PARADIGMA ORIENTADO A OBJETOS

PARADIGMAS DE PROGRAMACIÓN I

Carlos Rojas Sánchez

Licenciatura en Informática

Universidad del Mar

Contenido

1. Evolución y beneficios del paradigma orientado a objetos sobre otros paradigmas

2. Componentes básicos de la programación orientada a objetos

3. Características de la programación orientada a objetos

4. Principios de la POO

5. Ingeniería de software orientada a objetos

6. Diseño de clases empleando UML

Evolución y beneficios del paradigma orientado a objetos sobre otros paradigmas

Década de 1960

Los primeros conceptos relacionados con la programación orientada a objetos comenzaron a emerger en la década de 1960 con la creación del lenguaje Simula. Desarrollado por Ole-Johan Dahl y Kristen Nygaard en el Centro Noruego de Computación (Norsk Regnesentral), Simula fue originalmente creado para realizar simulaciones. Simula 67 fue la versión de Simula que introdujo conceptos como clases, objetos y herencia. Aunque no era conocido por el nombre de "orientado a objetos" en ese momento, estos conceptos fundamentales sentaron las bases para futuros desarrollos.

Década de 1970

En la década de 1970, Alan Kay, un investigador de Xerox PARC, acuñó el término "orientación a objetos" para describir un enfoque de programación basado en la simulación de sistemas biológicos. Kay desarrolló el lenguaje de programación Smalltalk, que se convirtió en una influencia importante en el desarrollo posterior de la programación orientada a objetos.

Década de 1970

Smalltalk fue el primer lenguaje de programación en implementar completamente el paradigma orientado a objetos tal como lo conocemos hoy. Smalltalk introdujo una serie de ideas revolucionarias, incluyendo el envío de mensajes entre objetos, la encapsulación y la herencia dinámica.

Década de 1980

La década de 1980 fue testigo del crecimiento y la popularización de la programación orientada a objetos. Bjarne Stroustrup en AT&T Bell Labs desarrolló C++, un lenguaje que evolucionó a partir del lenguaje C con la adición de características orientadas a objetos.

Década de 1980

C++ incorporó clases y objetos, así como constructores, destructores y sobrecarga de operadores. Su compatibilidad con C y su capacidad para manejar tanto programación de bajo nivel como de alto nivel hicieron de C++ un lenguaje popular en la industria del software.

Década de 1980

En la misma década, Brad Cox y Tom Love desarrollaron Objective-C, combinando las capacidades orientadas a objetos de Smalltalk con el lenguaje de programación C. Objective-C se convirtió en el lenguaje principal para el desarrollo de software en plataformas de Apple durante muchos años.

Década de 1990

En la década de 1990, Java se convirtió en uno de los lenguajes de programación más influyentes en el ámbito de la programación orientada a objetos. Desarrollado por Sun Microsystems (ahora parte de Oracle) y diseñado por James Gosling y su equipo, Java combinó la sintaxis de C++ con una arquitectura orientada a objetos.

Década de 2000

C#, desarrollado por Microsoft como parte de su plataforma .NET, se lanzó a principios de la década de 2000. Influenciado por C++ y, muy fuertemente por JavaScript, C# ha sido adoptado como el lenguaje principal para el desarrollo en la plataforma .NET, incluyendo aplicaciones de escritorio, web y móviles.

Ventajas

- Por medio de la herencia, podemos eliminar código redundante y extender el uso de clases existentes.
- Es posible tener múltiples copias de un objeto que coexisten sin ninguna interferencia.
- Es posible transformar objetos del dominio del problema a otros en los programas.
- Es fácil dividir el trabajo de un proyecto basado en objetos.

Componentes básicos de la programación orientada a objetos

Programación Orientada a Objetos (POO)

Mediante la POO, a la hora de tratar un problema, podemos descomponerlo en subgrupos de partes relacionadas. Estos subgrupos pueden traducirse en unidades autocontenidas llamadas objetos. Antes de la creación de un objeto, se debe definir en primer lugar su formato general, su plantilla, que recibe el nombre de clase.

Clases

Una clase describe las estructuras de datos que lo forman y las funciones asociadas con él. Una clase es un modelo con el que se construyen los objetos.

Objetos

Un objeto es un ejemplar concreto de una clase, que se estructura y comporta según se definió en la clase, pero su estado es particular e independiente del resto de ejemplares. Al proceso de crear un objeto se le llama generalmente instanciar una clase.

Métodos y mensajes

Un método define una operación sobre un objeto. En general, realizan dos posibles acciones: consultar el estado del objeto o modificarlo. Los métodos disponen de parámetros que permiten delimitar la acción del mismo.

Métodos y mensajes

- Consultan o modifican un atributo, normalmente nos referenciaremos a ellos como: getters & setters
- Realizan operaciones sobre el conjunto de atributos, calculando valores o realizando modificaciones.
- Inicializan los atributos al principio del ciclo de vida, o liberan los recursos al final del ciclo; nos referiremos a ellos como constructores o destructores

Métodos y mensajes

Un mensaje es la invocación de un método de un objeto. Podemos decir que un objeto lanza un mensaje (quien realiza la invocación) y otro lo recibe (el que ejecuta el método). Podemos decir que una clase es la descripción e implementación de un conjunto de atributos y métodos.

Características de la programación orientada a objetos

Características de la POO

- La implementación y desarrollo del paradigma está fundamentado en los objetos.
- Dar prioridad a los objetos y su abstracción como una parte fundamental en la solución de problemas.
- Definir los métodos, propiedades y características de los objetos así como su relación(interacción).

Abstracción

Proceso que implica reconocer y enfocarse en las características importantes de una situación u objeto, y filtrar o ignorar todos los detalles no esenciales.

Modularidad

- Propiedad que permite subdividir una aplicación en partes más pequeñas.
- La modularidad debe seguir los conceptos de bajo acoplamiento y alta cohesión.

Encapsulamiento

- Es el proceso de ocultar todos los detalles internos de un objeto del mundo exterior.
- Es una barrera protectora que impide que el código y los datos sean accesibles al azar por otro código o por fuera de la clase.

Herencia

Una clase heredada de otra quiere decir que esa clase obtiene los mismos métodos y propiedades de la otra clase.

Polimorfismo

Es la habilidad de dos o más objetos pertenecientes a diferentes clases para responder exactamente al mismo mensaje (llamada de método) de diferentes formas específicas de la clase.

Polimorfismo

La sobrecarga es definir dos o más métodos con el mismo nombre, pero con parámetros diferentes por cantidad o tipo. El objetivo de la sobrecarga es reducir el número de identificadores distintos para una misma acción pero con matices que la diferencian. La sobrecarga se puede realizar tanto en métodos generales, como en constructores.

La sobrecarga es un polimorfismo estático, ya que es el compilador quien resuelve el conflicto del método a referenciar.

Principios de la POO

Principios

Según Robert C. Martin existen 5 principios básicos que constituyen la programación orientada a objetos.

Principio de responsabilidad única

- Cada clase debe tener una única responsabilidad, y esta responsabilidad debe estar contenida únicamente en esa clase.
- Cada responsabilidad es el eje y la razón de cambio.
- Para contener la propagación del cambio, se deben separar las responsabilidades.

Principio de abierto - cerrado

- Una entidad (clase, módulo, función, etc.) debe quedarse abierta para su extensión, pero cerrada para su modificación.
- Si un cambio impacta a varios módulos, entonces la aplicación no está bien diseñada.
- Se deben diseñar módulos que procuren no cambiar y así, reutilizar el código más adelante (extensión).

Principio de sustitución de Liskov

- Si en alguna parte de un programa se utiliza una clase, y esta clase es extendida, se puede utilizar cualquiera de las clases hijas sin que existan modificaciones en el código.
- Cada clase que hereda de otra puede usarse como su padre sin necesidad de conocer las diferencias entre ellas.

Principio de segregación de interfaz

- Este principio hace referencia a que muchas interfaces cliente específicas son mejores que una interfaz de propósito general.
- Se aplica a una interfaz amplia y compleja para dividirla en otras más pequeñas y específicas, de tal forma que cada cliente use solo aquella que necesite pudiendo así ignorar al resto.

Principio de inversión de dependencias

- Los módulos de alto nivel no deben depender de los módulos de bajo nivel. Ambos deben depender de abstracciones.

Ingeniería de software orientada a objetos

Ingeniería del Software

- La Ingeniería del Software aplica los principios de la ciencia de la computación y las matemáticas para lograr soluciones costo-efectivas a los problemas de desarrollo de software.
- Proceso de ingeniería de software: Conjunto de etapas parcialmente ordenadas con la intención de lograr un producto software de calidad.

- Análisis-Diseño Orientado a Objetos: Es un método de análisis y diseño que examina los requerimientos desde la perspectiva de las clases y objetos encontrados en el vocabulario del dominio del problema.
- Metodología de Desarrollo: Es un conjunto integrado de técnicas y métodos (actividades) que permiten obtener de forma homogénea (sistemática) y abierta (a cambios y adaptaciones), cada una de las fases del ciclo de vida del software.

Ciclo de vida del software

Es el marco de referencia que contiene los procesos, actividades y tareas involucradas en el desarrollo, operación y mantenimiento de un producto SW, abarcando la vida del sistema desde la definición de los requisitos hasta la finalización de su uso.

El proceso de desarrollo OO

- Planificación y Especificación de Requisitos: Estudiar la especificación de requisitos para descubrir las secuencias típicas de acciones desde la perspectiva del usuario. Estas acciones son los denominados casos de uso.
- Un caso de uso es una secuencia típica de acciones en un sistema, desde el punto de vista del usuario, que muestra cómo el sistema interacciona con el exterior y que se obtiene como resultado del uso del sistema.

El proceso de desarrollo OO

- Los casos de uso son descritos en un documento en el que se detallan los siguientes puntos de cada uno.
  - Nombre del caso de uso
  - Actores participantes
  - Tipo de caso (importancia del mismo – primario, secundario)
  - Descripción del caso de uso

El proceso de desarrollo OO

- Análisis: Se intenta llegar a una buena comprensión del problema por parte del equipo de desarrollo, sin entrar en cómo va a ser la solución en cuanto a detalles de implementación.
- Trabajamos con los modelos de casos de uso construidos en la fase anterior, ampliándolos y refinándolos.
- Se construye un Modelo de Objetos Conceptual o Modelo de Análisis mediante un diagrama de clases, compuesto de clases y relaciones entre las clases.

El proceso de desarrollo OO

- En el Modelo de Objetos Conceptual se tiene una representación de conceptos (objetos - clases) del mundo real, es una primera aproximación al modelo de diseño.
- Se deberán identificar los conceptos más importantes del sistema (objetos físicos, roles de una persona, etc.), los atributos de los mismos y las relaciones existentes entre ellos.

El proceso de desarrollo OO

- Diseño: En la fase de Diseño se crea una solución a nivel lógico para satisfacer los requisitos, basándose en el conocimiento reunido en la fase de análisis.
- Las tareas que se realizan en esta fase son las siguientes:
  - Definir el Diagrama de Clases de Diseño detallado.
  - Definir las estructuras de datos necesarias para almacenar la información que utiliza el sistema.
  - Definir la Interfaz de Usuario e Informes.

El proceso de desarrollo OO

- Los diagramas de clases definidos en la fase anterior se pueden refinar con la especificación de atributos y operaciones para cada una de las clases y las relaciones con otras clases (generalización, agregación, composición, uso, etc.). Con la información obtenida en los casos de uso, se pueden derivar las operaciones y asignarse a las clases existentes.

Diseño de clases empleando UML

Unified Modeling Language (UML)

- El UML define un lenguaje de modelado orientado a objetos común para visualizar, especificar, construir y documentar los componentes de un sistema software OO.
- Está respaldado por el Object Management Group (OMG).
- El UML no es una metodología, sino una notación que trata de posibilitar el intercambio de modelos de software.

Unified Modeling Language (UML)

- Un modelo es una simplificación de la realidad creada para comprender mejor un sistema.
- Un proceso de desarrollo de software debe ofrecer un conjunto de modelos que permitan expresar el producto desde cada una de las perspectivas de interés.

Unified Modeling Language (UML)

- Los modelos de UML se utilizan para representar las distintas fases o etapas que se plantean en una metodología de desarrollo software. Ejemplo de una metodología es el Proceso Unificado.
- UML utiliza modelos orientados a objetos: Representación de un sistema a partir de los objetos o entidades que lo constituyen, con atributos y operaciones, y relaciones con otros objetos.

Unified Modeling Language (UML)

- Diagramas para modelar el Comportamiento del Sistema
  - Diagrama de Casos de Uso: Muestra un conjunto de casos de uso y actores y sus relaciones.
  - Diagrama de Secuencia: Diagrama de interacción con la relación temporal de los mensajes y los objetos.
  - Diagrama de Colaboración: Diagrama de interacción que resalta la organización estructural de los objetos que envían y reciben mensajes.
  - Diagrama de Estados: Muestra una máquina de estados, que consta de estados, transiciones, eventos y actividades. Vista dinámica del sistema.
  - Diagrama de Actividades: Muestra el flujo de actividades dentro de un sistema.

Unified Modeling Language (UML)

- Diagramas para modelar la Estructura del Sistema
  - Diagrama de Clases: Muestra un conjunto de clases, interfaces y colaboraciones, así como sus relaciones.
  - Diagrama de Objetos: Muestra un conjunto de objetos y sus relaciones.
  - Diagrama de Componentes: Muestra la organización y las dependencias entre un conjunto de componentes.
  - Diagrama de Despliegue: Representa la infraestructura de un sistema en tiempo de ejecución.

Diagrama de Clases

- Un Diagrama de Estructura Estática (conocido más popularmente como Diagrama de Clases) muestra la estructura estática del modelo del sistema, es decir, todo aquello que exista en el sistema, mostrando su estructura interna así como sus relaciones entre los diferentes elementos.
- En un diagrama de clases, los elementos que nos vamos a encontrar son: las clases y las relaciones entre clases.

Diagrama de Clases

```
public class HolaMundo {
   public HolaMundo() {
   }
   public static void main(String[] args) {
      System.out.println("Hola Mundo");
   }
}
```

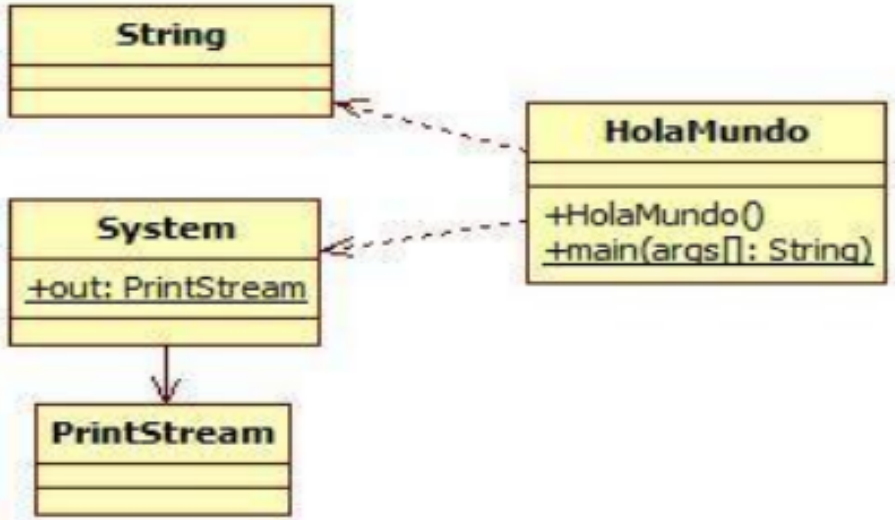

Diagrama de Clases

- Clase: Nombre + Atributos + Operaciones (métodos)

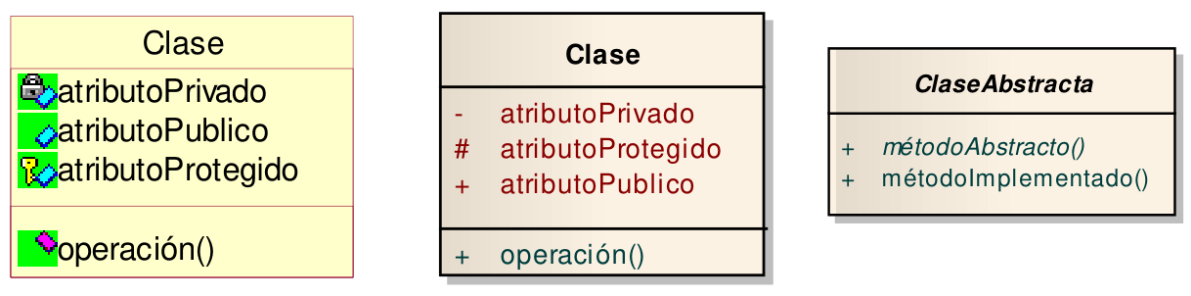

Diagrama de Clases

- Interfaces: Representan un conjunto de operaciones que especifican los servicios que puede brindar una clase o componente y nunca debe especificar sus implementaciones.

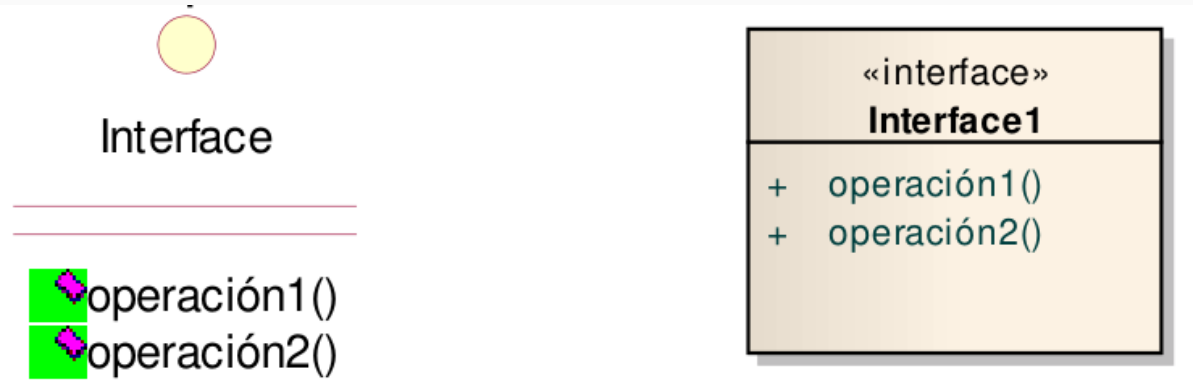

Diagrama de Clases

- Las asociaciones pueden estar formadas por un número indeterminado de clases pero las más comunes y utilizadas son las asociaciones binarias, es decir, aquellas relaciones entre dos clases. En los extremos de la relación especificaremos la cardinalidad y también podrán aparecer los atributos que representan la asociación.

Diagrama de Clases

- Agregación (relación del tipo todo/parte) entre clases se expresa mediante un rombo adyacente a la clase que representa la totalidad y de dicho rombo parten las asociaciones al resto de clases que forman dicha agregación.

Diagrama de Clases

- Composición (relación de pertenencia) es representada de igual forma que la agregación pero con el interior del rombo pintado de negro.

Diagrama de Clases

- Herencia (o generalización) se representa mediante un triángulo unido a la clase padre por un vértice y del cual salen las relaciones a las clases hijas.

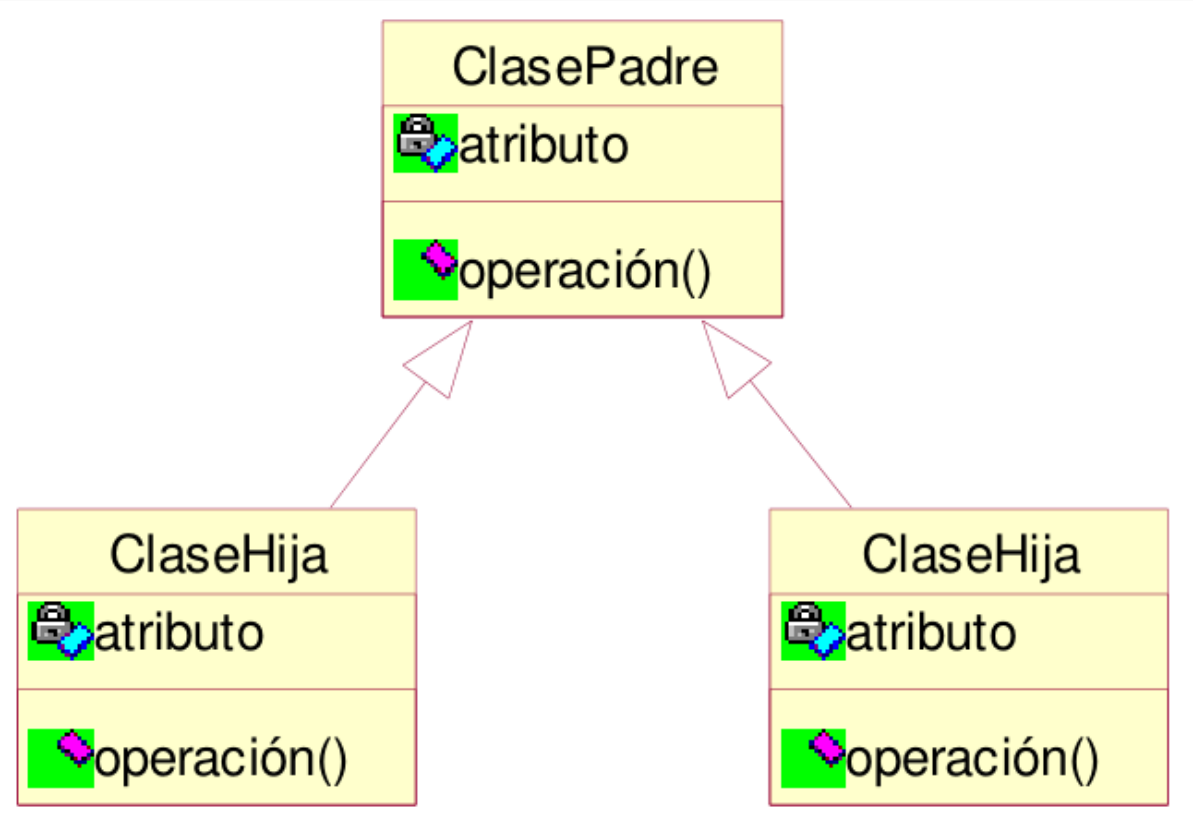

Diagrama de Clases

- En este ejemplo se pueden apreciar las relaciones que hay entre una facultad, sus departamentos, sus profesores y sus alumnos:

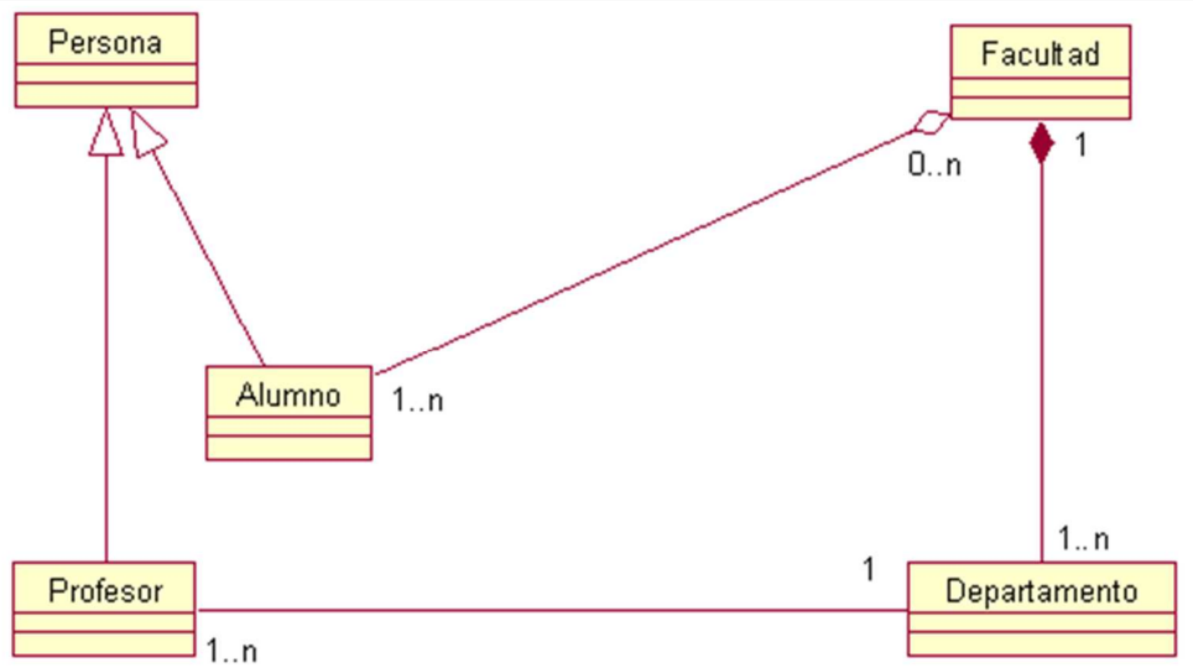

Diagrama de Clases

- Realizar un análisis sintáctico - gramatical de la documentación existente
  - Utilizar la documentación de los casos de uso.
  - Subrayar cada nombre (sustantivo) o cláusula nominal.
- Decidir qué objetos se admiten como objetos del sistema.
  - A partir de los nombres subrayados, proponer varios objetos potenciales.
- Identificación de relaciones
  - Las relaciones se obtienen analizando la estructura de la información del sistema.
  - Expresiones como: es, tiene, consta de, en la descripción del sistema, sugieren la existencia de relaciones entre objetos.

Diagrama de Clases

- Identificación de atributos
  - Los atributos se obtienen de la lista de objetos candidatos y de la descripción del sistema.
  - Los objetos descartados por simples serán atributos.
- Identificación de operaciones
  - Las operaciones de los objetos se derivan de los verbos que aparecen en la descripción del sistema.
  - Los parámetros de las operaciones se derivan de la información intercambiada por los objetos que interactúan.