Modelado de aplicaciones

INFO175

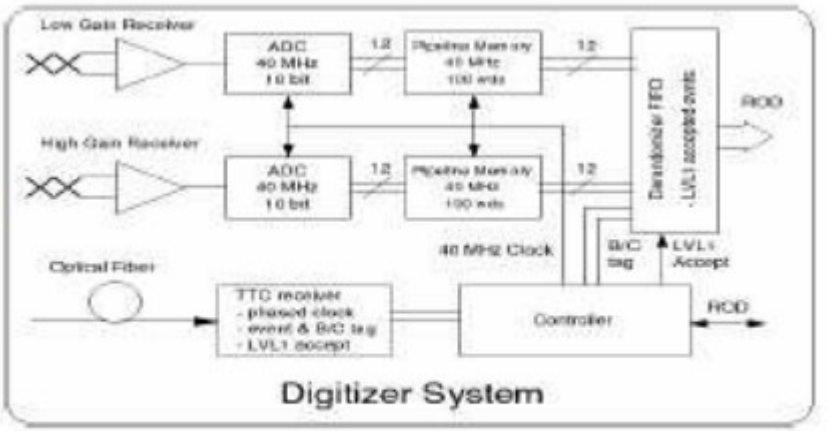
¿Por qué modelar antes de desarrollar?

- Idea clara de lo que se hará.
- Transmitir requerimientos de forma precisa al equipo del proyecto.
- Identificar aspectos relevantes de la problemática antes de comenzar a desarrollar.
- Adoptar estrategias de desarrollo que mejor se ajusten a la problemática.
- Investigar y analizar el mejor conjunto de tecnologías para abordar la problemática.

Modelado

- Un modelo es una abstracción de un sistema o entidad del mundo real.
- Una abstracción es una simplificación, que incluye sólo aquellos detalles relevantes para algún determinado propósito.
- El modelado permite abordar la complejidad de los sistemas

Modeled system



Functional model

Modelado de software

```
package codemodel;
public class Guitarist extends Person implements MusicPlayer {
    Guitar favoriteGuitar;
     public Guitarist (String name) {super(name);}
            // A couple of local methods for accessing the class's properties
    public void setInstrument(Instrument instrument) {
            if (instrument instanceof Guitar) {
                        this.favoriteGuitar = (Guitar) instrument;
            } else {
                        System.out.println("I'm not playing that thing!");
             }
    }
    public Instrument getInstrument( ) {return this.favoriteGuitar;}
}
```

- Representa sólo la lógica e ignora el resto.
- El ser humano lo interpreta lentamente.
- No facilita la reutilización ni la comunicación.

Modelado de software

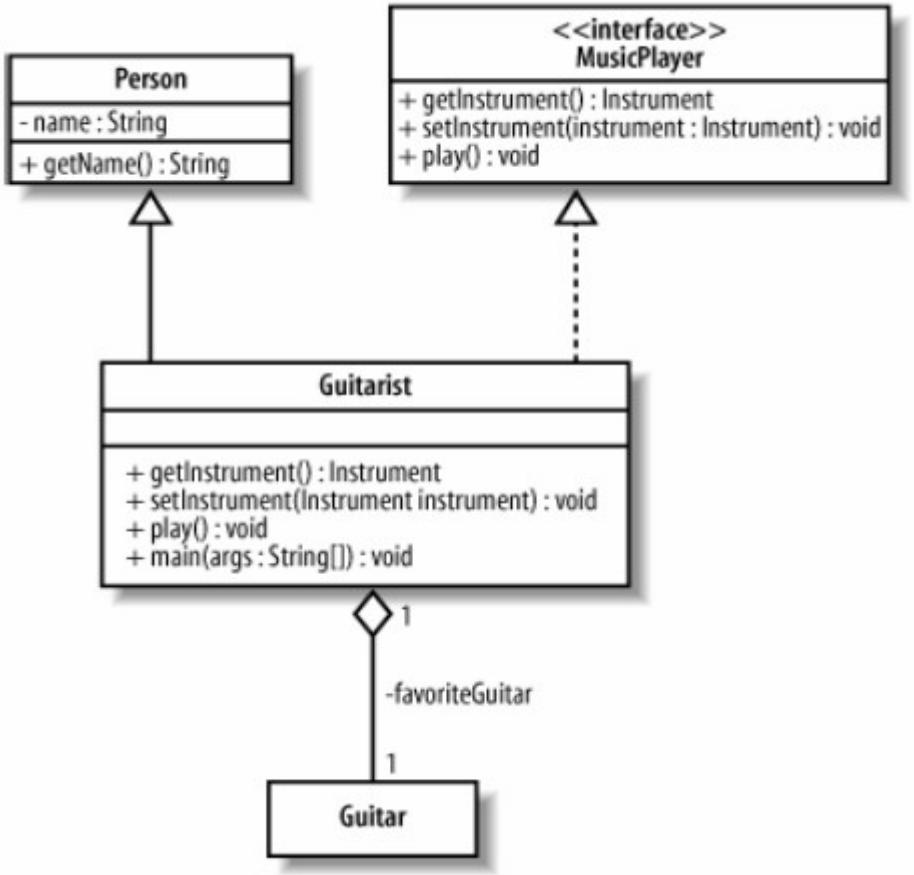
Guitarist is a class that contains six members: one static and five non-static. Guitarist uses, and so needs an instance of, Guitar; however, since this might be shared with other classes in its package, the Guitar instance variable, called favoriteGuitar, is declared as default.

Five of the members within Guitarist are methods. Four are not static. One of these methods is a constructor that takes one argument, and instances of String are called name, which removes the default constructor.

Three regular methods are then provided. The first is called setInstrument, and it takes one parameter, an instance of Instrument called instrument, and has no return type. The second is called getInstrument and it has no parameters, but its return type is Instrument. The final method is called play. The play method is actually enforced by the MusicPlayer interface that the Guitarist class implements. The play method takes no parameters, and its return type is void.

- Es ambigua y confusa.
- Es lenta de interpretar.
- Difícil de procesar.

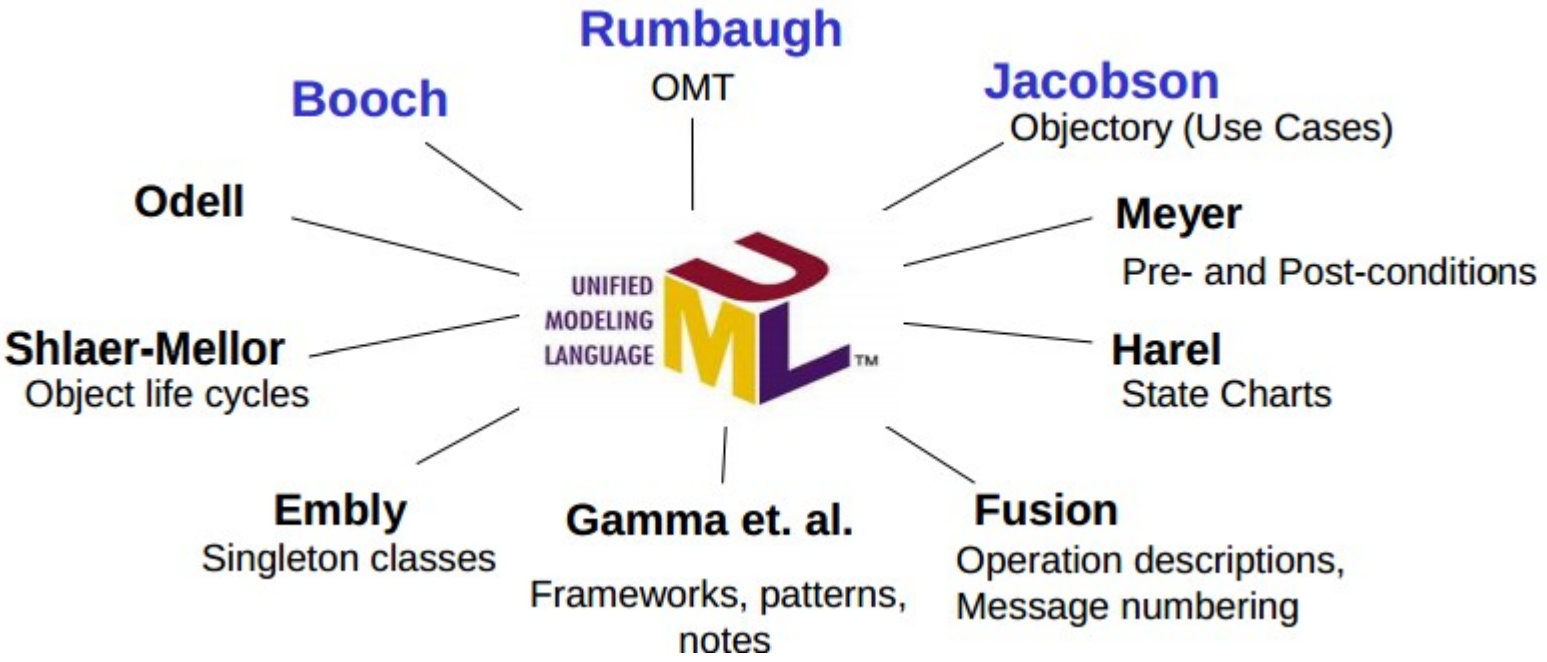
Modelado de software



- No es ambigua ni confusa (Si se conoce la semántica de cada elemento de modelado.
- Es fácil y rápida de interpretar.
- Es fácil de procesar por herramientas.

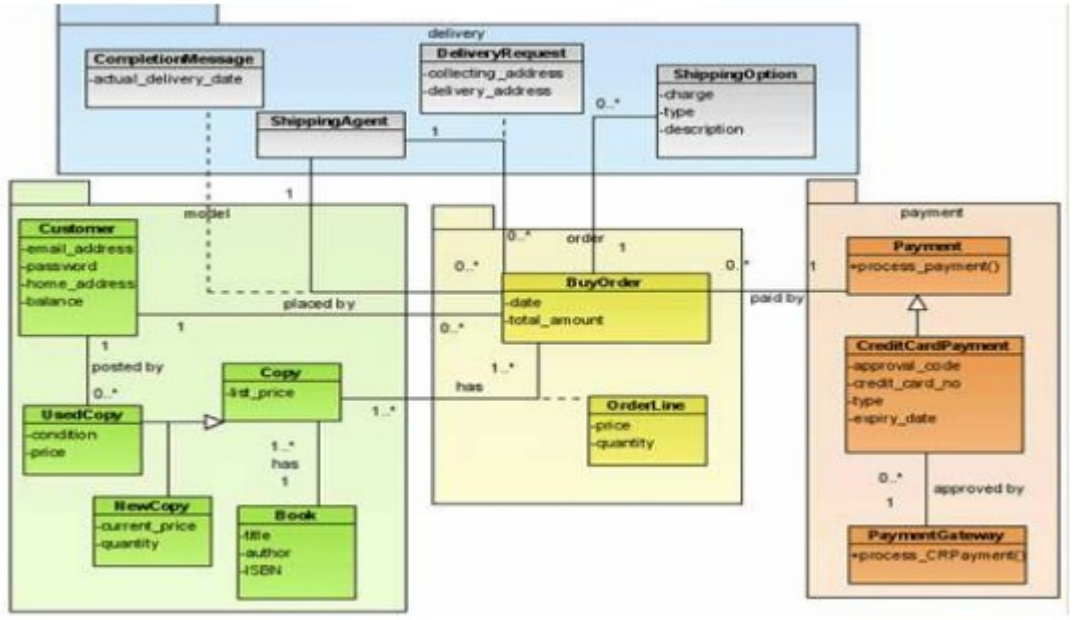
UML

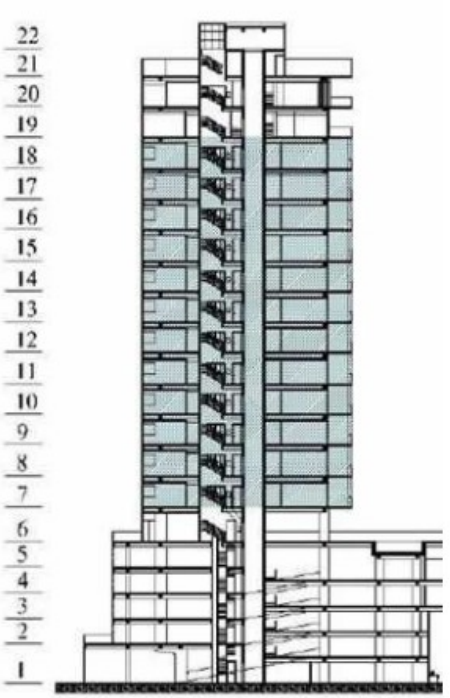
- **U**nified **M**odeling **L**anguage.
- Es un lenguaje de modelado visual de propósito general orientado a objetos.
  - Impulsado por el Object Management Group www.omg.org
- Estándar: Independiente de cualquier fabricante comercial.
- Agrupa notaciones y conceptos provenientes de distintos tipos de métodos orientados a objetos.



Características UML

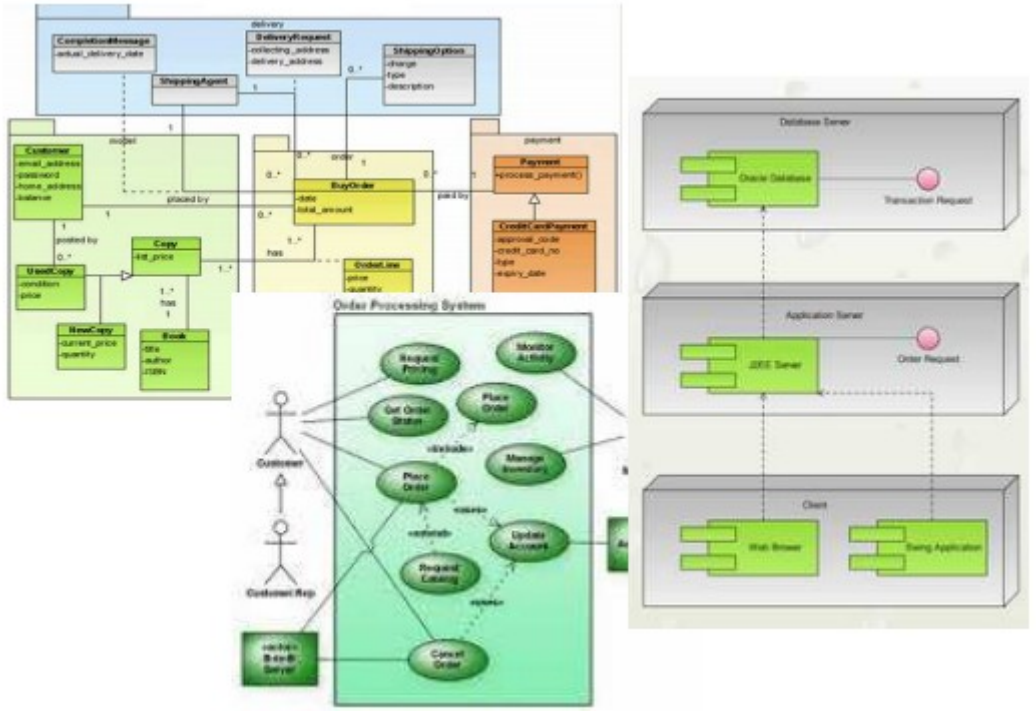
- “Lenguaje cuyo vocabulario y reglas se centran en la representación conceptual y física de un sistema” (Booch, Jacobson y Rumbaugh)
- Lenguaje = Notación + Reglas (Sintácticas, semánticas)
- Vocabulario y reglas.
  - Para crear y leer modelos bien formados.
  - Que constituyen los planos de un sistema de software.

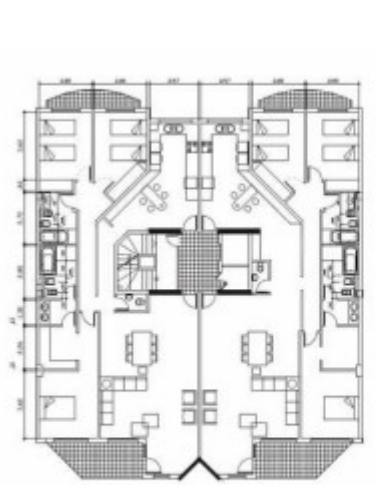


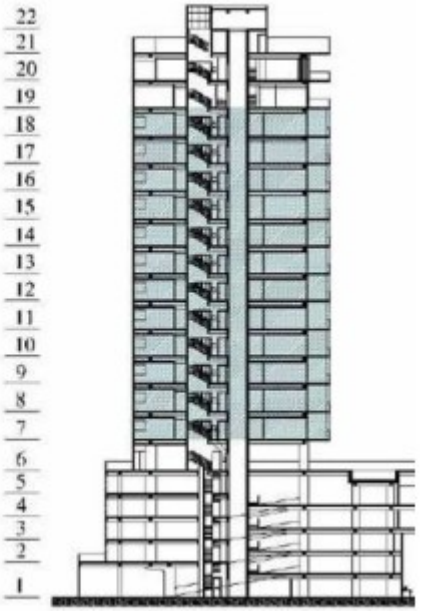


Características UML

- UML es independiente del proceso de desarrollo.
- UML **NO** es una metodología.
- UML cubre las diferentes vistas de la arquitectura de un sistema mientras evoluciona a través del ciclo de vida del desarrollo de software.





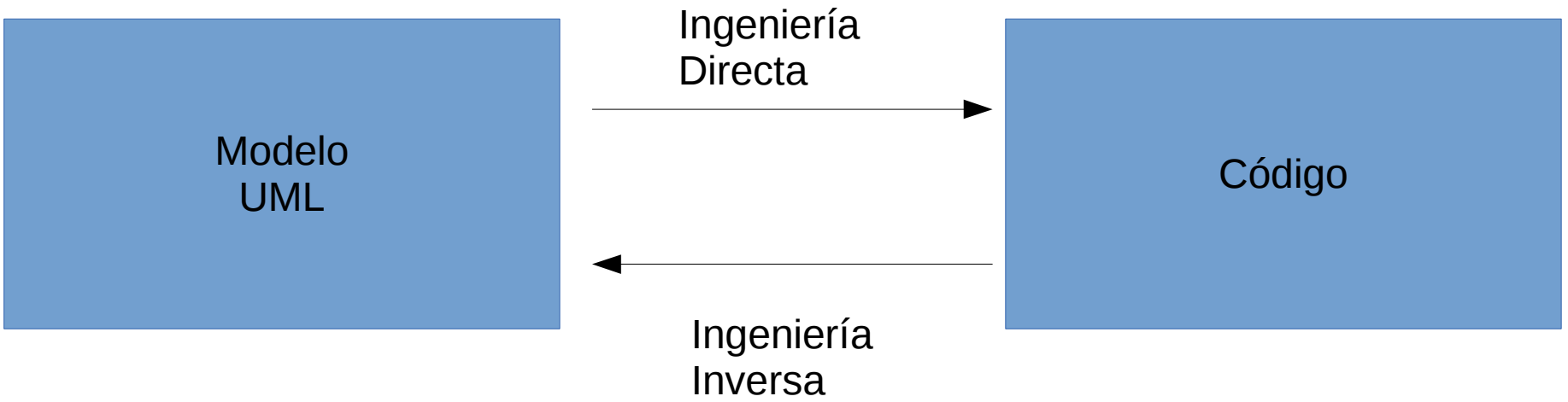


Ventajas de UML

- Es estándar → Facilita la comunicación.
- Se basa en una notación gráfica concisa y fácil de aprender y utilizar (lo básico).
- Se puede utilizar para modelar sistemas de software en diversos dominios.
  - Sist. de información empresariales, WEB, tiempo real, etc.
  - Incluso en sistemas que no son de software.
- Es fácilmente extensible.
- Fuerza al diseñador a profundizar en el problema.
  - Perimte adelantarse a escenarios complejos.

Ventajas UML

- UML no es un lenguaje de programación visual, pero es posible establecer correspondencias entre un modelo UML y lenguajes de programación (Java, C++, etc) y bases de datos (Relacionales, OO)



Desventajas de UML

- No cubre todas las necesidades de especificación de un proyecto de software.
  - Ej. Diseño de interfaces de usuario.
- Puede resultar complejo alcanzar un conocimiento completo del lenguaje.
- Útil sólo para el paradigma orientado a objetos.
- Toma mucho tiempo mantener los diagramas actualizados con el código fuente a medida que éste progresa.
- Hay aspectos de diseño que se modelan mejor cuando se está codificando.
- Algunas cosas de UML no pueden ser siempre implementadas en el lenguaje de programación escogido. Así mismo, ciertas características de los lenguajes de programación no pueden ser modeladas con UML.
- Es difícil anticipar todos los problemas de implementación que se tendrán. Esto puede producir grandes diferencias entre lo modelado e implementado.

Diagramas UML

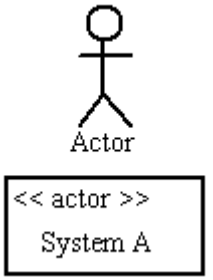
Estructurales o estáticos	Comportamiento o dinámicos
Clases	Casos de uso
Objetos	Estado
Componentes	Actividades
Despliegue	Secuencia
Paquetes	Comunicación
Estructura compuesta	

Casos de uso

- “Los casos de uso describen bajo la forma de acciones y reacciones el comportamiento de un sistema desde la perspectiva del usuario”. Ivar Jacobson.
- Es una técnica para capturar información respecto de los servicios que un sistema proporciona a su entorno (especificación de requisitos)
- Los elementos que pueden aparecer son: actores, casos de uso y relaciones entre casos de usos.

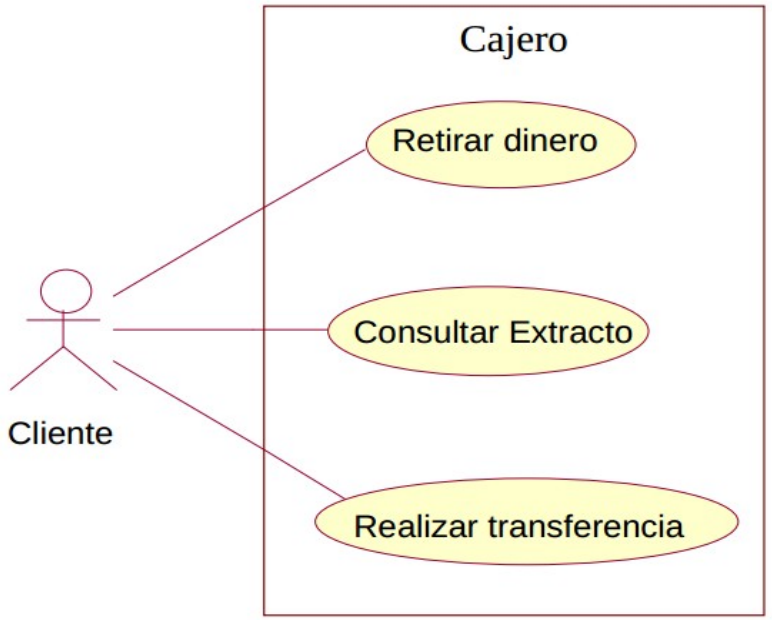
Casos de uso: Actores

- Un actor es una entidad externa al sistema que realiza algún tipo de interacción con el mismo.
- Pueden ser personas, otros sistemas, sensores, etc.
- Los actores se distinguen por el rol que cumplen en el sistema.

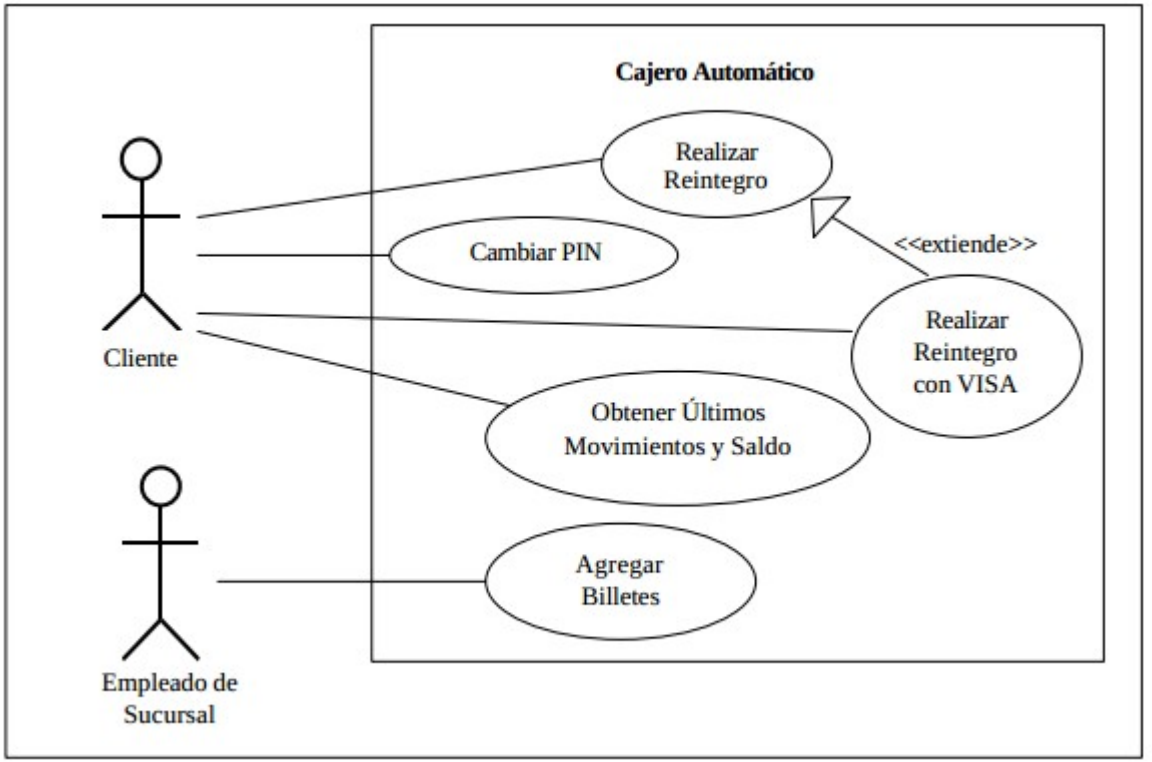


Caso de uso

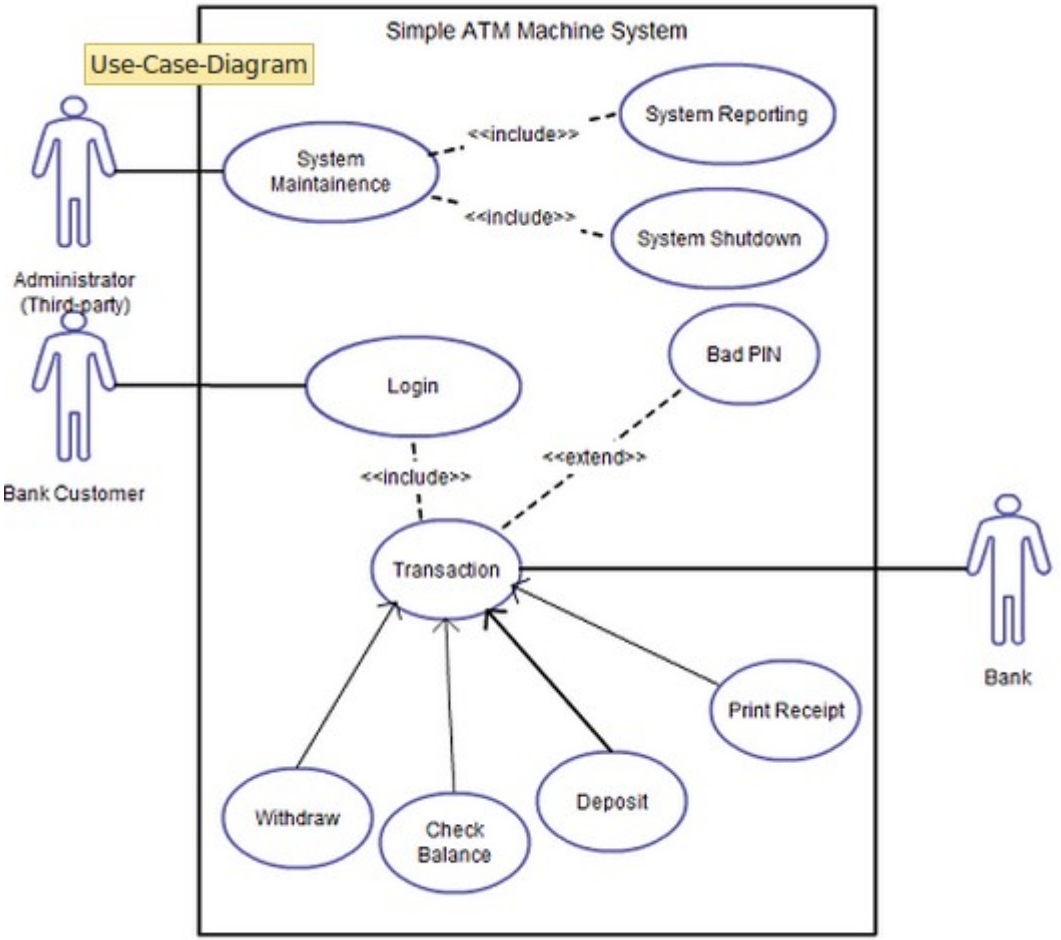
- Descripción de la secuencia de interacciones que se producen entre un actor y el sistema.
- Expresa una unidad coherente de funcionalidad.
  - Ej: Login, retirar dinero, comprar producto, etc
- Se resprenta mediante una elipse con el nombre del caso de uso en su interior.



Ejemplo caso de uso



Ejemplo caso de uso



Use Case diagram showing Actors and main processes

Diagrama de clases

- Muestran un conjunto de clases, interfaces y colaboraciones, así como las relaciones entre ellos.
- Son la piedra angular y diagramas más comunes en el análisis y diseño de un sistema.
- Describen el diseño detallado de un software OO.
- Abarcan la vista de diseño estática de un sistema.

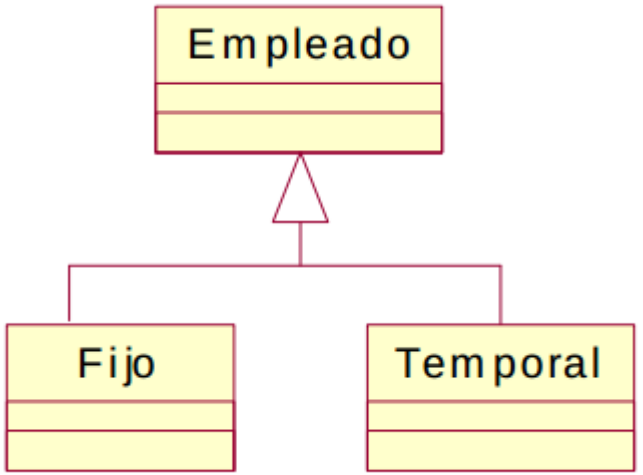


Diagrama de clases

- Principalmente un diagrama de clases contiene:
  - Clases (Con atributos, operaciones y visibilidad)
  - Relaciones: dependencia, generalización, asociación, agregación y composición.

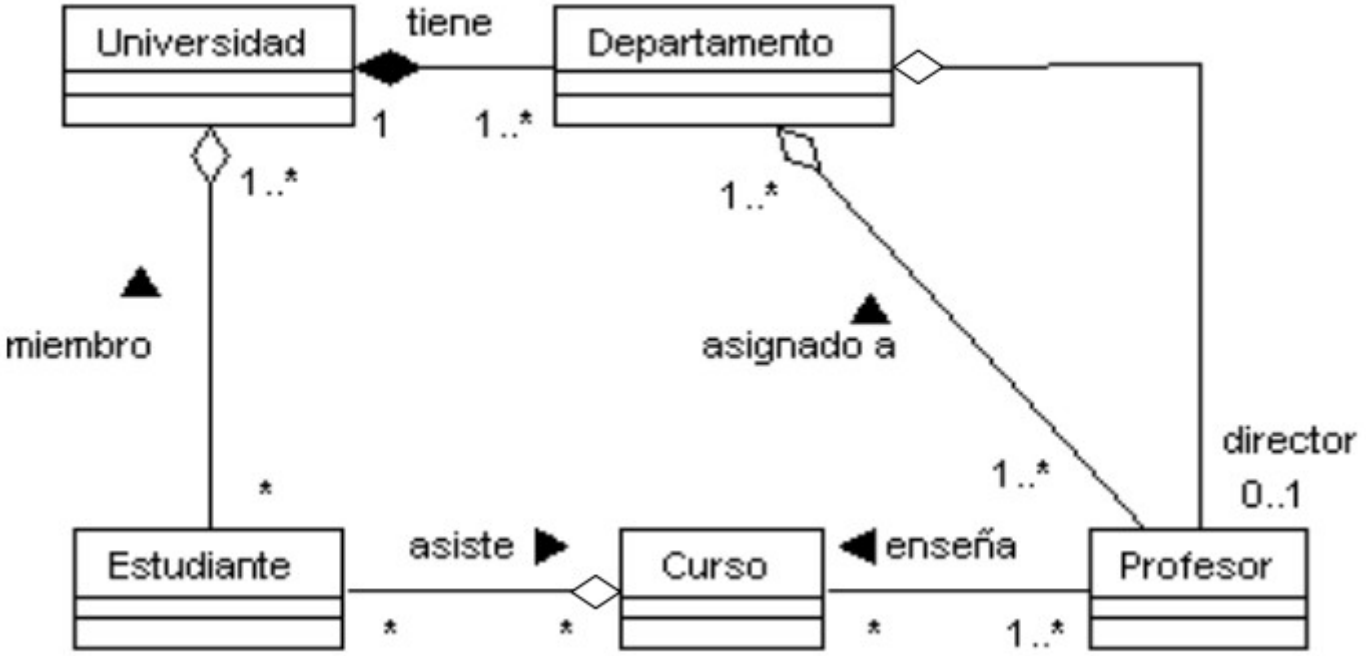
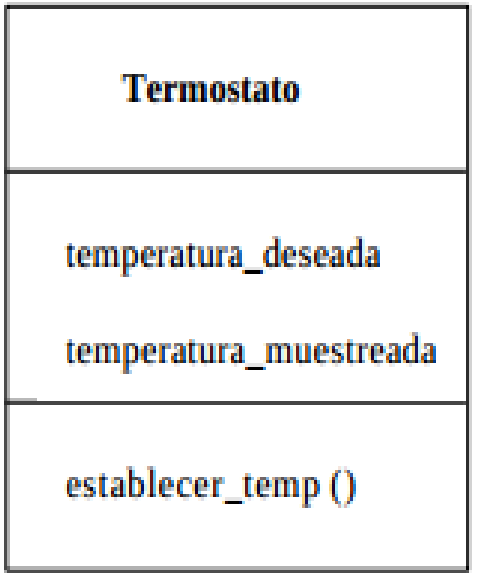


Diagrama de clases
Símbolos y notaciones básicas

- Clase
- Visibilidad
- Asociaciones
- Cardinalidad
- Restricciones
- Composición y agregación
- Generalización

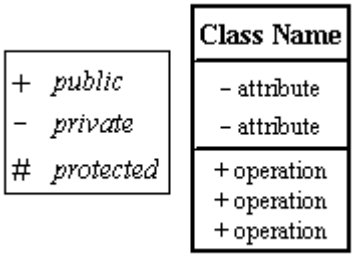
Clase

- Las clases representan una abstracción de entidades con características comunes.
- Se representa mediante una clase dividida en tres partes:
  - Nombre de clase
  - Atributos
  - Operaciones



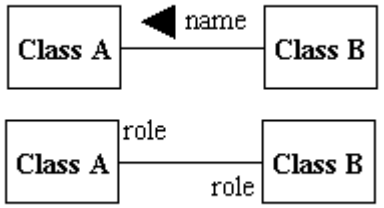
Visibilidad

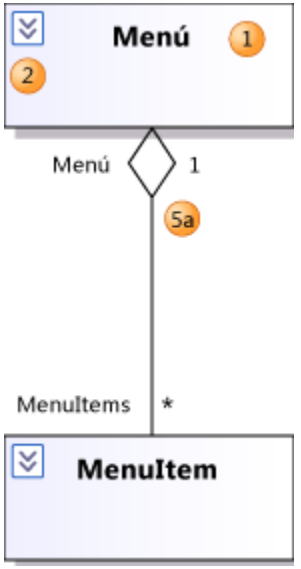
- Los marcadores de visibilidad se utilizan para indicar quiénes pueden acceder a la información de una clase.
  - Private: Sólo la misma clase puede ver la info.
  - Protected: Sólo las clases que heredan.
  - Public: Todo el resto de clases.

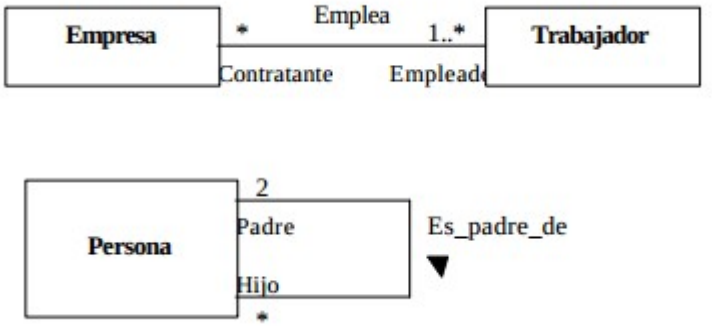


Asociaciones

- Representan relaciones estáticas entre las clases (entidades).
- Atributos disponibles.
  - Nombre.
  - Dirección.
  - Roles: La forma en que las clases se ven entre sí.
  - Generalmente se indica el rol o el nombre, no ambos.

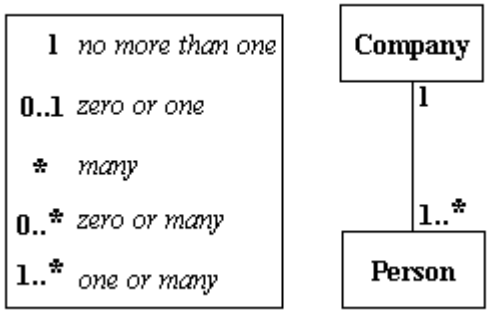


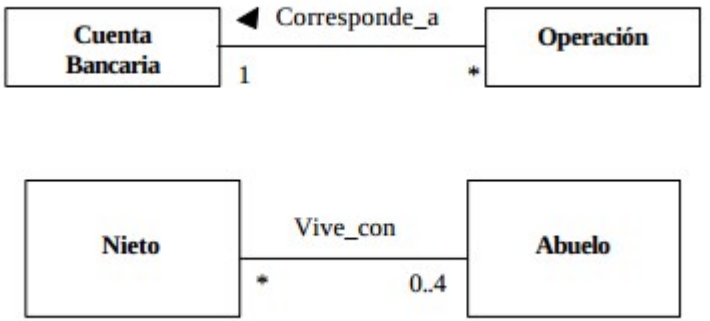




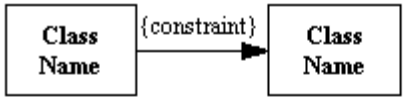
Cardinalidad

- Indican el número de instancias de una clase asociadas con una instancia de otra clase.
- “Una compañía puede tener uno o más empleados, pero un empleado trabaja sólo para una compañia”.

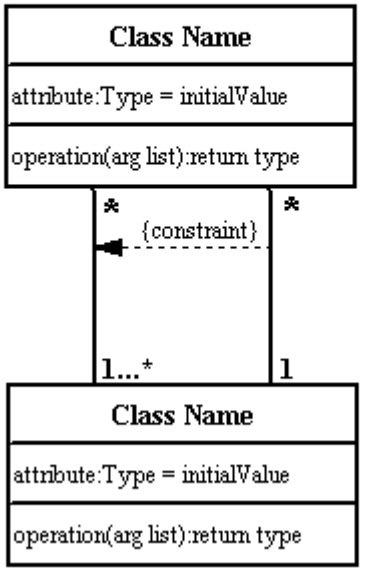




Restricciones



Simple Constraint



Herencia

- La relación de herencia se representa mediante un triángulo en el extremo de la relación que corresponde a la clase más general o la clase padre.