



Introducción al Paradigma de Objetos

Clase 2



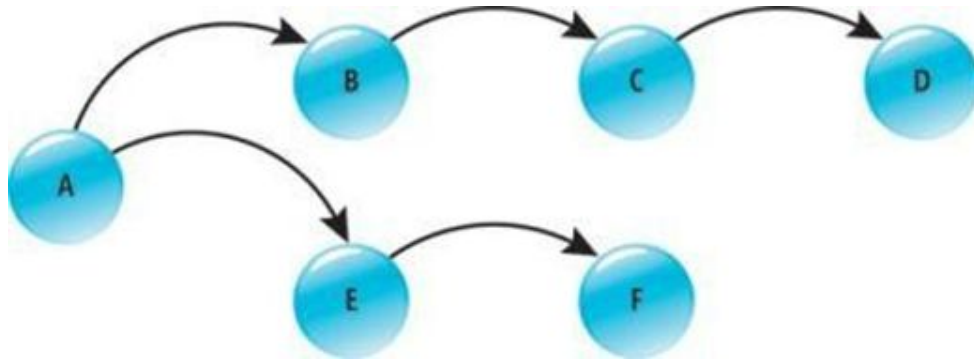
Roberto Andrés Villa
ravilla.educacionit@gmail.com

Repaso Clase 1

- **Paradigma:** es un modelo o patrón en cualquier disciplina científica para intentar resolver un problema.
- Un **Paradigma de Programación** es una propuesta tecnológica que es adoptada por una comunidad de programadores.
- El **Paradigma de Programación Orientada a Objetos** es la
- implementación de un Paradigma de Programación
- Vimos un poco de historia...
- Les vendí el paradigma orientado a objetos..

Repaso Clase 1

- Modelo: Abstracción de la Realidad
- El Modelado Orientado a Objetos: Consiste en interpretar un sistema como partes independientes que se comunican entre sí. Las partes independientes se denominan Objetos. La comunicación entre los objetos se realiza a través de Mensajes.



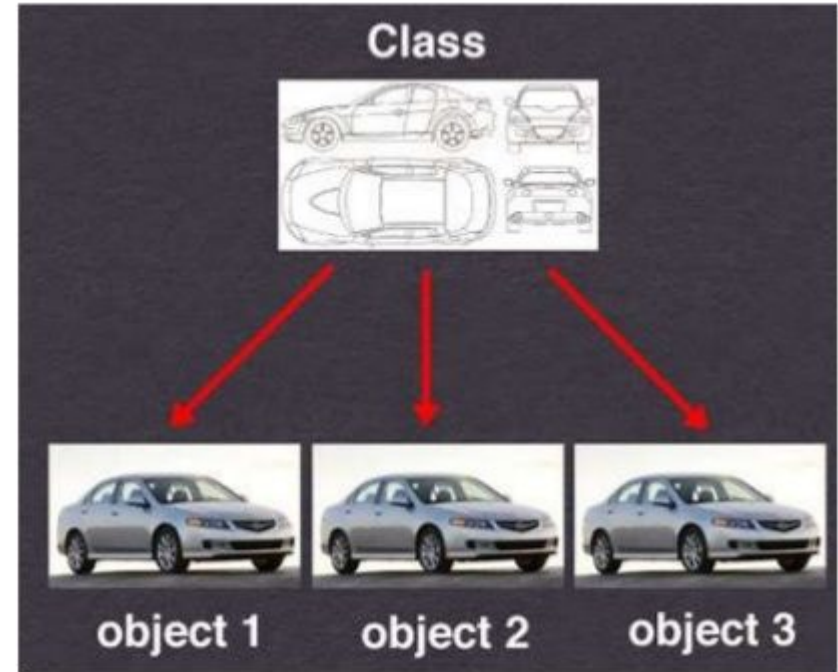
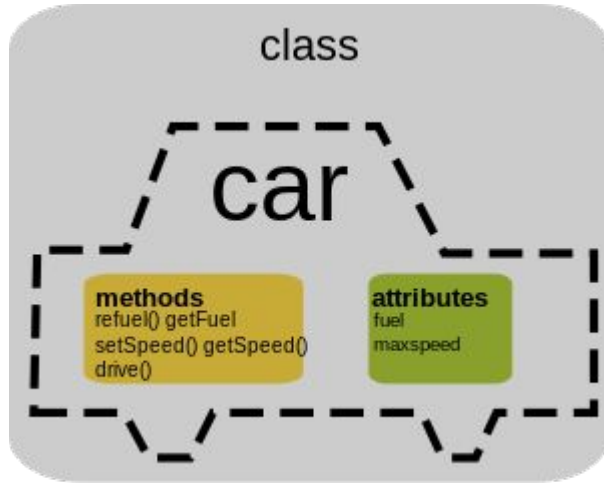
Repaso Clase 1

Un **Paradigma** es un modelo o patrón en cualquier disciplina científica.

Un **Paradigma de Programación** es una propuesta tecnológica que es adoptada por una comunidad de programadores cuyo núcleo central es incuestionable en cuanto a que unívocamente trata de resolver uno o varios problemas claramente delimitados.

El **Paradigma de Programación Orientada a Objetos** es la implementación de un Paradigma de Programación

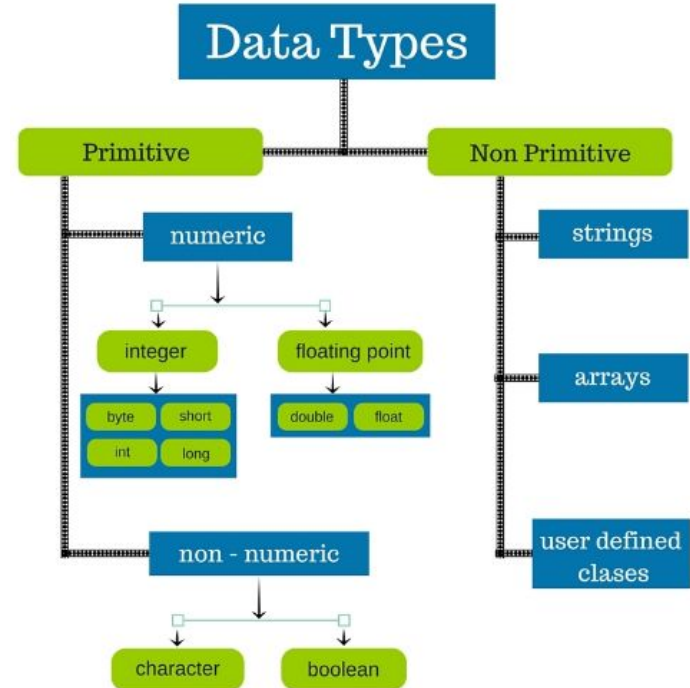
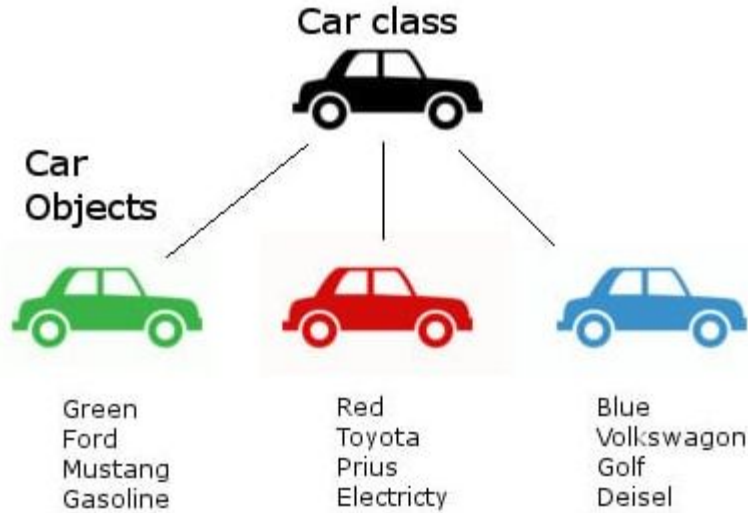
Repaso Clase 1



Representación generica. Se detecta como sustantivos en singular.

Repaso Clase 1

Atributos



Repaso Clase 1

Operaciones

Operaciones: Acciones en una clase, definen su comportamiento.

Pueden recibir valores de entrada.
Pueden o no devolver valores.

Se detectan como verbos



Sobrecarga de Operadores

Es la aparición de métodos dentro de una misma clase que se llaman igual, pero realizan acciones (levemente) diferentes

Normalmente varían en cantidad y/o tipo de parámetros

Por ejemplo, en la clase Auto podríamos encontrar variantes del método `acelerar()`, por ejemplo:

- `acelerar()` acelera 10km/h
- `acelerar(int km)` acelera de acuerdo al parametro “km”
- `acelerar(int km, boolean tieneNitro)` idem caso anterior, pero si el parámetro “tieneNitro” es verdadero acelera el doble!

Relaciones simples

Se produce cuando una clase se relaciona con otra clase

La relación **es única**

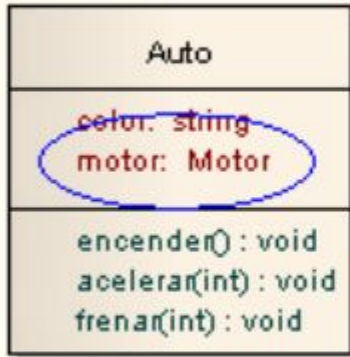
Por ejemplo: Auto tiene una relación simple con Motor, por que un auto puede tener un motor únicamente

Se puede presentar como:

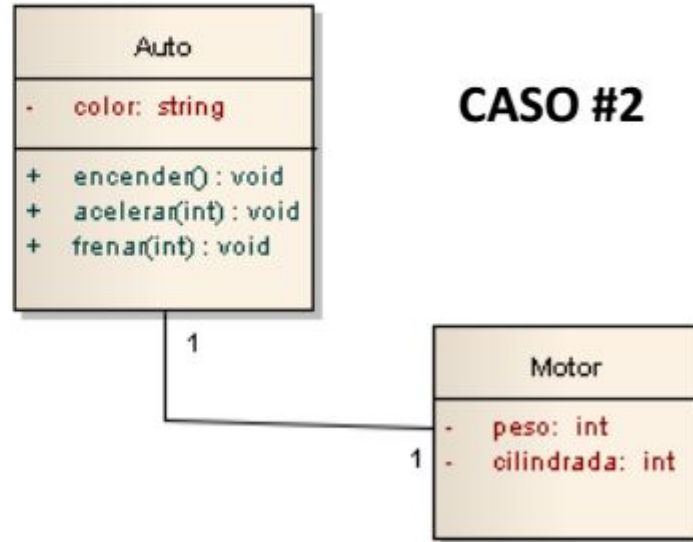
“... un Auto tiene un Motor ...”

Representar relaciones simples

CASO #1



CASO #2



Ejercicio #4

A partir de las clases detectadas previamente, identifique las relaciones simples que existen entre las mismas.

TIP: Para este ejercicio asumiremos que cada cliente (cualquier tipo) puede tener una única cuenta

Ejercicio #4 – Solución

ClientePyme **tiene una** CuentaCorriente

ClienteCorporacion **tiene una** CuentaCorriente

ClienteIndividuo **tiene una** CajaDeAhorro

Banco **tiene un** DirectorGeneral

Sucursal **tiene un** DirectorDeSucursal

Relaciones Múltiples

Se produce cuando una clase se relaciona con una o muchas otras clases. La relación **es múltiple**.

Por ejemplo: Auto tiene una relación múltiple con Rueda, por que un auto puede tener varias ruedas

Se puede presentar como:

“... un Auto **tiene de una a muchas** Rueda(s) ...”

“... una Universidad **tiene de uno a muchos** Alumno(s) ...”

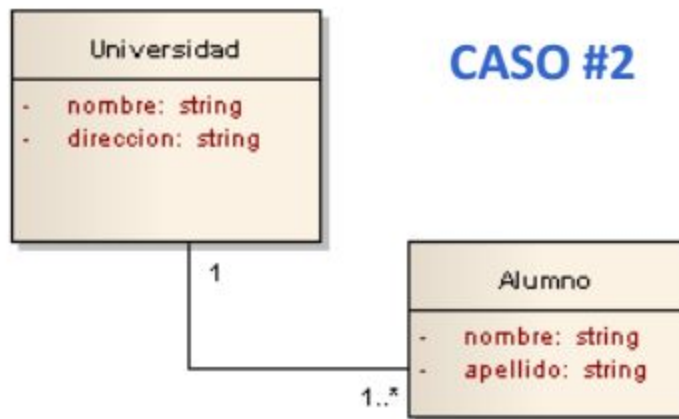
Se dice que una Universidad **tiene una colección** de alumnos

Representación relaciones múltiples

CASO #1



CASO #2



Ejercicio #5

A partir de las clases detectadas previamente, identifique las relaciones múltiples que existen entre las mismas

TIP: Para este ejercicio asumiremos que cualquier tipo de cliente puede tener mas de una cuenta corriente

Ejercicio #5 – Solución

Banco **tiene de uno a muchas** Sucursal(es)

Banco **tiene de uno a muchos** DirectorRegional(es)

ClienteCorporacion **tiene de uno a muchas** CuentaCorriente(s)

GrupoFinanciero **tiene de uno a muchos** Banco(s)

Sucursal **tiene de uno a muchos** Servicio(s)

Visibilidad

Es la posibilidad de “ver” un atributo o método

Si un atributo o método es **privado (-)** : solo puede verse dentro de la clase

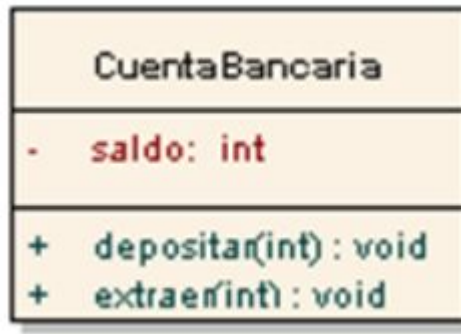
Si un atributo o método es **publico (+)** : puede verse desde otras clases

La visibilidad es establecida por los **modificadores de visibilidad**: **private** y **public**

Representación de visibilidad



El atributo velocidad es privado, solo puede modificarse a través de los métodos acelerar() y frenar()



El atributo saldo es privado, solo puede modificarse a través de los métodos depositar() y extraer()

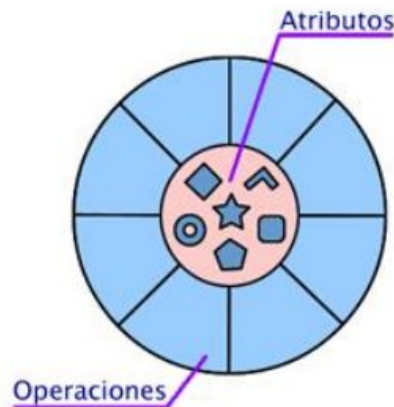
Encapsulamiento

Es el ocultamiento del estado de un objeto

El estado (atributos) podrá accederse únicamente a través de sus operaciones (métodos)

En la clase, los **atributos** deben ser **privados** y los **métodos** para acceder a los atributos deben ser **públicos**

El atributo **saldo** esta **encapsulado**, solo puede accederse a través de los métodos depositar() y extraer()



CuentaBancaria
- saldo: int
+ depositar(int): void
+ extraer(int): void

Setters y Getters

Son **métodos de acceso públicos** a atributos privados

Representan la única forma de acceder a los atributos

Setter: método utilizado para setear un valor a un atributo

Getter: método utilizado para obtener un valor de un atributo

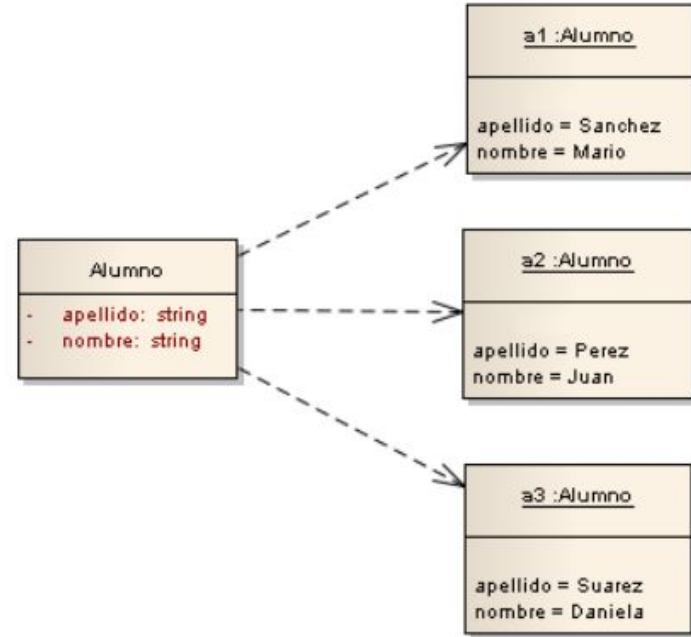
Los IDEs generalmente permiten generar los setters y los getters de forma automática!

Banco
<ul style="list-style-type: none">- nombre: String- cantidadDeEmpleados: int- cantidadDeSucursales: int- fechaDeConstitucion: Date
<ul style="list-style-type: none">+ getCantidadDeEmpleados() : int+ setCantidadDeEmpleados(int) : void+ getCantidadDeSucursales() : int+ setCantidadDeSucursales(int) : void+ getFechaDeConstitucion() : Date+ setFechaDeConstitucion(Date) : void+ getNombre() : string+ setNombre(string) : void

Objetos vs. Clases

La clase representa un concepto, es un molde, una plantilla

Los objetos representan instancias de una clase. Sería como tomar una plantilla (una clase) y personalizarla (completar sus atributos)



Constructores para construir objetos

Los objetos son contruidos a partir de una clase. Todos los objetos dependen de una clase.

Para construir un objeto es necesario utilizar un **constructor**.

El constructor es “un método” de la clase que se invoca al construir un objeto, y en su interior tiene un conjunto de acciones a realizar.

El constructor tiene el mismo nombre que la clase, y para invocarlo hay que utilizar una palabra clave del lenguaje de programación que se denomina **new**

FORMA GENERICA: NombreDeClase nombreDeObjeto = new Constructor();

EJEMPLO: Auto a = new Auto();

Ejercicio #6

Asumiendo que cada clase cliente cuenta con dos constructores (un constructor vacío y un constructor que recibe como parámetros un identificador, la razón social, y la dirección) y un método `informarDatos()` el cual informa el valor de sus atributos.

Suponiendo que se ejecuta el siguiente código, cual es la salida en pantalla?

```
ClientePyme c1 = new ClientePyme();  
c1.informarDatos();
```

CASO #1

```
ClientePyme c2 = new ClientePyme(3020,  
                                "Pirulo SRL", "Av Maipu 2587");  
c2.informarDatos();
```

CASO #2

```
ClienteCorporacion c3 = new ClienteCorporacion(3084, "Ford SA",  
                                                "Lavalle 1256 piso 10 oficina 30")  
c3.informarDatos();
```

CASO #3

Constructores - Conceptos Avanzados

Como se llama cuando una clase tiene más de un constructor? **Sobrecarga de constructores**

Una clase **debe tener al menos un constructor**. Si no se agrega un constructor, normalmente se asume que posee el constructor vacío

Una clase puede tener todos los constructores que sean necesarios

Que ocurre si hay dos constructores con la misma firma? Por ejemplo:

```
Alumno(String n){  
    nombre = n;  
}  
  
Alumno(String a){  
    apellido = a;  
}
```

NO FUNCIONA!

Al igual que los métodos, si tienen misma cantidad de parámetros, deben tener diferente tipo de dato

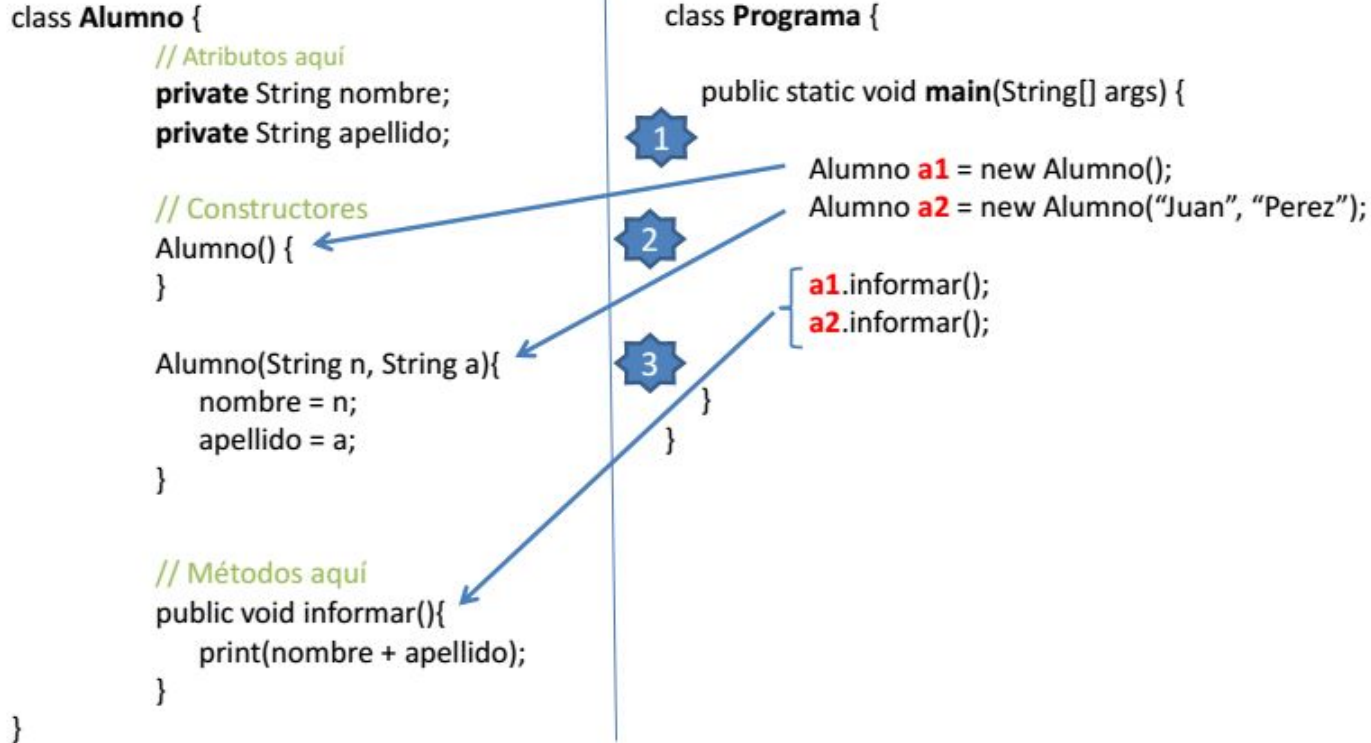
¿El huevo o la gallina?

Quien instancia el primer objeto?

La clase Programa a través de un método llamado `main()`, que se invoca automáticamente al ejecutar nuestra aplicación

```
class Programa {  
    public static void main(String[] args) {  
        // código aquí  
        // todo el código ubicado aquí se ejecuta automáticamente  
        // al ejecutar nuestra aplicación  
    }  
}
```

Interacción entre “Programa” y alumnos





¿Preguntas?



¡Gracias!

(Por favor apagar computadoras, revisar elementos personales...)

Sobrecarga de Operaciones – Codificación

```
class Auto {  
    // Atributos aquí  
    int velocidad;  
  
    // Métodos aquí  
    void acelerar() {  
        velocidad = velocidad + 10;  
    }  
  
    void acelerar(int km) {  
        velocidad = velocidad + km;  
    }  
}
```

```
void acelerar(int km, boolean tieneNitro){  
    if(tieneNitro == false){  
        acelerar(km);  
    } else {  
        acelerar(km*2);  
    }  
}
```

Ejercicio #4 - Codificación

```
class ClientePyme {  
    // Atributos aquí  
    String razonSocial;  
    CuentaCorriente cuenta;  
}  
  
class Banco {  
    // Atributos aquí  
    String nombre;  
    GerenteGeneral gerente;  
}
```

Ejercicio #5 – Codificación

```
class ClienteCorporacion {  
    // Atributos aquí  
    String razonSocial;  
    Collection<CuentaCorriente> cuentas;  
}  
  
class Banco {  
    // Atributos aquí  
    String nombre;  
    GerenteGeneral gerente;  
    Collection<Sucursal> sucursales;  
    Collection<DirectorRegional> directores;  
}
```


Visibilidad – Codificación

```
class Auto {  
    // Atributos aquí  
    private int velocidad;  
    // Métodos aquí  
    public void acelerar() {  
        velocidad = velocidad + 10;  
    }  
    public void acelerar(int km) {  
        velocidad = velocidad + km;  
    }  
}
```

```
    public void acelerar(int km, boolean tieneNitro){  
        if(tieneNitro == false){  
            acelerar(km);  
        } else {  
            acelerar(km*2);  
        }  
    }  
}
```

Getters y Setters - Codificación

```
class Banco {  
    // Atributos aquí  
    private String nombre;  
    private int cantidadDeEmpleados;  
    // Métodos aquí  
    public void setNombre(String n) {  
        nombre = n;  
    }  
    public String getNombre() {  
        return nombre;  
    }  
}
```

```
    public void setCantidadDeEmpleados(int c) {  
        cantidadDeEmpleados = c;  
    }  
    public int getCantidadDeEmpleados() {  
        return cantidadDeEmpleados;  
    }  
}
```