



Introducción al Paradigma de Objetos

Clase 3



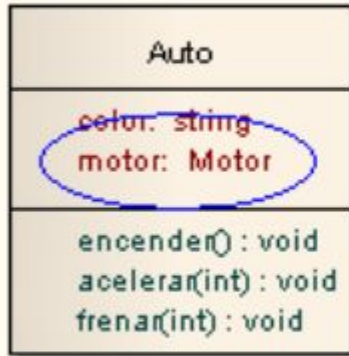
Roberto Andrés Villa
ravilla.educacionit@gmail.com

Repaso Clase 2

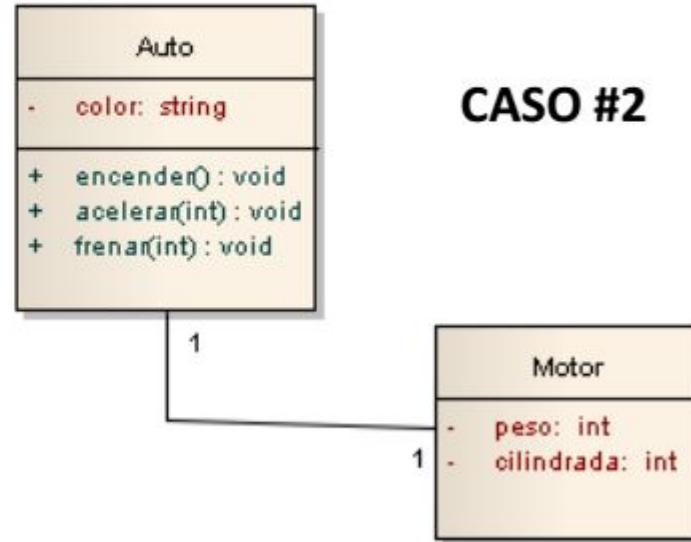
- **Sobrecarga de operaciones:** Es la aparición de métodos dentro de una misma clase que se llaman igual, pero realizan acciones (levemente) diferentes
- **Relaciones Simples:** Se produce cuando un objeto se relaciona con otro objeto
- **Relaciones Múltiples:** Se produce cuando un objeto se relaciona con uno o muchos objetos. La relación **es múltiple**
- **Visibilidad:** Es la posibilidad de “ver” un atributo o método
- **Encapsulamiento:** Es el ocultamiento del estado de un objeto
- **Getters y Setters:** Métodos “Especiales” para obtener o setear atributos
- **Constructores:** Inicialización de objetos

Representar relaciones simples

CASO #1



CASO #2

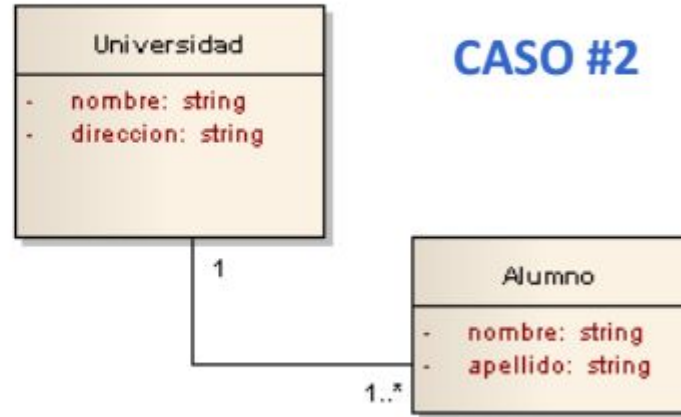


Representación relaciones múltiples

CASO #1



CASO #2



Interacción entre Objetos - Ejemplo

Asumiendo que un cliente individual utiliza el cajero automático para realizar un deposito de \$2500, ingresa su dni numero 27.014.589 como pin y realiza el deposito.

La interacción básica sería:

// Obtiene el cliente

```
ClienteIndividuo c = new ClienteIndividuo("27014589");
```

// Obtiene la caja de ahorro del cliente

```
CajaDeAhorro cda = c.obtenerCajaDeAhorro();
```

// Realiza el deposito

```
cda.depositar(2500);
```

El código completo correspondiente se presenta a continuación:

Interacción entre Objetos - Ejemplo

```
class Programa {  
    public static void main(String[] args) {
```

```
        ClienteIndividuo c1 = new ClienteIndividuo("27014589");  
        CajaDeAhorro cda = c1.obtenerCajaDeAhorro();  
        cda.depositar(2500);  
    }  
}
```

```
class CajaDeAhorro {  
    // Atributos aquí  
    private float saldo;  
  
    // Metodos aquí  
    public void depositar(float monto) {  
        saldo = saldo + monto;  
    }  
}
```

```
class ClienteIndividuo {
```

```
    // Atributos aquí
```

```
    private String dni;  
    private CajaDeAhorro cuenta;
```

```
    // Constructores
```

```
    ClienteIndividuo(String d) {  
        dni = d;
```

```
        // busca en la Base de Datos los datos  
        // de este cliente según el dni y  
        // completa los atributos  
    }
```

```
    // Métodos aquí
```

```
    public CajaDeAhorro obtenerCajaDeAhorro(){  
        return cuenta;  
    }  
}
```

1

2

3

Ejercicio #7 – Uso de Objetos

Suponiendo que un cliente corporativo desea realizar una extracción de una cuenta corriente, teniendo en cuenta los siguientes datos:

- el cuit de la empresa es 30-12345678-1
- desea extraer \$20.000
- en la cuenta hay disponible \$18.000
- el giro en descubierto es de \$5.000

Realizar el código en forma genérica utilizando las clases Programa, ClienteCorporativo y CuentaCorriente

TIP: el giro en descubierto es el monto máximo que el banco le presta al cliente en caso de que el saldo en su cuenta sea cero. Es decir que si el cliente no tiene fondos en la cuenta, y desea realizar una extracción o un pago, puede utilizar automáticamente hasta \$5.000

Ejercicio #7 – Codificación

Código de la clase Programa

```
class Programa {  
    public static void main(String[] args) {  
  
        // Obtiene el cliente de acuerdo a su cuit  
        ClienteCorporativo cliente = new ClienteCorporativo("30-12345678-1");  
  
        // Obtiene la cuenta corriente del cliente CuentaCorriente  
        CuentaCorriente cc = cliente.obtenerCuentaCorriente();  
  
        // Realiza la extracción  
        cc.extraer(20000);  
    }  
}
```

Ejercicio #7 – Codificación

Código de la clase ClienteCorporativo

```
class ClienteCorporativo {  
    // Atributos aquí  
    private String cuিত;  
    private CuentaCorriente cuenta;  
    // Constructores  
    public ClienteCorporativo(String c)  
    {  
        cuিত = c;  
        // busca en la Base de Datos los datos  
        // de este cliente según el cuিত y  
        // completa los atributos  
    }  
    // Métodos aquí  
    public CuentaCorriente obtenerCuentaCorriente()  
    {  
        return cuenta;  
    }  
}
```

Ejercicio #7 – Codificación

Código de la clase CuentaCorriente

```
class CuentaCorriente {  
    private float saldo;  
    private float saldoDescubierto;  
  
    public void extraer(float monto) {  
        if ( saldo >= monto ) {  
            saldo = saldo - monto;  
            print("Extraccion ok");  
        } else {  
            montoExcedente = monto - saldo;  
            if ( montoExcedente <= saldoDescubierto) {  
                saldo = 0;  
                saldoDescubierto = saldoDescubierto - montoExcedente;  
                print("Extraccion ok");  
            } else {  
                print("No hay fondos suficientes");  
            }  
        }  
    }  
}
```

The Four Pillars



Mecanismo de Herencia

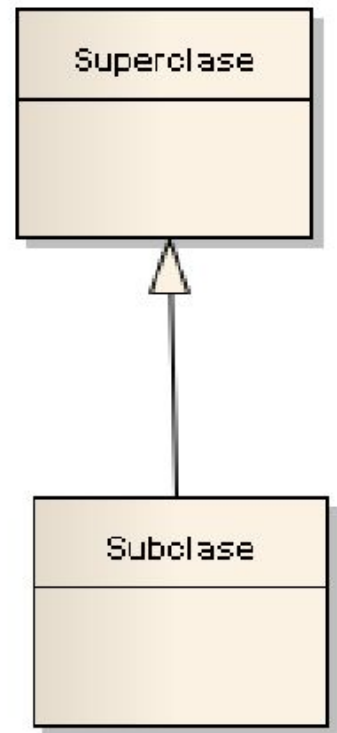
Mecanismo que permite que una clase "herede de otra clase" o "extienda otra clase", **recibiendo o heredando atributos y operaciones** de su clase "padre".

La **clase principal** se denomina: superclase o clase padre

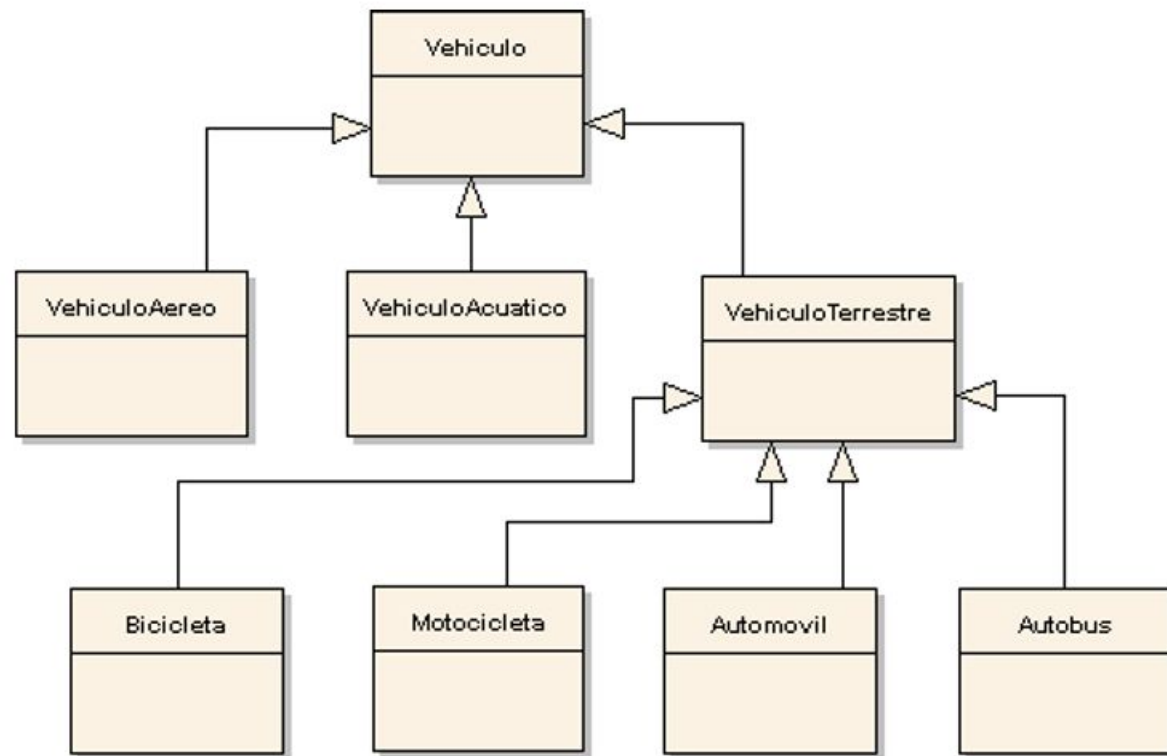
La **clase que hereda** se denomina: subclase o clase hija o clase derivada

La relación se interpreta como **“ES UN”**

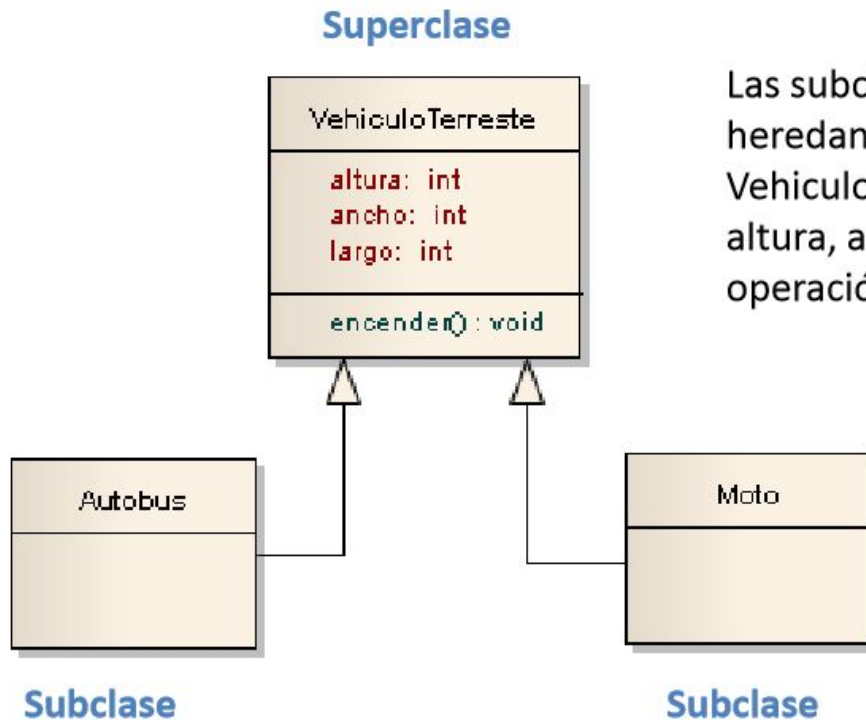
Ejemplo: un Auto **ES UN** VehiculoTerrestre



Ejemplo de Herencia



Ejemplo de Herencia



Las subclases **Autobus** y **Moto** heredan de la superclase **VehiculoTerreste** los atributos `altura`, `anchura`, `largo`, y también la operación `encender()`

Ejercicio #8

A partir de las clases detectadas previamente, identificar las relaciones de herencia que existe y escribirlos en un diagrama UML

**Banco – Sucursal – GrupoFinanciero – Servicio – ClientePyme – ClienteCorporacion –
ClienteIndividuo – CuentaComercial – CajaDeAhorro – CuentaCorriente – DirectorGeneral –
DirectorRegional – DirectorDeSucursal**

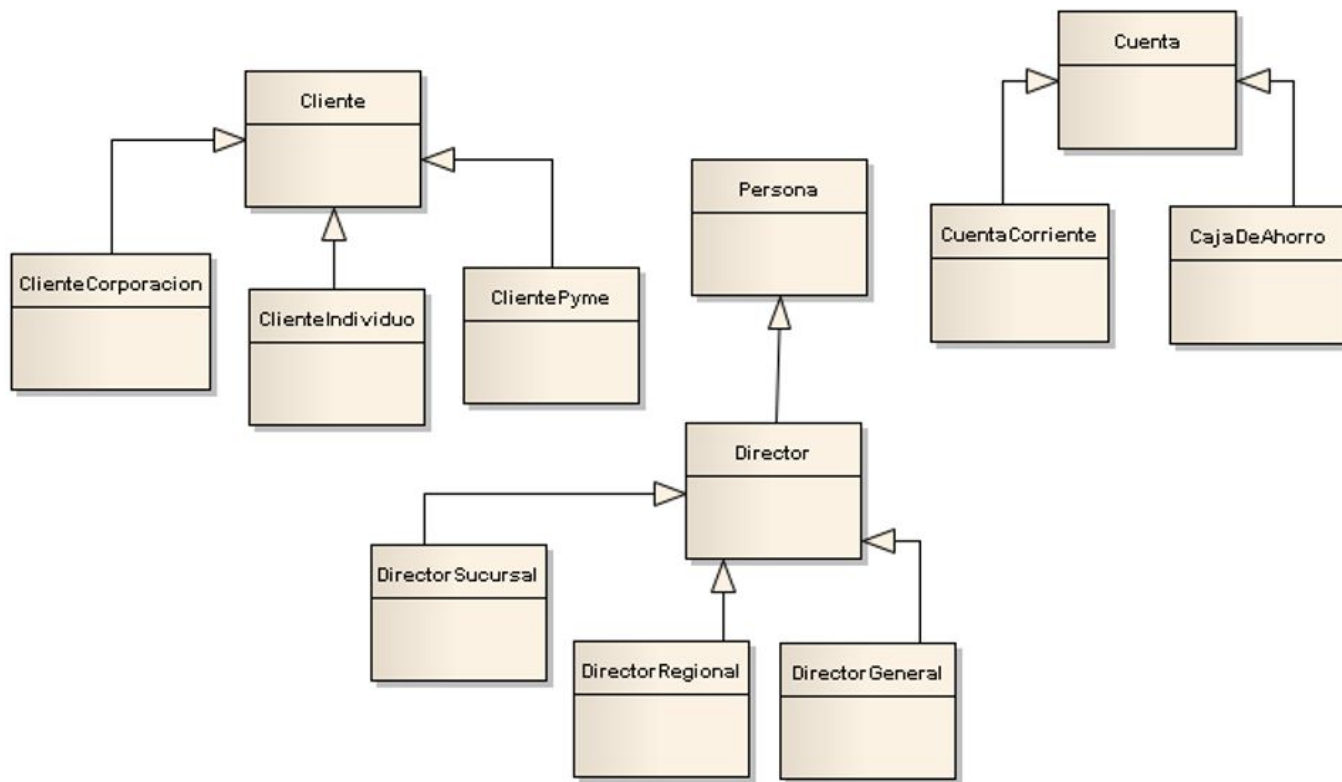
TIP: Si consideras útil y/o necesario, puedes proponer nuevas clases.

Observacion: Sigamos pensando que “DirectorRegional”, “DirectorGeneral”, etc son clases válidas

Ejercicio #8

**Banco – Sucursal – GrupoFinanciero – Servicio – ClientePyme – ClienteCorporacion – ClienteIndividuo –
CuentaComercial – CajaDeAhorro – CuentaCorriente – DirectorGeneral – DirectorRegional – DirectorDeSucursal**

Ejercicio #8 - Solución



Ejercicio #8 – Codificación

```
class Cliente {  
    // Atributos  
    String cuit;  
    String direccion;  
  
    // Constructores  
    // Métodos  
}
```

```
class ClientePyme extends Cliente {  
    // Atributos - Los atributos heredados no se vuelven a codificar!  
    String razonSocial;  
  
    // Constructores  
    // Métodos  
}
```

Clases Abstractas y Concretas

Las Clases Concretas

- son clases que se pueden instanciar
- por ejemplo la clase Alumno existe para generar diversos objetos del tipo Alumno

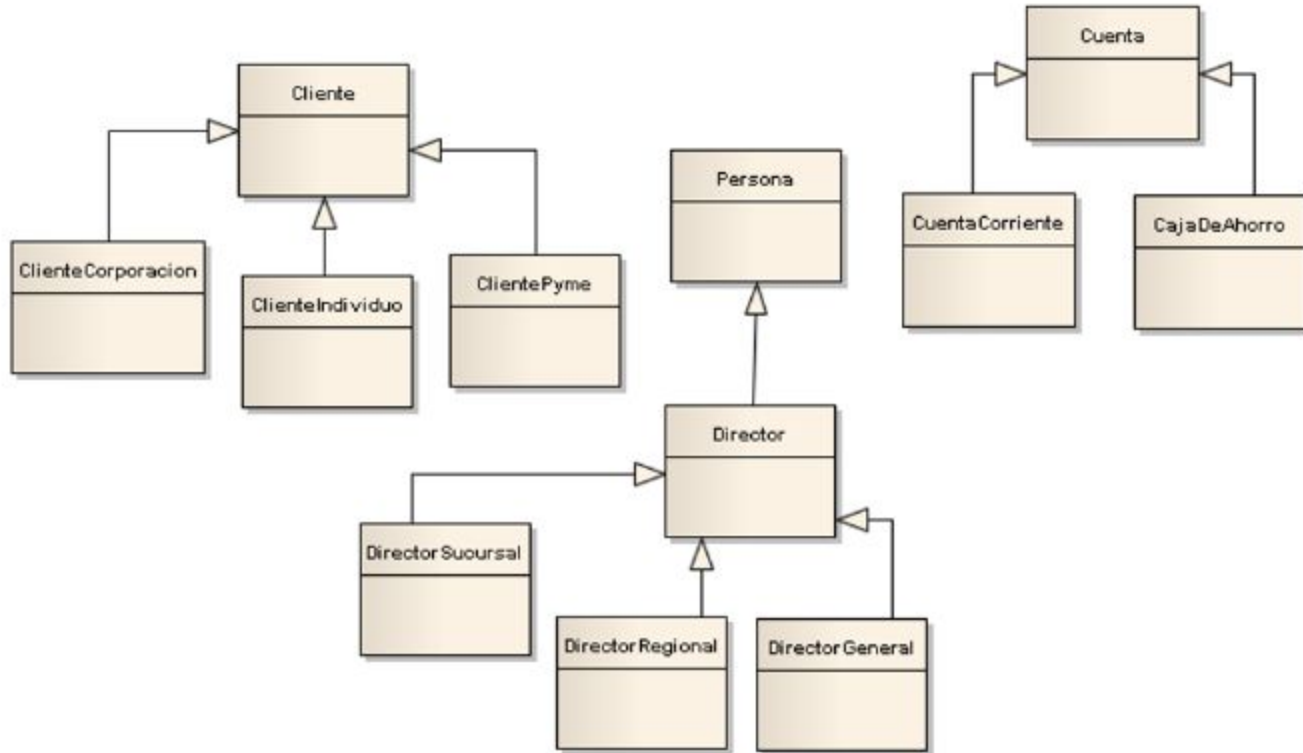
Las Clases Abstractas

- son clases que no se pueden instanciar
- representan conceptos muy genéricos de la realidad
- por ejemplo la clase Vehiculo o la clase Persona son conceptos muy abstractos, seria difícil pensar en armar un objeto a partir de estas clases

Codificación:

```
abstract class Persona {  
    // Atributos aquí  
    // Métodos aquí  
}
```

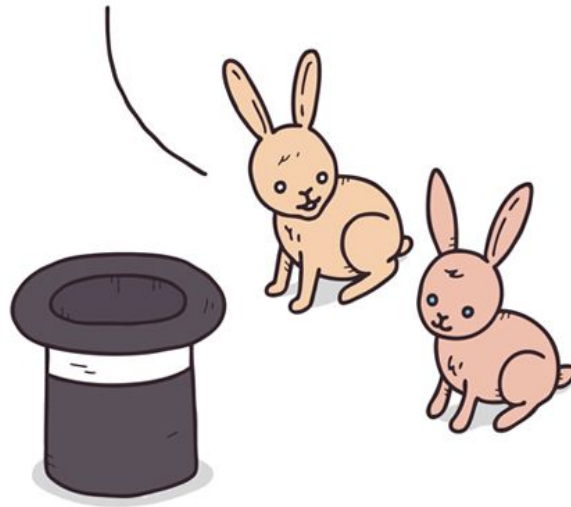
Clases Abstractas y Concretas - Ejemplo



The Four Pillars



THIS IS WHERE
THE MAGIC HAPPENS



» »

Polimorfismo

En programación orientada a objetos, el **polimorfismo o poliformismo** se refiere a la propiedad por la que es posible **enviar mensajes sintácticamente iguales a objetos de tipos distintos**. El único requisito que deben cumplir los objetos que se utilizan de manera polimórfica es saber responder al mensaje que se les envía.

Una posible definición...

2 Tipos de Polimorfismo: sin redefinición y con redefinición

Polimorfismo sin redefinición: Una clase que posee varios métodos llamados iguales pero con diferentes firmas. También llamado Sobrecarga de Operaciones

Auto	
-	color: string
+	acelerar(): void
+	acelerar(int): void
+	acelerar(int, boolean): void
+	frenar(int): void
+	encender(): void



3 versiones del método acelerar()

Otra definición (A mi gusto, más acertada)

- Un método está sobrecargado si dentro de una clase existen dos o más declaraciones de dicho método con el mismo nombre pero con parámetros distintos, por lo que no hay que confundirlo con polimorfismo.
- En definitiva: La sobrecarga se resuelve en tiempo de compilación utilizando los nombres de los métodos y los tipos de sus parámetros; el polimorfismo se resuelve en tiempo de ejecución del programa, esto es, mientras se ejecuta, en función de la clase a la que pertenece el objeto.

Polimorfismo - Codificación

```
class CajaDeAhorro {  
    // Atributos  
    float saldo;  
    // Métodos  
    public void extraer(int monto) {  
        saldo = saldo - monto;  
    }  
}  
  
class CajaDeAhorroDescubierto extends CajaDeAhorro {  
    // Atributos  
    float saldoDescubierto;  
    // Métodos  
    public void extraer(int monto) {  
        // este método se vuelve a escribir, misma firma pero diferente cuerpo  
        // aquí el código que contemple la extracción con saldo descubierto  
    }  
}
```

The Four Pillars



Métodos de Instancia y de Clase

Métodos de Instancia: son métodos que pertenecen a un objeto en particular e impactan en el comportamiento de ese objeto únicamente

Métodos de Clase: son métodos que pertenecen a la clase y no a un objeto o instancia de clase. Esto significa que son métodos compartidos por todos los objetos. También llamados métodos estáticos. Para definir un método estático se utiliza la palabra clave static

```
class CajaDeAhorroConDescubierto {  
    // Atributos aquí  
    public float saldo;  
    private static int descubierto = 5000;  
    // Metodos aquí  
    public static int leerDescubierto() {  
        return descubierto;  
    }  
}
```

CajaDeAhorroConDescubierto	
-	<u>descubierto: int</u>
+	<u>leerDescubierto(): int</u>
+	<u>modificarDescubierto(int): void</u>

Generación Automática de Código

El mecanismo de generación automática de código se denomina **Ingeniería Directa**

Existen herramientas que permiten generar código fuente a partir de diagramas, por ejemplo el Enterprise Architect

Estas aplicaciones permiten generar código fuente en diversos lenguajes, como ser: C#, VB.NET, PHP, Java, Actionscript, y mas

Finalmente...

LA PRÁCTICA HACE
LA PERFECCIÓN



FIN

¡Muchas Gracias!

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