

Ingeniería de Software.

Conceptos Fundamentales de  
Tecnología de Objetos.

# Principios de la Orientación a Objetos.

Los principios de Orientación a Objetos (OO) afectan todo el proceso de desarrollo:

- Los seres humanos piensan en términos de sustantivos (objetos) y verbos (acciones de los objetos).
- En el proceso de desarrollo de software con Orientación a Objetos, el sistema se modela utilizando los conceptos de OO.
- El UML es muy adecuado para representar modelos mentales.
- Los lenguajes de programación OO acercan la implementación al modelo mental.
- El UML representa un puente entre el modelo mental y la implementación.

## Principios de la Orientación a Objetos (2).

El paradigma de Orientación a Objetos se desarrolló debido principalmente a los siguientes 3 factores:

- Complejidad del software.
- Enfoque jerárquico del software (programación procedimental).
- Costos de desarrollo del software.

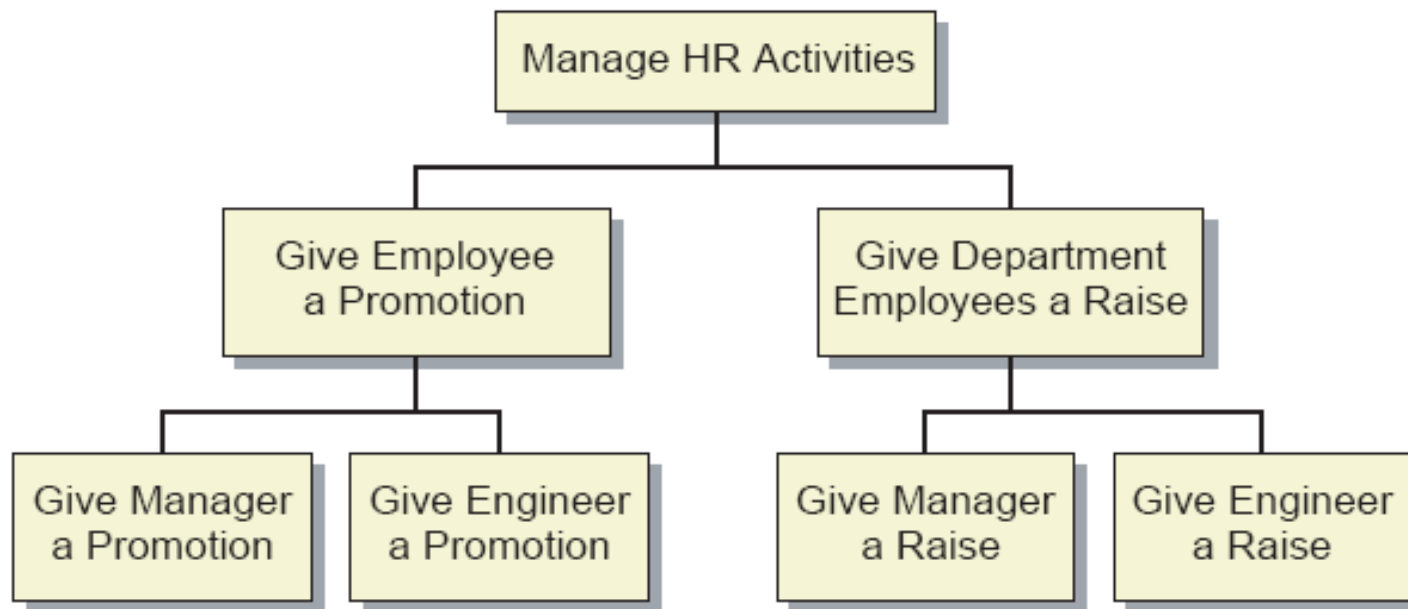
# Complejidad del Software.

Los sistemas actuales se han vuelto muy complejos y tienen las siguientes características:

- La elección de que componentes son primitivos en el sistema es cuestión de diseño.
- El sistema se puede dividir en relaciones intra- y entre-componentes.
- Este concepto de *separación de responsabilidades* (*separation of concerns*) permite estudiar cada parte del sistema en forma más o menos aislada.
- Normalmente aun los sistemas más complejos están compuestos de unos cuantos *tipos de componentes* en muchas combinaciones.
- Casi siempre, un sistema complejo exitoso evolucionó de un sistema simple que funcionó correctamente.

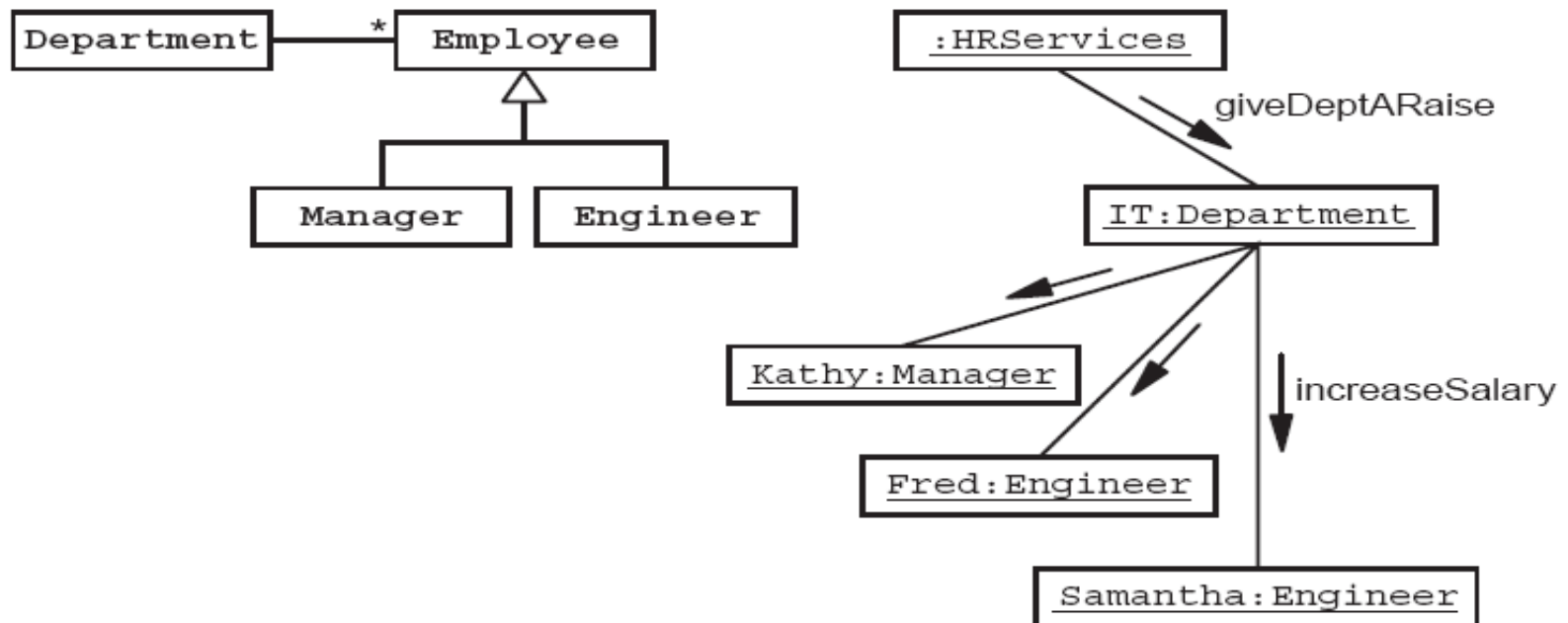
# Descomposición jerárquica del software.

En el paradigma convencional de desarrollo de Software, conocido como modelo procedimental, el software se descompone en una jerarquía de procedimientos o funciones.



## Descomposición jerárquica del software (2).

En el paradigma de Orientación a Objetos, el software es descompuesto en componentes interrelacionados e interactuantes.

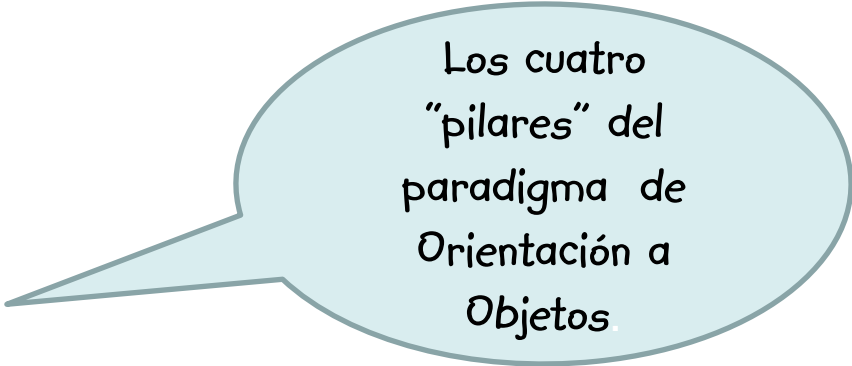


# Costos del Software.

- El paradigma de orientación a objetos disminuye los costos del software.
  - Desarrollo:
    - Los principios de OO proporcionan una técnica natural para modelar entidades de negocio y procesos desde una etapa muy temprana en el proyecto.
    - Las entidades de negocio y los procesos modelados con técnicas de OO son más fáciles de implementar en lenguajes OO.
  - Mantenimiento:
    - La facilidad de hacer cambios, la flexibilidad y la adaptabilidad del software son importantes para poder tener sistemas operando por mucho tiempo.
    - Las entidades de negocio y los procesos modelados con tecnología OO se pueden adaptar fácilmente a nuevos requerimientos funcionales

# Conceptos Fundamentales de OO.

- Objetos.
- Clases.
- ***Abstracción.***
- ***Encapsulamiento.***
- ***Herencia.***
- ***Polimorfismo.***
- Cohesión.
- Acoplamiento (*coupling*).
- Asociaciones entre objetos.



Los cuatro  
"pilares" del  
paradigma de  
Orientación a  
Objetos

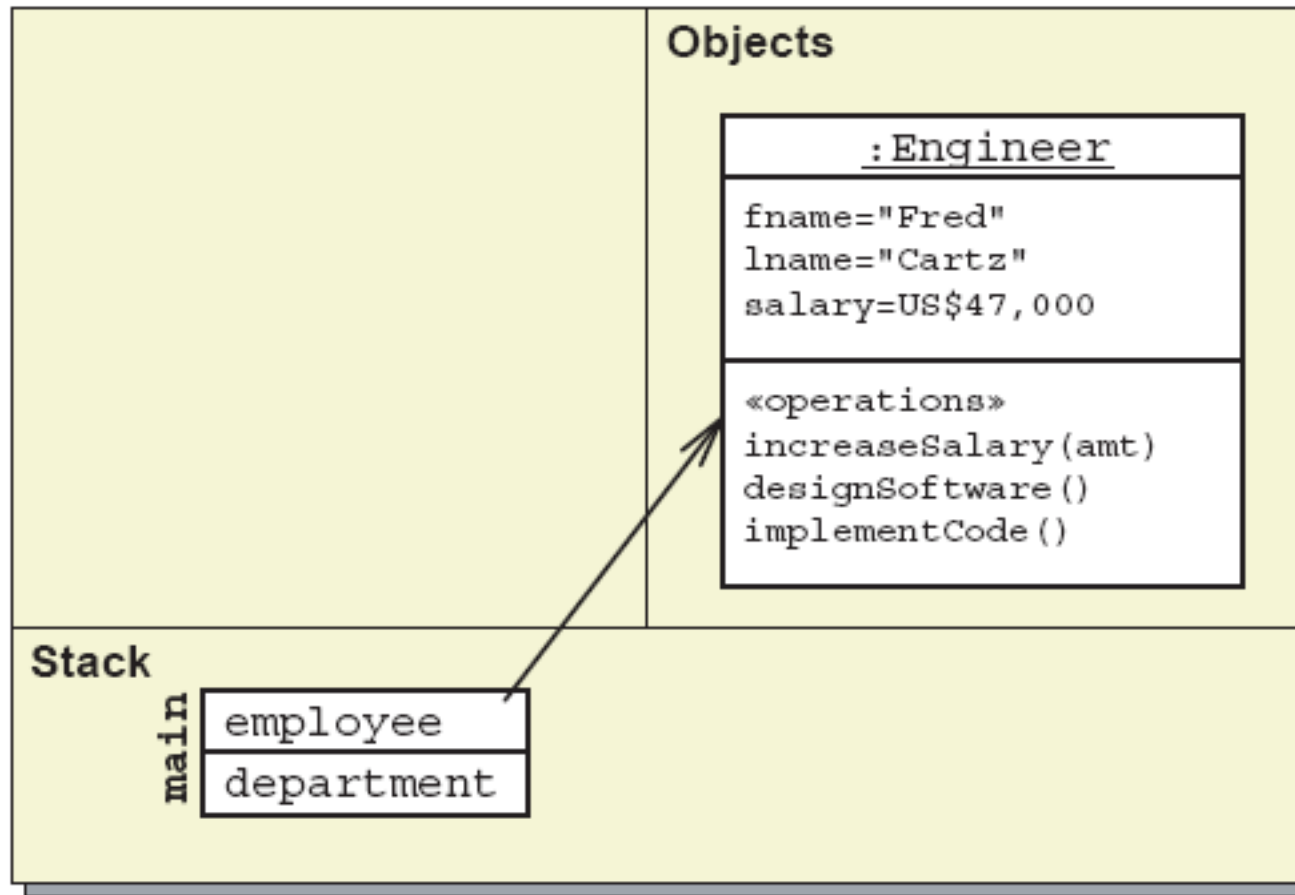


# Objetos.

Objeto = estado + comportamiento.

- Tienen identidad.
- Son ejemplares (instancias) de una sola clase.
- Tienen valores de atributos (estado) únicos.
- Tienen operaciones (métodos) comunes a la clase.

# Ejemplo de objeto.



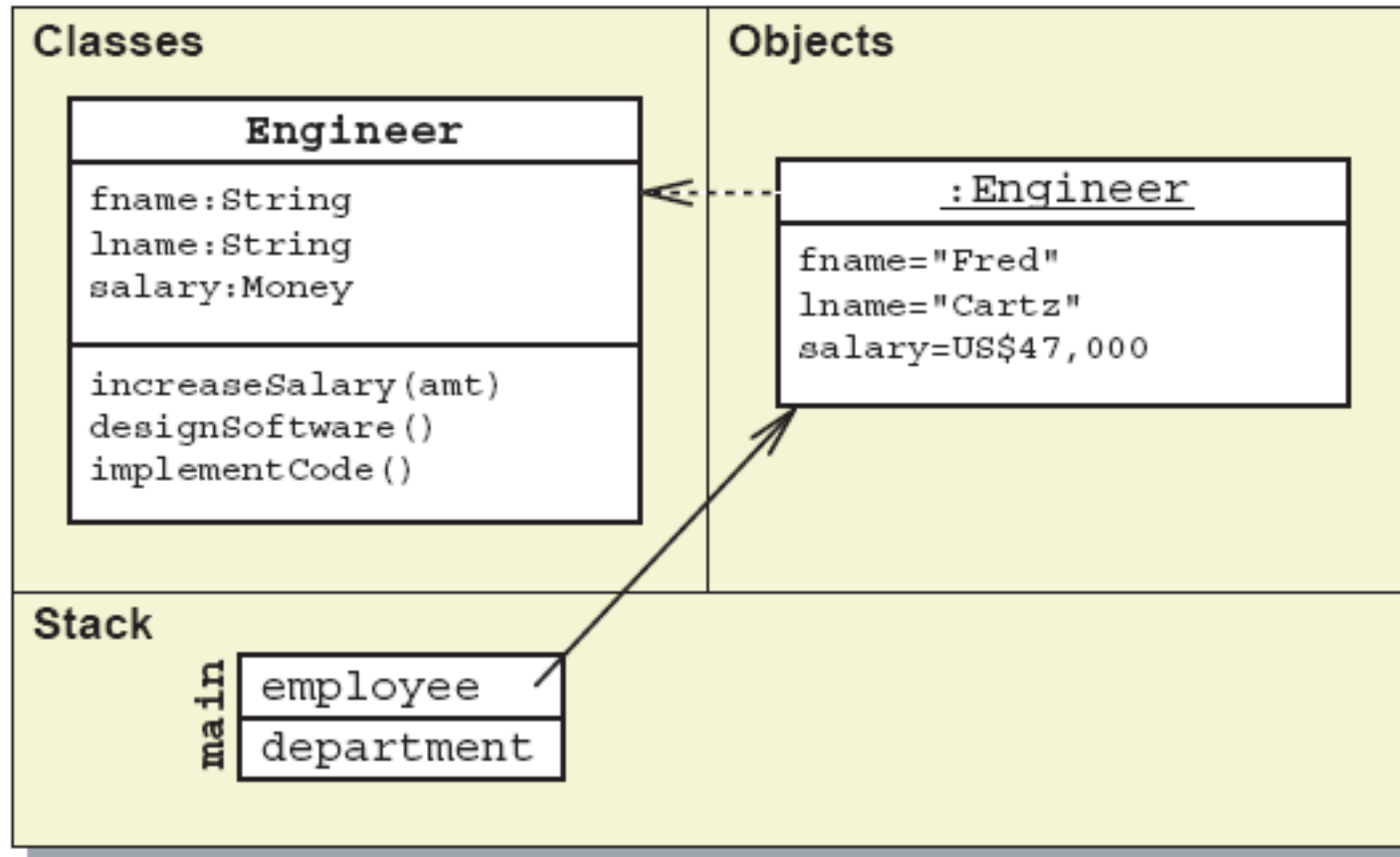
# Clases.

**Clase = plantilla de objetos que comparten una estructura y un comportamiento comunes.**

Las clases proporcionan:

- Las definiciones de los atributos (metadatos) .
- La definición de los métodos u operaciones (firma o huella).
- La implementación de los métodos (usualmente).
- Los constructores que inicializan los atributos en el momento de la creación (*instanciación*) de cada objeto de la clase.

# Ejemplo de Clase.



# Abstracción.

**Abstracción = algo que resume o concentra los puntos esenciales o más importantes de algo.**

En el software, el concepto de abstracción permite:

- crear una interfaz simplificada a un objeto.
- tomar en cuenta únicamente lo relevante del objeto
- ignorar los detalles superfluos.

# Ejemplo de mala y buena abstracción.

Engineer
fname:String lname:String salary:Money
increaseSalary(amt) designSoftware() implementCode()

✓ Buena abstracción

Engineer
fname:String lname:String salary:Money fingers:int toes:int hairColor:String politicalParty:String
increaseSalary(amt) designSoftware() implementCode() eatBreakfast() brushHair() vote()

✗ Mala abstracción

# Encapsulamiento.

**Encapsulamiento = encerrar en una cápsula.**

- El encapsulamiento es esencial a un objeto.
- Un objeto es una cápsula que mantiene el estado interno de un objeto dentro de una frontera determinada.
- En la mayoría de los lenguajes OO el encapsulamiento se implementa como "*information hiding*", que se define como **esconder los detalles de la implementación detrás de un conjunto de métodos públicos.**

# Problema de Integridad de Datos.

- La definición de la clase:

```
public class MyDate {  
    public int day;  
    public int month;  
    public int year;  
}
```

- El Problema:

```
MyDate d = new MyDate();  
d.day = 32;                // inválido  
d.month = 2; d.day = 30;   // plausible pero inválido  
d.day = d.day + 1;         // no checa por posible invalidez
```



# Clase “Encapsulada”.

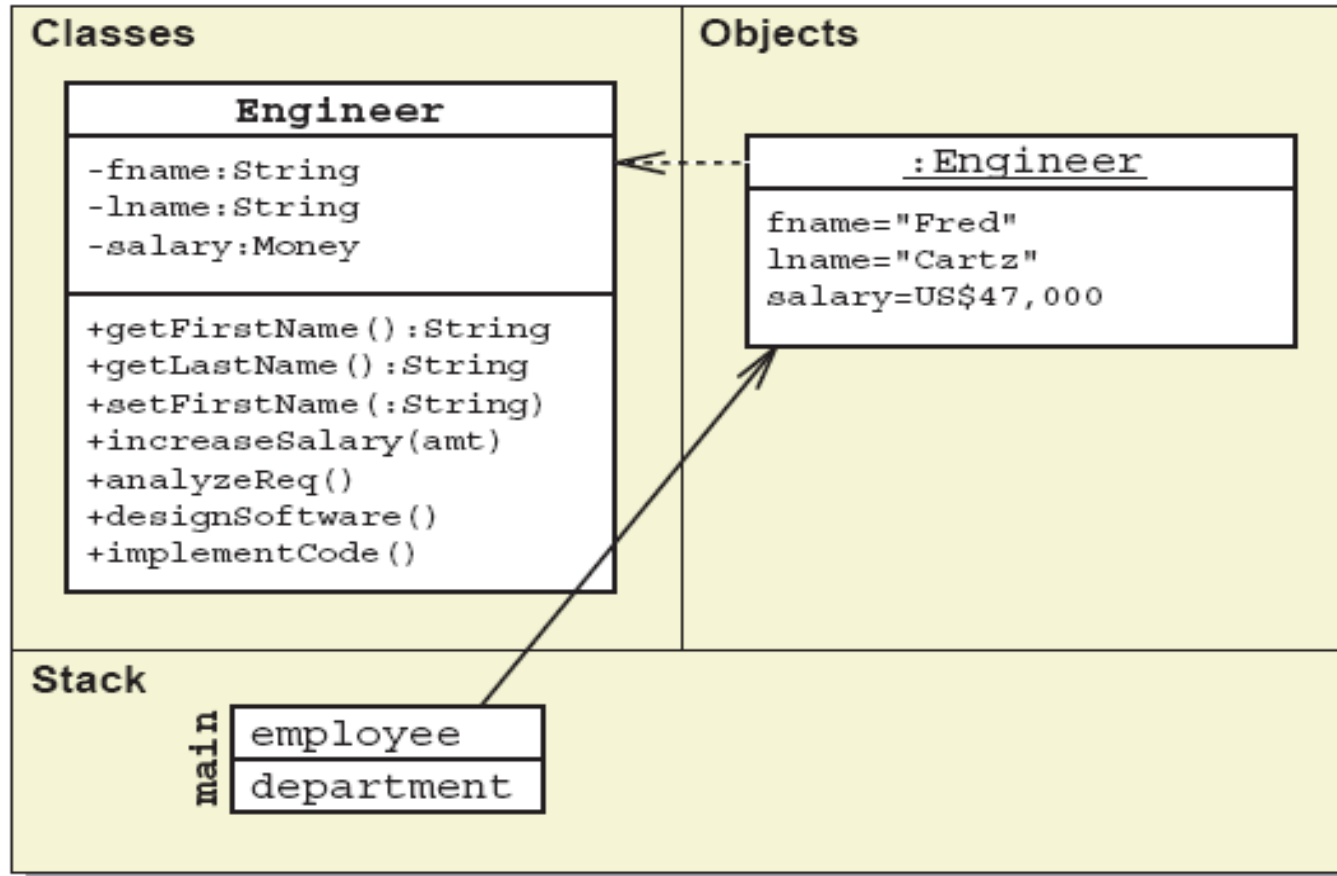
- La definición de la clase:

```
public class MyDate {  
    private int day;  
    private int month;  
    private int year;  
    public int getDay() {  
        return day,  
    }  
    public boolean setDay(int day) {  
        // valida day y regresa falso si es inválido  
    }  
    // métodos similares para month y year  
}
```

## Solución del Problema.

```
MyDate d = new MyDate();  
d.setDay(32);           // regresa false  
d.setDay(25);           // regresa true  
  
int goodDay = d.getDay() // entrega day validado
```

# Ejemplo de encapsulamiento.



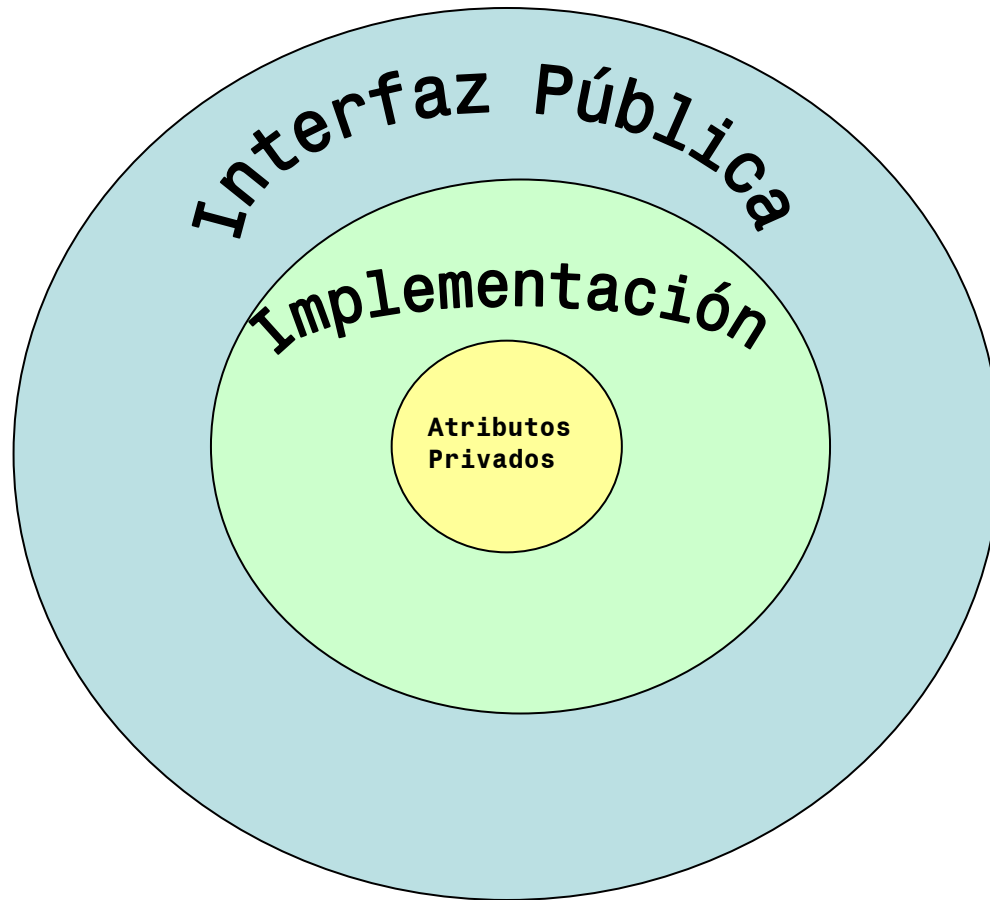
✗ `name = employee.fname;`

✗ `employee.fname = "Samantha";`

✓ `name = employee.getFirstName();`

✓ `employee.setFirstName("Samantha");`

# Encapsulamiento.



# Herencia.

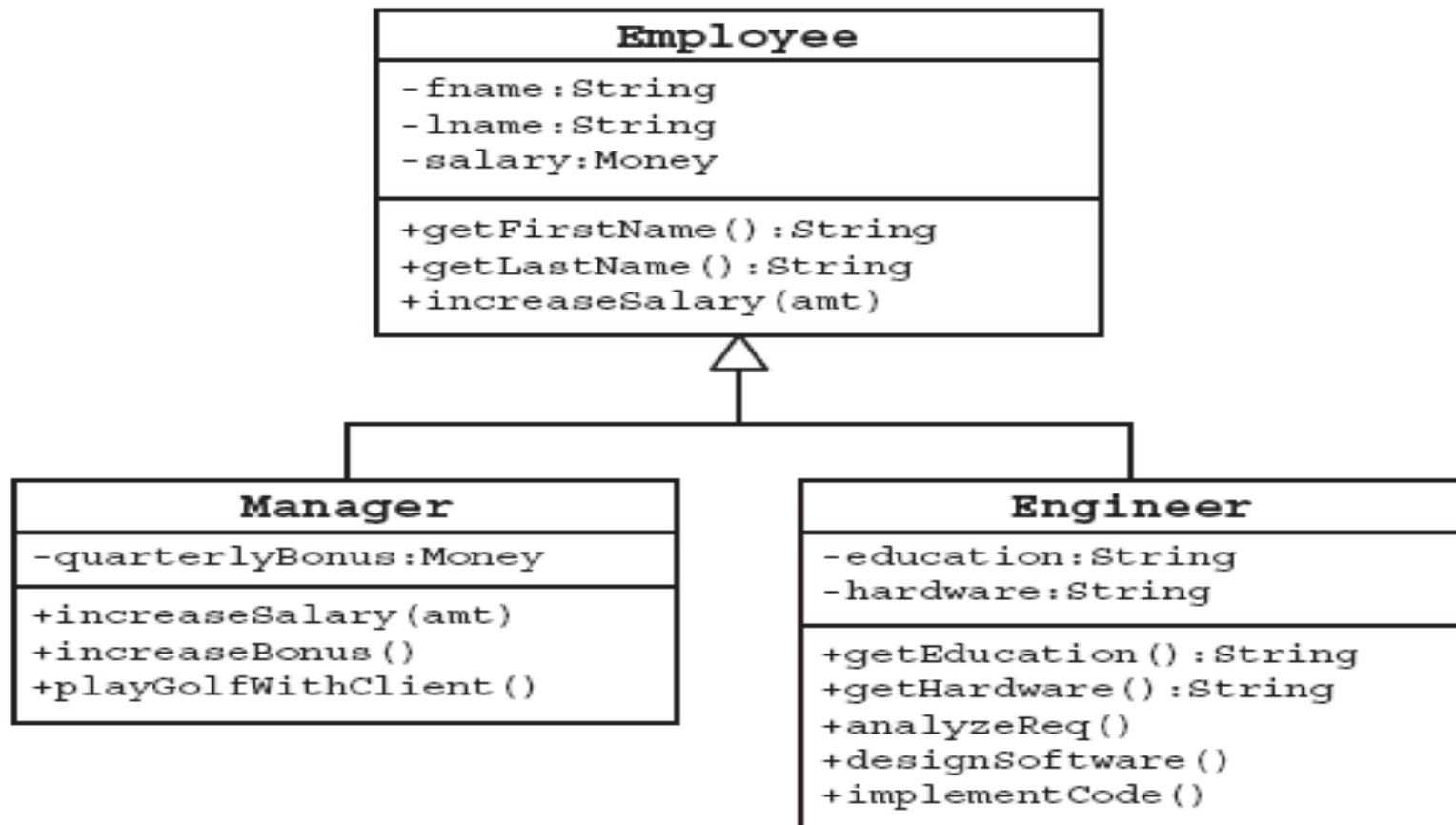
**Herencia = mecanismo que permite definir una clase en base a otra, añadiendo otras características propias.**

## Características de la herencia:

- Los atributos y operaciones de una clase (base o superclase) se incluyen en una nueva clase (derivada o subclase).
- Las operaciones de la subclase pueden substituir (*override*) las operaciones de la superclase.
- Una subclase puede heredar de varias superclases (herencia múltiple) o de una sola superclase (herencia simple).
- Algunos lenguajes de programación como C++ permiten la herencia múltiple mientras que otros como Java, sólo soportan herencia simple.

**Nota. La herencia múltiple tiene algunas desventajas en el diseño de lenguajes.**

# Ejemplo de herencia.



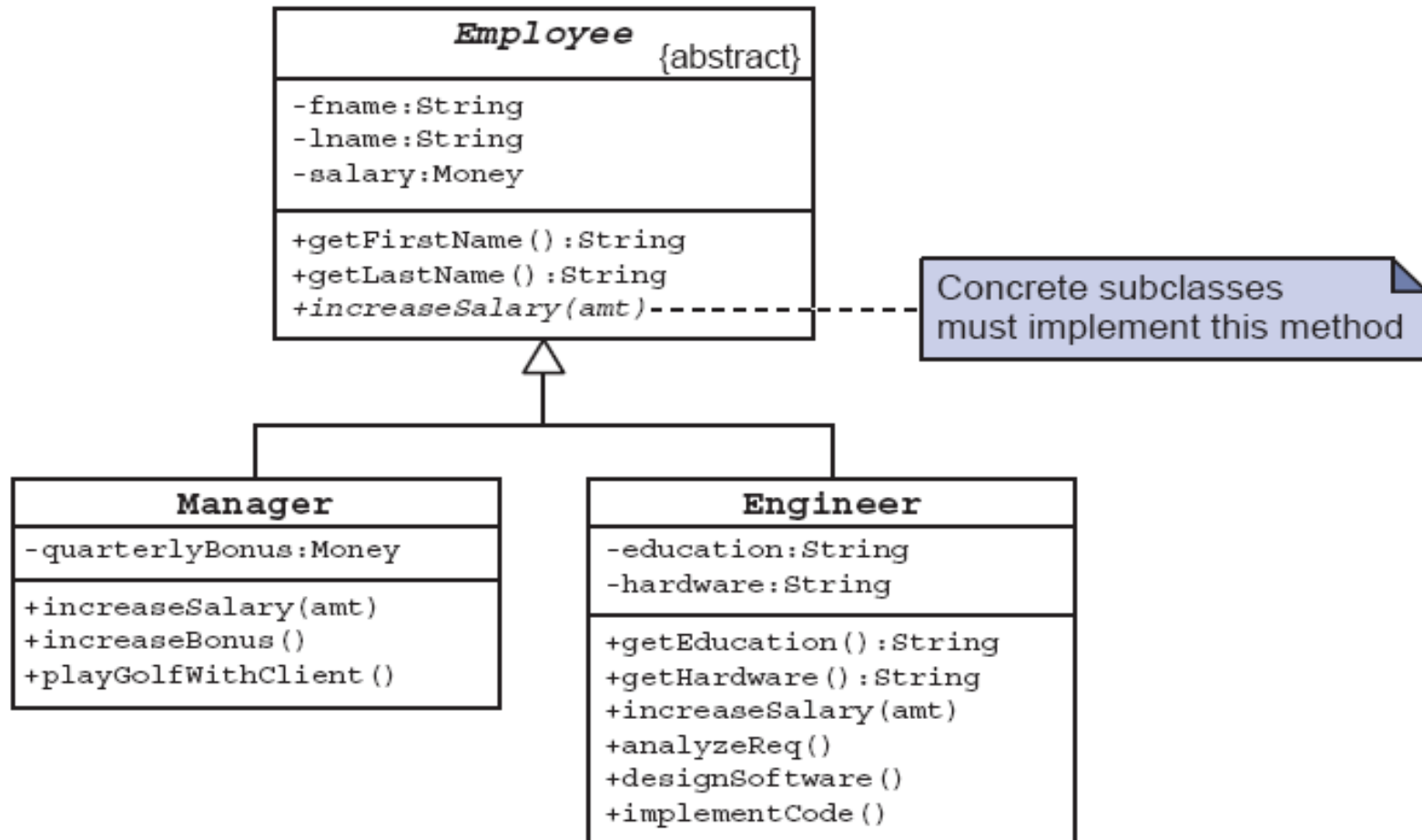
# Clases Abstractas.

**Clase Abstracta = clase de la cual no se pueden crear (instanciar) objetos.**

## Características de las clases abstractas:

- Pueden tener atributos.
- Pueden tener métodos, algunos de los cuales pueden ser declarados abstractos, que son métodos que no tienen implementación y por tanto deben ser implementados en una subclase no abstracta.
- Pueden tener constructores, que son ejecutados cuando se crea un objeto de una subclase no abstracta de la clase abstracta.
- Las subclases de una clase abstracta deben proporcionar las implementaciones de todos los métodos abstractos (a menos que la subclase también se declare como abstracta).

# Ejemplo de clase abstracta.





# Interfaces.

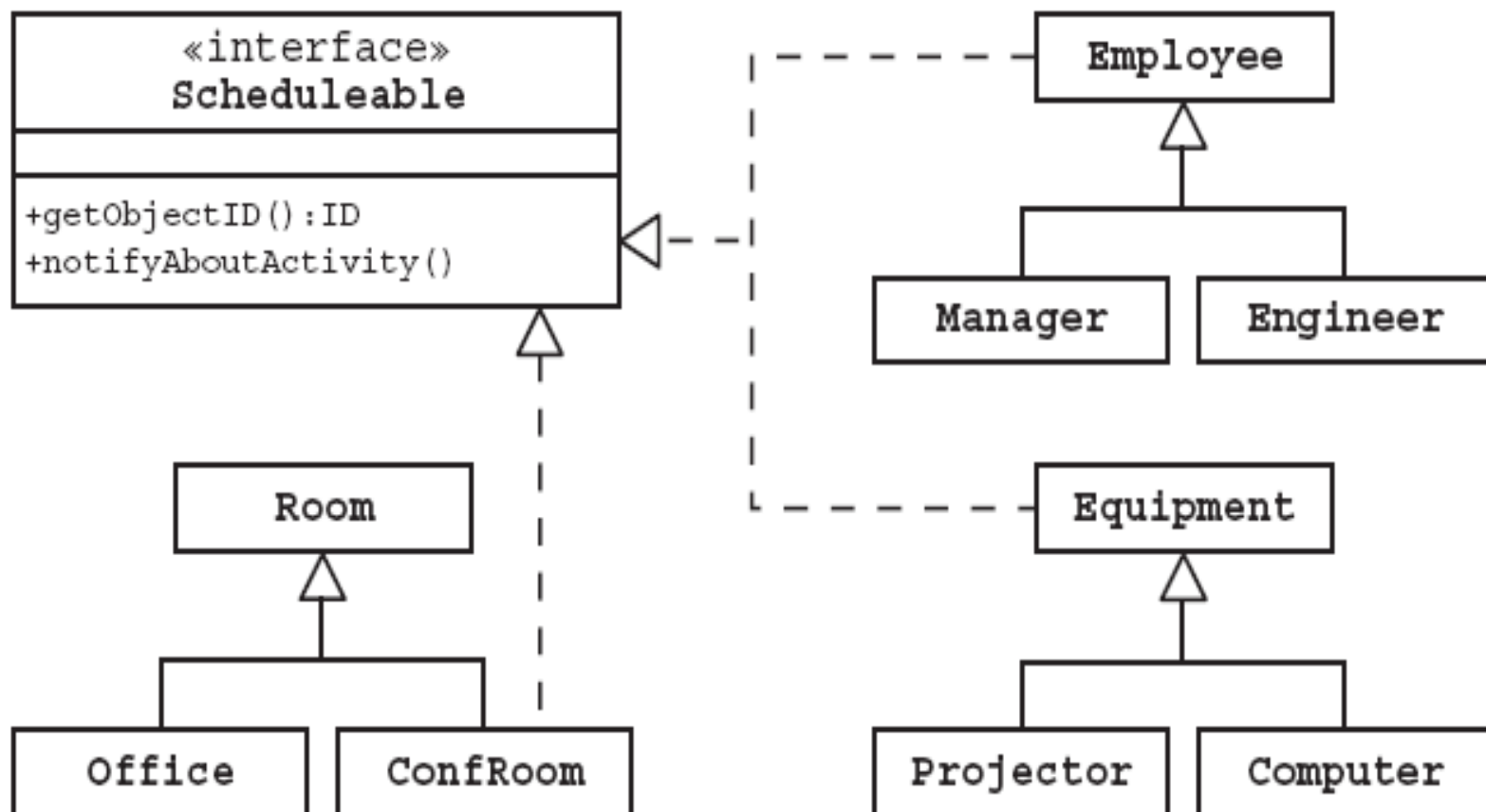
**Interfaz = conjunto de operaciones agrupadas bajo un nombre, que caracterizan el comportamiento de un elemento.**

Características de las interfaces:

- No pueden tener atributos (excepto constantes).
- Pueden tener solamente métodos abstractos.
- No pueden tener constructores.
- Se pueden definir subinterfaces, para formar una jerarquía de interfaces.
- Una clase puede implementar una o más interfaces.

*“Las interfaces son clases abstractas llevadas al extremo”*

# Ejemplo de interfaz.



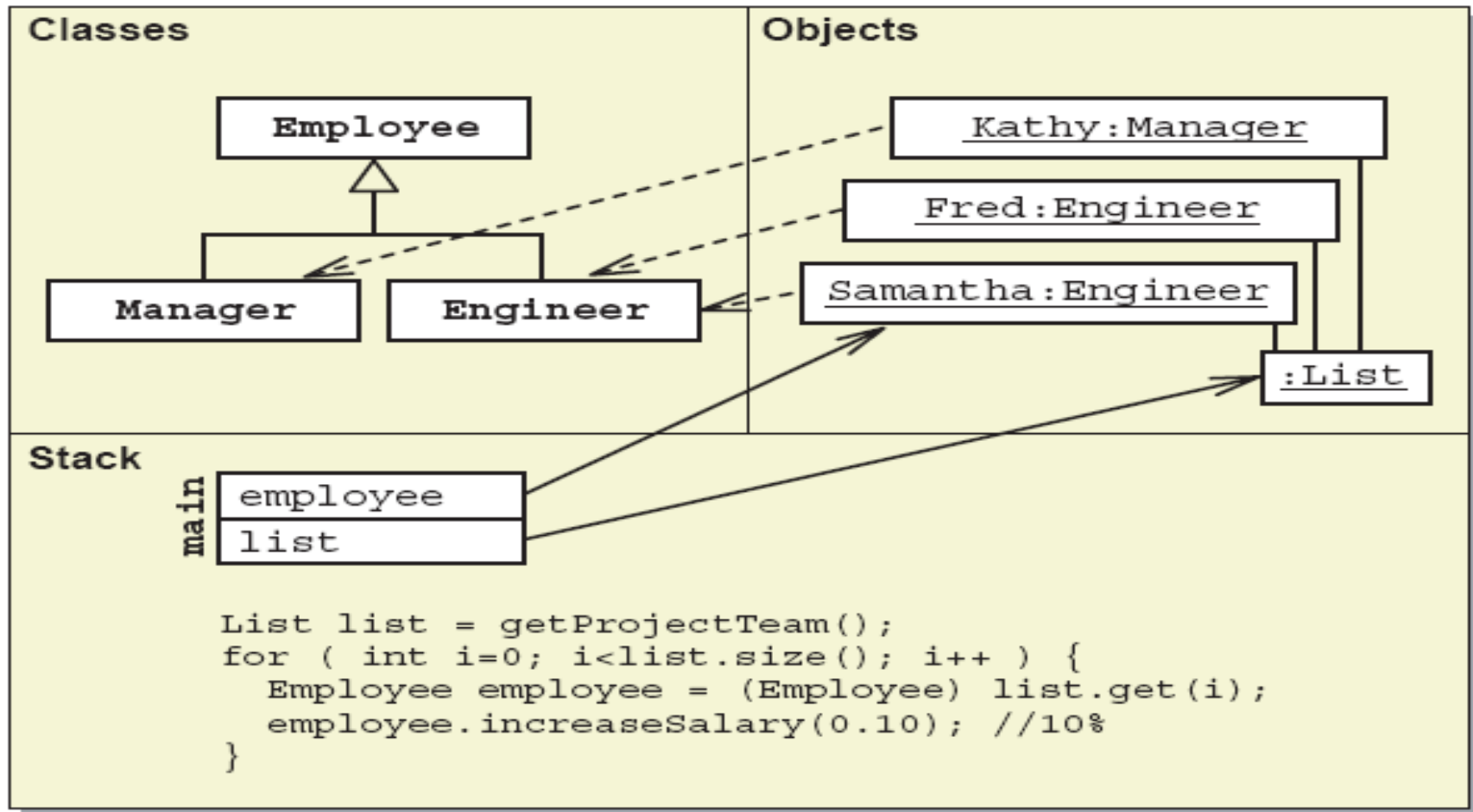
# Polimorfismo.

**Polimorfismo = un nombre (como una variable) puede denotar objetos de varias clases diferentes, pero relacionadas por una superclase común.**

## Aspectos del polimorfismo:

- Se pueden asignar diferentes tipos de objetos a una variable en el momento de la ejecución de un programa.
- La implementación de un método se determina por el tipo de objeto y no por el tipo de declaración.
- El polimorfismo solamente aplica a clases relacionadas por herencia.

# Ejemplo de polimorfismo.



# Cohesión.

**Cohesión = medida de que tanto una entidad (componente o clase) soporta un propósito particular de un sistema.**

- Hay baja cohesión cuando un componente realiza muchas funciones poco relacionadas entre sí.
- Hay alta cohesión cuando un componente lleva a cabo sólo un conjunto pequeño de funciones.
- Clases pequeñas con pocas, pero altamente relacionadas, funciones son más fáciles de mantener.

# Ejemplo de alta y baja cohesión.

## Low Cohesion

<b>SystemServices</b>
makeEmployee makeDepartment login logout deleteEmployee deleteDepartment retrieveEmpByName retrieveDeptByID

## High Cohesion

<b>LoginService</b>
login logout

<b>EmployeeService</b>
makeEmployee deleteEmployee retrieveEmpByName

<b>DepartmentService</b>
makeDepartment deleteDepartment retrieveDeptByID

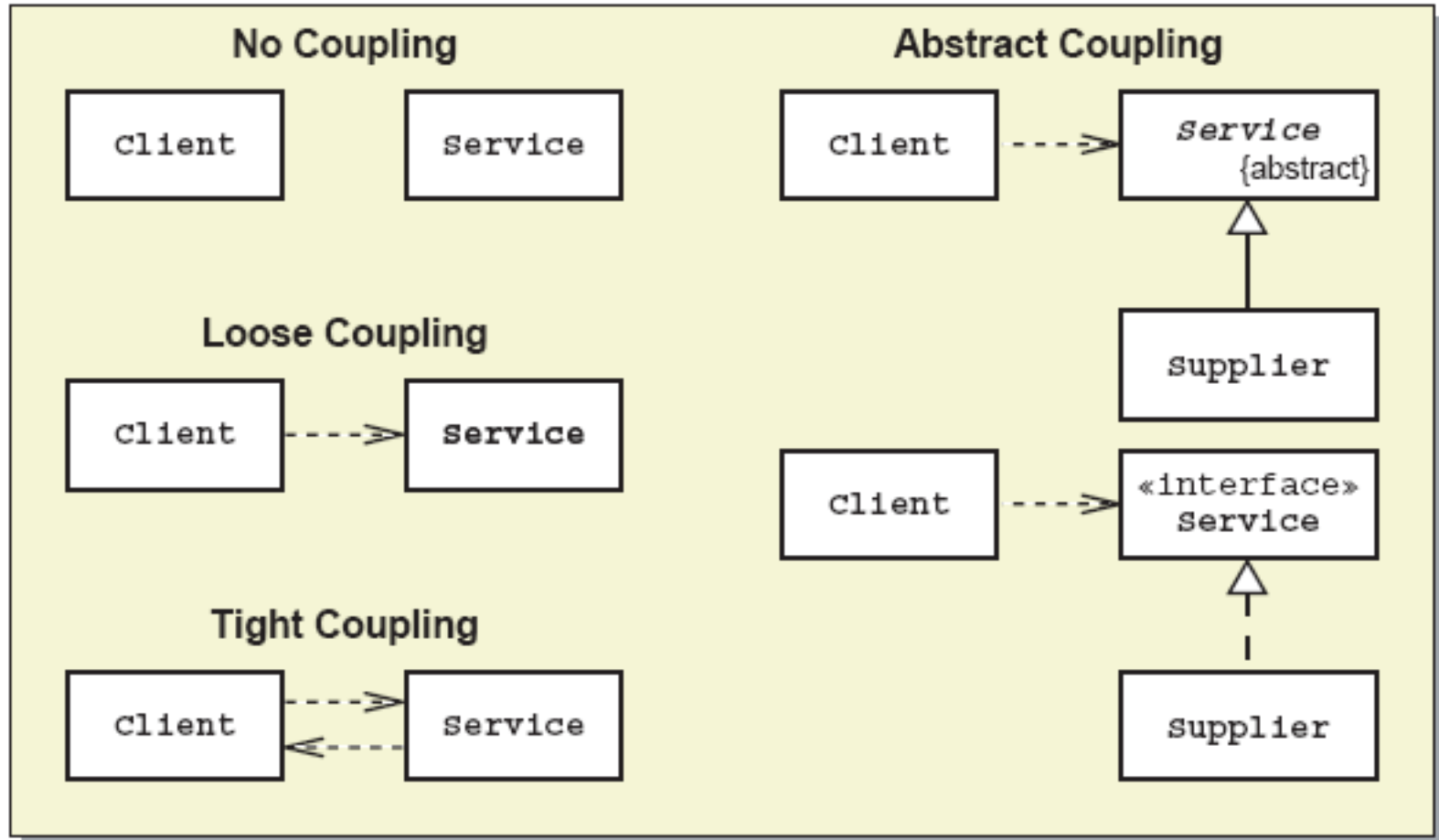
# Acoplamiento.

**Acoplamiento = grado de dependencia entre las clases definidas en un sistema.**

## Características del acoplamiento:

- Se refiere a que tanto una clase hace uso de otra.
- Hay acoplamiento suave (*loose*) y acoplamiento fuerte (*tight*).
- Cuando dos clases no están relacionadas se dice que tienen acoplamiento nulo.
- También existe la idea de acoplamiento abstracto, que es cuando una clase cliente hace uso de un servicio mediante una clase abstracta (o interfaz), pero no sabe que clase concreta proporciona el servicio.

# Ejemplos de acoplamiento.





# Asociaciones entre objetos.

**Asociación = relación que denota una conexión semántica entre dos clases.**

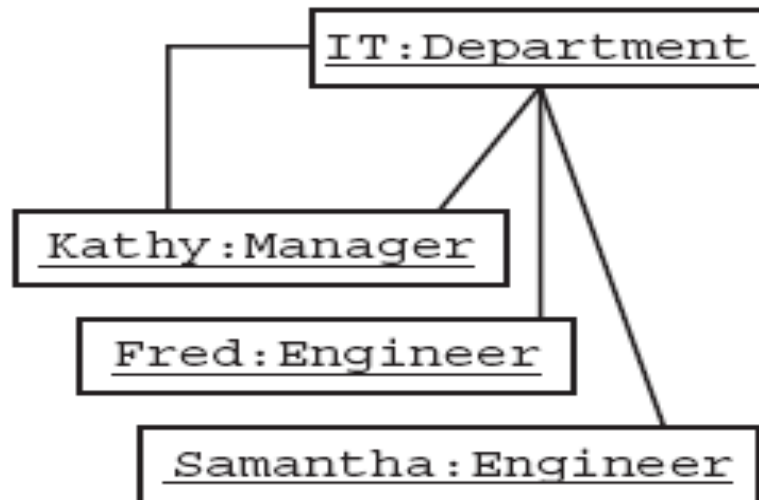
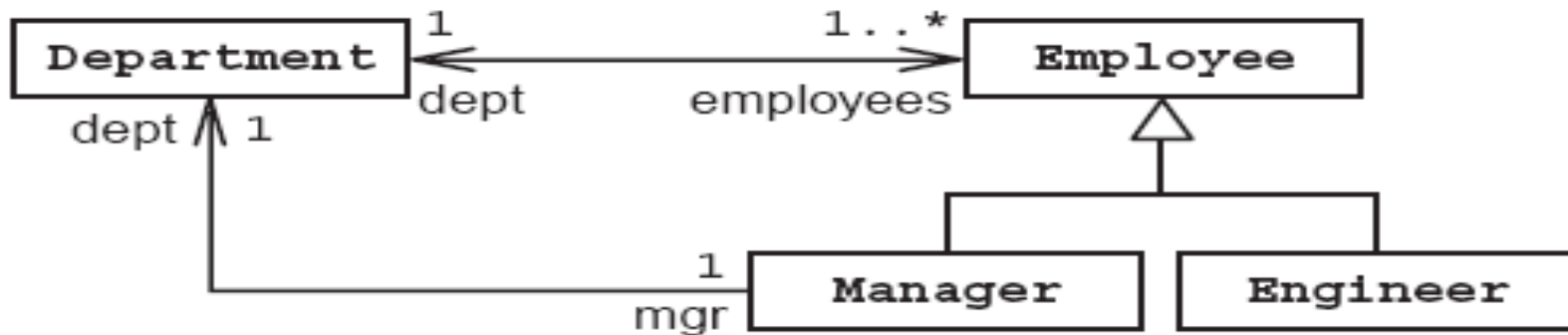
Características de las asociaciones:

- Los roles que juega cada clase.
- La multiplicidad de cada rol.
- La dirección (navegabilidad de la asociación).

Tipos de Asociaciones:

- Asociación simple
- Agregación
- Composición

# Ejemplos de asociaciones entre objetos.



## Ejercicio 1: Características del paradigma procedimental.

**Marque los postulados verdaderos acerca del modelo procedimental.**

✓	Descripción.
✓	Los programas procedimentales son frágiles porque un cambio a menudo afecta partes diversas del programa.
✓	En el modelo procedimental, un sistema complejo se descompone en una jerarquía de subrutinas.
✗	Los programas procedimentales corresponden fuertemente con los modelos mentales del sistema.
✗	Los programas procedimentales son más baratos de mantener que los programas orientados a objetos.
✗	Los programas procedimentales son más fáciles de diseñar que los programas orientados a objetos.

## Ejercicio 2: Características del paradigma OO.

**Marque los postulados verdaderos acerca del modelo OO.**

✓	Descripción.
✓	Los programas orientados a objetos son flexibles.
✗	El modelo orientado a objetos es difícil de diseñar porque la tecnología OO no es cercana al modelo mental humano.
✓	Los programas orientados a objetos son más baratos de mantener que los programas procedimentales.
✓	En la tecnología OO un sistema complejo se descompone en una jerarquía de objetos que colaboran entre sí.
✗	Usualmente, los programas orientados a objetos son frágiles.

## Ejercicio 3: Principios de OO.

Haga corresponder los términos con sus definiciones.

	<b>Término</b>		<b>Definición</b>
a	Polimorfismo. <b>6</b>	1	Se implementa como " <i>information hiding</i> ", es decir, se esconden los detalles de la implementación detrás de un conjunto de métodos públicos.
b	Objeto. <b>7</b>	2	Habilidad de derivar nuevas clases de clases base existentes. Las nuevas clases adquieren los atributos y operaciones de las clases base.
c	Herencia. <b>2</b>	3	La plantilla o definición de objetos.
d	Encapsulamiento <b>1</b>	4	Mecanismo consistente en ignorar los detalles para identificar aspectos relevantes y proporcionar una interfaz simple.
e	Clase. <b>3</b>	5	Medida de que tanto una entidad soporta un propósito particular dentro de un sistema.
f	Abstracción. <b>4</b>	6	Mecanismo que permite a variables, en el momento de la ejecución de un programa hacer referencia a objetos de diferentes clases.
g	Asociación. <b>8</b>	7	Instancia o ejemplar de una clase.
h	Acoplamiento. <b>9</b>	8	Relación entre dos clases y subsecuentemente entre dos objetos de esas dos clases.
i	Cohesión. <b>5</b>	9	Grado en el que las clases de un sistema dependen unas de otras.