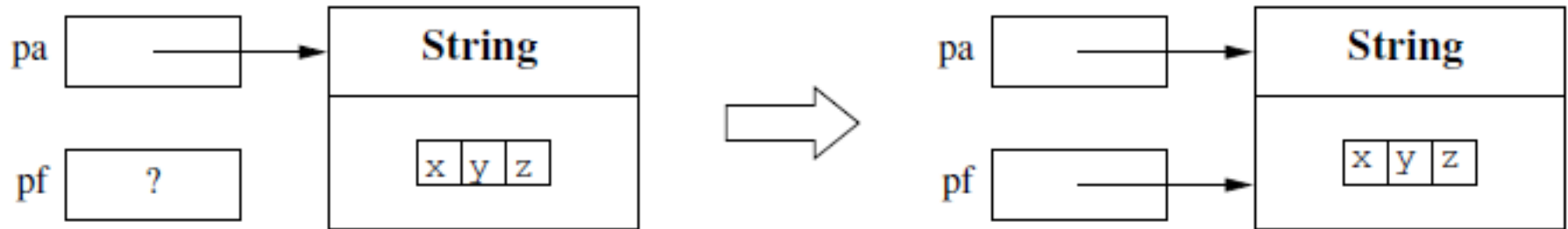
---

- ▶ Al final de la ejecución del método, la ubicación de memoria reservada para el parámetro formal se libera y el valor almacenado se pierde.
- ▶ El valor de una variable que aparece en la expresión **pa** no se modifica por la ejecución del método. Sin embargo, si tal valor es una referencia a un objeto, el método puede cambiar al objeto denotado por la referencia.



# Paso de parámetros (4)

- Un ejemplo de paso de parámetros para el caso en el que el parámetro es una referencia a un objeto.



# La sobrecarga de métodos (1)

---

- ▶ Java distingue métodos basados en su distintivo, y no sólo en su nombre.
- ▶ En la misma clase se pueden definir más de un método con el mismo nombre, pero se deben diferenciar en el número o tipo de sus parámetros formales.
- ▶ Esta característica se denomina sobrecarga de métodos.





# La sobrecarga de métodos (2)

---

```
public class Saludos2 {  
    public static void imprimeSaludo() {  
        System.out.println("¡Hola!");  
    }  
    public static void imprimeSaludo(String nombre) {  
        System.out.println("Hola" + nombre);  
    }  
}
```



# Clases vistas como módulos

- 
- ▶ Una clase como un módulo, se caracteriza por las siguientes características :
    - ▶ Los **servicios exportados**: los métodos públicos, es decir, los métodos visibles fuera de la clase;
    - ▶ **Interfaz**: las cabeceras de los métodos públicos;
    - ▶ Los **servicios importados**: otros métodos o clases utilizados para realizar la representación de los objetos y de los métodos de la clase;
    - ▶ **Estructura interna**: la representación de los objetos y la realización de los métodos de la clase.



# Definición de una clase

- ▶ En Java, la definición de una clase se caracteriza por:
  - ▶ El nombre que identifica a la clase.
  - ▶ Las **variables de instancia** (campos de datos) permiten almacenar datos dentro de los objetos;
  - ▶ Los **métodos** (campos de operación) pueden ser invocados en los objetos de la clase para realizar operaciones.



# Definición de una clase

## Sintaxis:

---

```
public class Nombre {  
    campo1  
    ...  
    campon}
```

- ▶ Los modificadores de acceso especifican:
    - ▶ qué campos deben ser **visibles** en la parte exterior de la clase, es decir, para los clientes de la clase (campos públicos);
    - ▶ qué campos deben permanecer ocultos a los clientes, ya que no son relevantes para ellos (campos privados).
-

# Definición de una clase

## Semántica:

---

- ▶ Define una clase.
- ▶ Los campos de datos (o variables de instancia) se utilizan para representar la estructura interna de los objetos de la clase.
- ▶ Los campos de operación (o métodos) se utilizan para realizar las funciones de la clase.



```
public class Persona {  
    // variables de instancia (campos de datos)  
    private String nombre;  
    private String residencia;  
  
    //métodos (campos de operación)  
    public String getNombre() {  
        return nombre;  
    }  
    public String getResidencia() {  
        return residencia;  
    }  
    public void setResidencia(String nuevaResidencia) {  
        residencia = nuevaResidencia;  
    }  
}
```

---



# Variables de instancia

- ▶ Las variables se definen en la clase fuera del cuerpo de los métodos:
  - ▶ Las variables se definen dentro de la clase, pero fuera de todos los métodos;
  - ▶ Las variables están precedidas por un modificador de acceso (generalmente privado);
  - ▶ Las **variables** siempre **se inicializan** cuando el **objeto es creado**, ya sea implícitamente (un valor por default), o explícitamente por el **constructor**.



# El uso de una clase definida (1)

- ▶ Una clase definida por el programador se utiliza de la misma manera que una clase predefinida.

```
public class ClaseClientePersona {  
    public static void main(String[] args) {  
// objeto      operador  
        Persona p1;  
        p1 = new Persona(); //constructor estándar  
        p1.setResidencia("Puebla");  
        System.out.println(p1.getResidencia());  
    }  
}
```



# El uso de una clase definida (2)

- ▶ La clase `ClienteClasePersona` es un cliente de la clase `Persona`.
- ▶ El cliente define el método `main ()` en el que:
  - ▶ se declara una variable local para `main ()`, de tipo `Persona` (del tipo que referencia a un objeto como una instancia de `Persona`);
  - ▶ se crea un nuevo objeto de la clase `Persona`, y se le asigna a `p1` una referencia a éste;
  - ▶ se llama al método `setResidencia()` de la clase `Persona` en el objeto denotado por `p1`, y se le pasa al método el parámetro actual "Puebla".
  - ▶ se llama al método `getResidencia()` en `p1` para mostrar la residencia del objeto denotado por `p1`.

# Reglas de acceso a los campos de una clase

---

- ▶ El acceso a los campos de una clase:
  - ▶ Los métodos correspondientes a las funcionalidades de interés para los clientes se declaran como públicos para que puedan ser visibles fuera de la clase.
  - ▶ Las variables de instancia y los métodos auxiliares que son necesarios para soportar esas funcionalidades, que no son de interés para los clientes, son visibles sólo dentro de la clase y se declaran como privados.

***Nota:*** Al conjunto de campos públicos de una clase se llama *interfaz pública de la clase*.

---



# Definición de métodos no estáticos

- ▶ **public tipodevuelto nombremetodo (parámetros)**
  - ▶ **public** indica que el método definido es accesible desde fuera de la clase.
  - ▶ **tipodevuelto** es el tipo del resultado devuelto por el método, o **void**, si el método no devuelve ningún resultado.
  - ▶ **nombremetodo** es el nombre del método.
  - ▶ **parámetros** es una lista de declaraciones de parámetros (tipo y nombre) separados por comas, cada parámetro es una variable, la lista de parámetros también puede estar vacía
  - ▶ El **cuerpo del método** (bloque) contiene las instrucciones que se ejecutarán cuando el método sea invocado.

***Nota:** un método no estático es similar a un estático, aunque la palabra **static** no aparece, lo cual indica que el método requiere un objeto de invocación.*

# Constructores

- ▶ Un constructor es un método de una clase que tiene el mismo nombre que la clase y no tiene un valor de retorno explícito (ni siquiera **void**).
- ▶ Un constructor sirve para inicializar las variables de instancia.
- ▶ Un constructor para la clase Persona con parámetros: nombre y residencia de la persona a definir.

```
public class Persona {
```

```
...
```

```
// constructor nombre-residencia
```


```
public Persona(String n, String r) {
```

```
    nombre = n;
```

```
    residencia = r;
```

```
}  
▶
```

```
public class Persona {  
    // variables de instancia (campos de datos)  
    private String nombre;  
    private String residencia;  
    // métodos (campos de operación)  
    // constructor nombre-residencia  
    public Persona(String n, String r) {  
        nombre = n;  
        residencia = r;  
    }  
    public String getNombre() {  
        return nombre;  
    }  
    public String getResidencia() {  
        return residencia;  
    }  
    public void setResidencia(String nuevaResidencia) {  
        residencia = nuevaResidencia;  
    }  
}
```



# Invocación de un constructor (1)

- ▶ Un constructor se llama automáticamente cuando un objeto se crea mediante el operador new.

*// objeto    operador    constructor   nombre-residencia*

```
Persona p = new Persona("Luz Sánchez", "Puebla");  
System.out.println(p.getResidencia());    // imprime "Puebla"
```

- ▶ Llama al constructor Persona (String n, String r), que crea un objeto de la clase Persona e inicializa nombre y residencia con los valores pasados como parámetros.
- ▶ La referencia al objeto recién creado se asigna a la variable p.



# Invocación de un constructor (2)

Persona p; // (1)

p = new Persona ("Luz Sánchez", "Puebla"); // (2)

- ▶ En (1), se define una variable p del tipo de referencia a un objeto Persona,
- ▶ En (2) se crea un nuevo objeto Persona, y se asigna la referencia a la variable p.
- ▶ El nuevo operador utiliza un constructor para crear un objeto y devuelve una referencia a éste. La referencia puede ser:
  - ▶ pasado como parámetro actual a un método que tiene un parámetro formal del tipo de referencia a Persona.
  - ▶ el resultado de un método cuyo valor devuelto es del tipo de referencia a Persona.

# Invocación de un constructor

---

- ▶ **Nota:** Es importante que todos los constructores se declaren como campos públicos de la clase. Si fueran declarados como privados, entonces cualquier intento de crear un objeto de una clase podría generar un error.





# Sobrecarga de constructores

- ▶ Java admite la sobrecarga de métodos, y un constructor es un caso especial de un método, es posible definir varios constructores para una clase.
- ▶ Se puede definir un constructor que ponga a null la residencia de persona.

*// nombre del constructor*

```
public Persona (String n) {  
    nombre = n;  
    residencia = null;  
}
```



# El uso de constructores

---

```
Persona p1 = new Persona ("Luz Sánchez");
```

*// llamando al constructor nombre*

```
Persona p2 = new Persona ("Laura Diaz", "México");
```

*// llamando al constructor nombre-residencia*

```
System.out.println(p1.getNombre());
```

*// imprime "Luz Sánchez"*

```
System.out.println(p2.getNombre());
```

*// imprime " Laura Diaz "*

- ▶ **Nota:** Cuando se crea un objeto mediante un operador new, el compilador determina qué constructor utilizar, basado en el número y tipo de parámetros especificados en la operación new.



# Constructor estándar

- ▶ Cuando se crea un objeto de una clase que no contiene la definición de un constructor (como la primera versión de la clase Persona), se llama al constructor estándar.
- ▶ El constructor estándar no tiene parámetros y se genera automáticamente por el compilador para todas aquellas clases que no contienen la definición de un constructor.
- ▶ Deja las variables de instancia inicializadas a sus valores por default, valor asignado automáticamente durante el tiempo de ejecución cuando la posición de memoria asociada a la variable está siendo reservada.



# Constructor estándar

- ▶ El constructor estándar se inhibe automáticamente por el compilador cuando se define cualquier constructor (con o sin parámetros) en la clase.
- ▶ Un constructor sin parámetros para la clase Persona:

```
public Persona () {    // constructor sin parámetros
    nombre = "Luz Sánchez";
    residencia = null;
}
```

**Nota:** No siempre tiene sentido definir un constructor sin parámetros para una clase. Un constructor sin argumentos para la clase Persona puede ser cuestionable.



# El parámetro formal implícito **this** (1)

- ▶ Todos los métodos de instancia (no estáticos) tienen un parámetro formal implícito indicado por **this**, que denota el objeto de invocación.
- ▶ Cuando se llama a un método, éste se enlaza al objeto de invocación (la referencia), que actúa como un parámetro actual.
- ▶ El parámetro **this** se utiliza para acceder a las variables de instancia y a los métodos del objeto de invocación.
- ▶ En general, se puede omitir **this**, como hasta ahora. Java lo inserta automáticamente cada vez que se utiliza una variable de instancia o un método de instancia de la clase.




# El parámetro formal implícito this (2)

```
public class Persona {  
    // variables de instancia (campos de datos)  
    private String nombre;  
    private String residencia;  
    //metodos (campos de operación )  
    public String getNombre() {  
        return this.nombre;  
    }  
    public String getResidencia() {  
        return this.residencia;  
    }  
    public void setResidencia(String nuevaResidencia) {  
        this.residencia = nuevaResidencia;  
    }  
}
```

# El uso de **this**

- ▶ Generalmente, **this** se utiliza cuando hay una variable local (o un parámetro formal) declarada dentro del método con el mismo nombre que una variable de instancia, lo cual permite distinguirlas.

```
public class Persona {  
    private String nombre;  
    private String residencia;  
    // constructor nombre-residencia  
    // this se utiliza para distinguir la variable de instancia de la variable local  
    public Persona(String nombre, String residencia) {  
        this.nombre = nombre;  
        this.residencia = residencia;  
    }  
    public String getNombre() {  
        return nombre;  
    }  
    public String getResidencia() {  
        return residencia;  
    }  
    public void setResidencia (String residencia) {  
        this.residencia = residencia;  
    }  
}
```






# **Metodología del diseño de una clase**

## **Realización de una clase**

---

1. A partir de la especificación de una clase, se identifican las propiedades y los servicios de la clase a realizar.
  2. Elegir una representación de los objetos de la clase, mediante la identificación de las variables de instancia necesarias.
  3. Elegir las cabeceras de los métodos públicos de la clase (interfaz de la clase). En este paso se decide la manera en que los clientes de la clase tienen que utilizar los objetos de la clase.
  4. Realizar el cuerpo de los métodos públicos, posiblemente mediante la introducción de métodos auxiliares para simplificar y estructurar el código.
- 

# Metodología del diseño de una clase

## Clientes de la clase

---

- ▶ Una vez que se realiza una clase, se puede realizar un cliente de la clase para verificar prácticamente cómo utilizar la clase .
  - ▶ Los clientes no tienen que conocer los cuerpos de los métodos públicos de la clase, ya que a los clientes sólo les importa lo que hacen los métodos y no cómo lo hacen.
  - ▶ Esto significa que se podría anticipar la realización de los clientes de la clase antes de tener realizados los cuerpos de los métodos públicos (y por tanto también antes de introducir métodos auxiliares).
- 



# Metodología del diseño de una clase

## Clientes de la clase

---

- ▶ En la práctica, después del paso 3, se puede realizar el esqueleto de la clase, es decir, la propia clase, en la que se tiene sólo los encabezados de los métodos públicos, en lugar de sus definiciones, y sin ningún método privado.
- ▶ El esqueleto de la clase es suficiente para realizar los clientes de la clase.



# El diseño de una clase

## Especificación

---

- ▶ Realizar una clase **Coche** en Java. Las propiedades (atributos) para un coche son: número de placa, modelo, color, y la persona propietaria del coche. Las dos primeras propiedades no se pueden modificar, la tercera y cuarta sí. Inicialmente, un coche no tiene propietario, éste puede ser asignado cuando el coche se vende.
- ▶ A partir de la especificación anterior, las funciones de la clase Coche son:
  - ▶ crear un objeto de la clase con atributos: placa, modelo y color inicializados a valores adecuados, y ningún propietario;
  - ▶ devolver el valor de cada una de las propiedades placa, modelo, color, y propietario; cambiar el color o el propietario.



# El diseño de una clase

## Representación de los objetos

---

```
public class Coche {  
    // representación de los objetos  
    private String placa;  
    private String modelo;  
    private String color;  
    private Persona propietario;  
    // métodos públicos que realizan las funcionalidades requeridas  
}
```



# El diseño de una clase

## Interfaz pública

- ▶ Elegir la interfaz de la clase Coche, a través de la cual los clientes pueden utilizar sus objetos.
- ▶ Para cada función hay que definir un método público y determinar su encabezado. Las funcionalidades son:
  - ▶ Crear un objeto de la clase con atributos: placa, modelo y color, inicializados adecuadamente, y ningún propietario.
    - ▶ Para crear un objeto de una clase, se utiliza un **constructor**, que debe inicializar las variables de instancia que representan placa, modelo y color, utilizando parámetros adecuados (las dos primeras propiedades no pueden cambiar). La variable de instancia propietario, en cambio, debe ser inicializado al valor null.



# El diseño de una clase

## Interfaz pública

- ▶ El encabezado del constructor:

```
public Coche(String p, String m, String c)
```

- ▶ Métodos públicos que devuelvan el valor de cada atributo. Los encabezados de estos métodos:

```
public String getPlaca()
```

```
public String getModelo()
```

```
public String getColor()
```

```
public Persona getPropietario()
```


- ▶ Para modificar el color y el propietario se introducen dos métodos cuyos encabezados son:

```
public void setColor(String nuevoColor)
```

```
public void setPropietario(Persona nuevoPropietario)
```

*//El esqueleto de la clase Coche*

```
public class Coche {  
    private String placa;                                // representación de los objetos  
    private String modelo;  
    private String color;  
    private Persona propietario;  
    public Coche (String p, String m, String c) {      // constructor  
        ...  
    }  
    public String getPlaca() {  
        ...  
    }  
    public String getModelo() {  
        ...  
    }  
    public String getColor() {  
        ...  
    }  
    public Persona getPropietario() {  
        ...  
    }  
    public void setColor(String nuevoColor) {  
        ...  
    }  
    public void setPropietario (Persona nuevoPropietario) {  
        ...  
    }  
}
```





# El diseño de una clase

## Métodos

- ▶ Los métodos y sus cuerpos. Empezar por el **constructor**:

```
public Coche(String p, String m, String c) {  
    placa = p;  
    modelo = m;  
    color = c;  
    propietario= null;  
}
```

- ▶ **Nota:** Si se omite la declaración *propietario=null*; automáticamente el *propietario* se inicializa a *null*, que es el valor por default para las referencias a objetos.
- ▶ Realizar los otros métodos de una manera similar.

# Métodos

```
public Coche(String p, String m, String c) {  
    placa = p;  
    modelo = m;  
    color = c;  
    propietario = null;  
}  
// otros métodos públicos  
public String getPlaca() {  
    return placa;  
}  
public String getModelo() {  
    return modelo;  
}  
public String getColor() {  
    return color;  
}  
public Persona getPropietario() {  
    return propietario;  
}  
public void setColor(String nuevoColor) {  
    color = nuevoColor;  
}  
public void setPropietario(Persona nuevoPropietario) {  
    propietario = nuevoPropietario;  
}
```



```
}
```

# El diseño de una clase

## Cliente

- ▶ Un cliente, ServiciosCoche, de la clase Coche. La clase ServiciosCoche contiene dos métodos estáticos:
  - ▶ pintura() con parámetros un objeto Coche y un objeto String que representa el color nuevo del coche, y modifica el objeto Coche cambiando su color;
  - ▶ registramodelo() con parámetros dos objetos String que representan la placa y el color; y devuelve una referencia a un nuevo objeto Coche, cuyo modelo es "AudiA1" y cuya placa y color son especificados por los parámetros.

```
public class ServiciosCoche {  
    public static void pintura(Coche coche, String color) {  
        coche.setColor(color);  
    }  
    public static Coche registramodelo(String pla, String col) {  
        return new Coche(pla, "AudiA1", col);  
    }  
}
```

```
public class Principal {  
    // métodos auxiliares  
    private static void imprimeDatosCoche(Coche a) {  
        System.out.println("Coche: " + a.getPlaca() + ", "  
+ a.getModelo()+ ", "  
+ a.getColor());  
    }  
    private static void imprimeDatosPropietario(Coche a) {  
        System.out.println("Propietario: " + a.getPropietario().getNombre() + ", "  
+ a.getPropietario().getResidencia());  
    }  
    public static void main(String[] args) {  
        Coche a = new Coche("313", "Fiat 500", "Negro");  
        imprimeDatosCoche(a);  
        Persona p = new Persona("Leonardo Vazquez", "Monterrey");  
        a.setPropietario(p);  
        imprimeDatosPropietario(a);  
        ServiciosCoche.pintura(a, "Rojo brillante");  
        imprimeDatosCoche(a);  
        Coche b = ServiciosCoche.registramodelo("131", "Azul Cobalto");  
        imprimeDatosCoche(b);  
        Persona c = new Persona("Juan Gutierrez", "Puebla");  
        b.setPropietario(c);  
        imprimeDatosPropietario(b);  
    }  
}
```