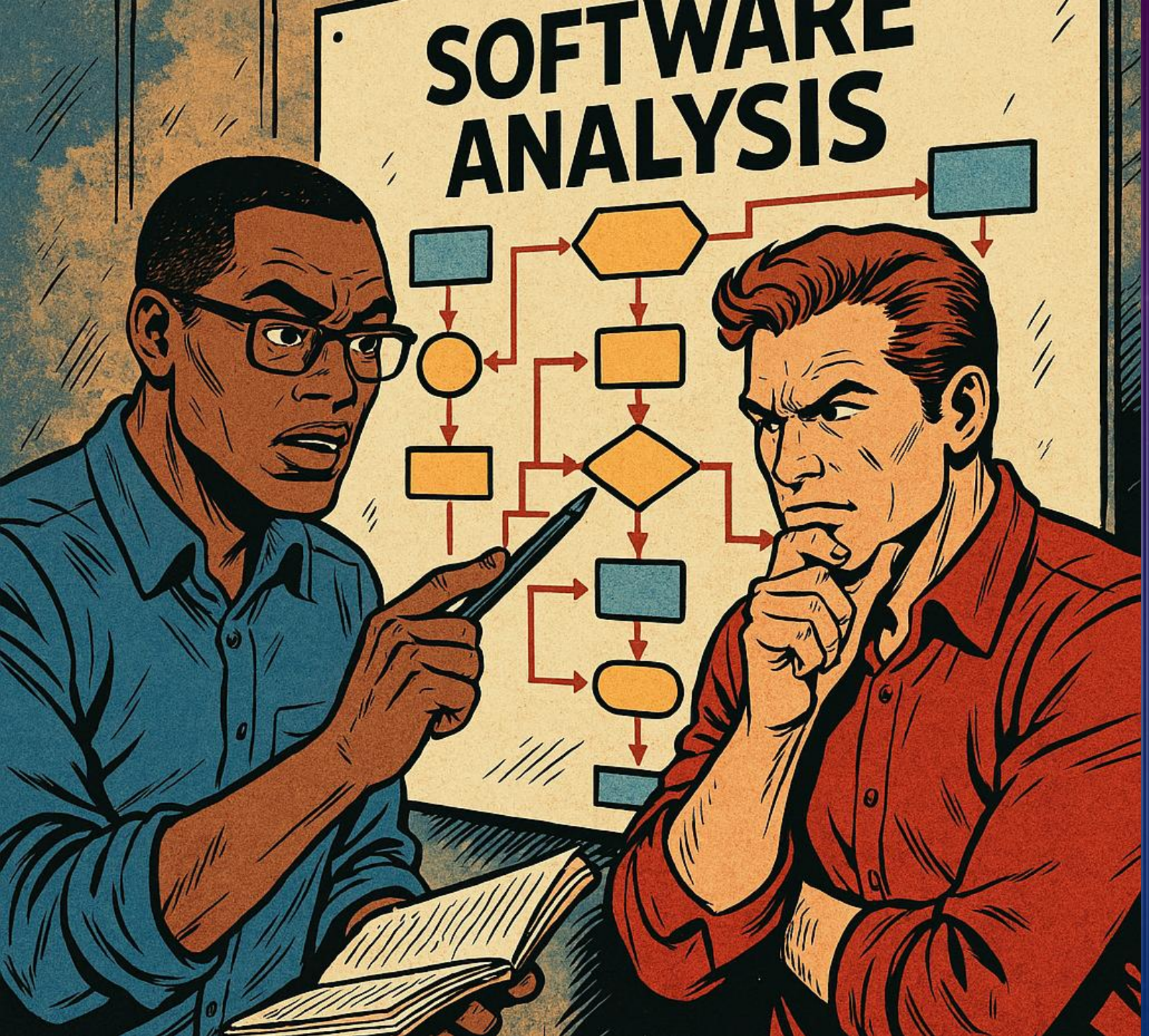
UPEN
UNIVERSIDAD POLITÉCNICA DEL ESTADO DE NAYARIT
BIS

SOFTWARE ANALYSIS AND DESIGN

I. SOFTWARE ENGINEERING

ITIID 4

I.S.C. LUIS GERARDO AGUIRRE CERVANTES



WHAT IS SOFTWARE ANALYSIS?

- Analysis is the phase where the problem to be solved is understood.
- Its main objective is to define what the system should do, without yet considering how it will be built.

WHAT IS THE DIFFERENCE BETWEEN ANALYSIS AND DESIGN?

Software Analysis

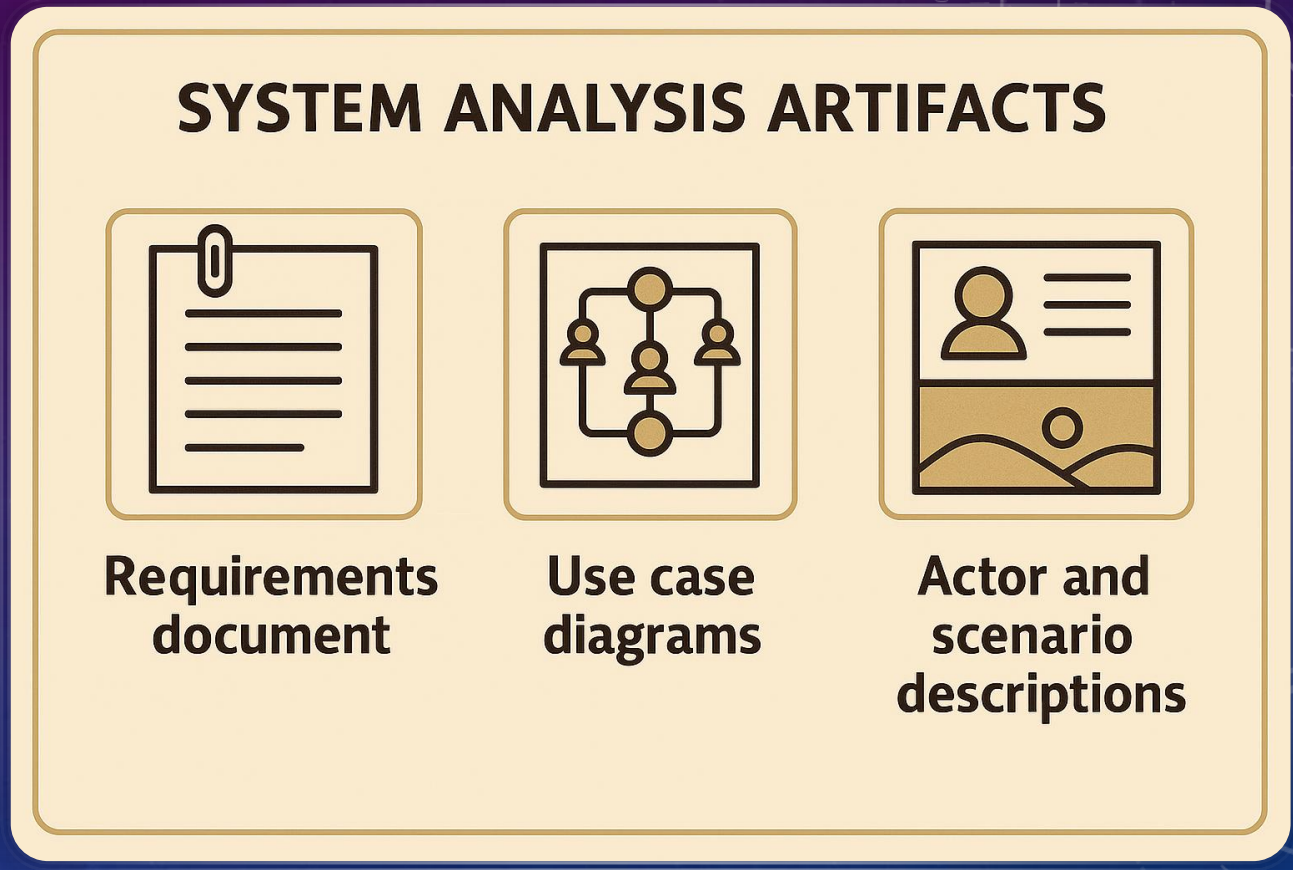
- **What is it?** A process of researching and deeply understanding the requirements and objectives that the software must meet.
- **Objective:** To define what the software must do, not how it will do it.
- **Activities:** Interacting with users, identifying problems, defining functionalities, and creating the Requirements Specification Document (RSD).

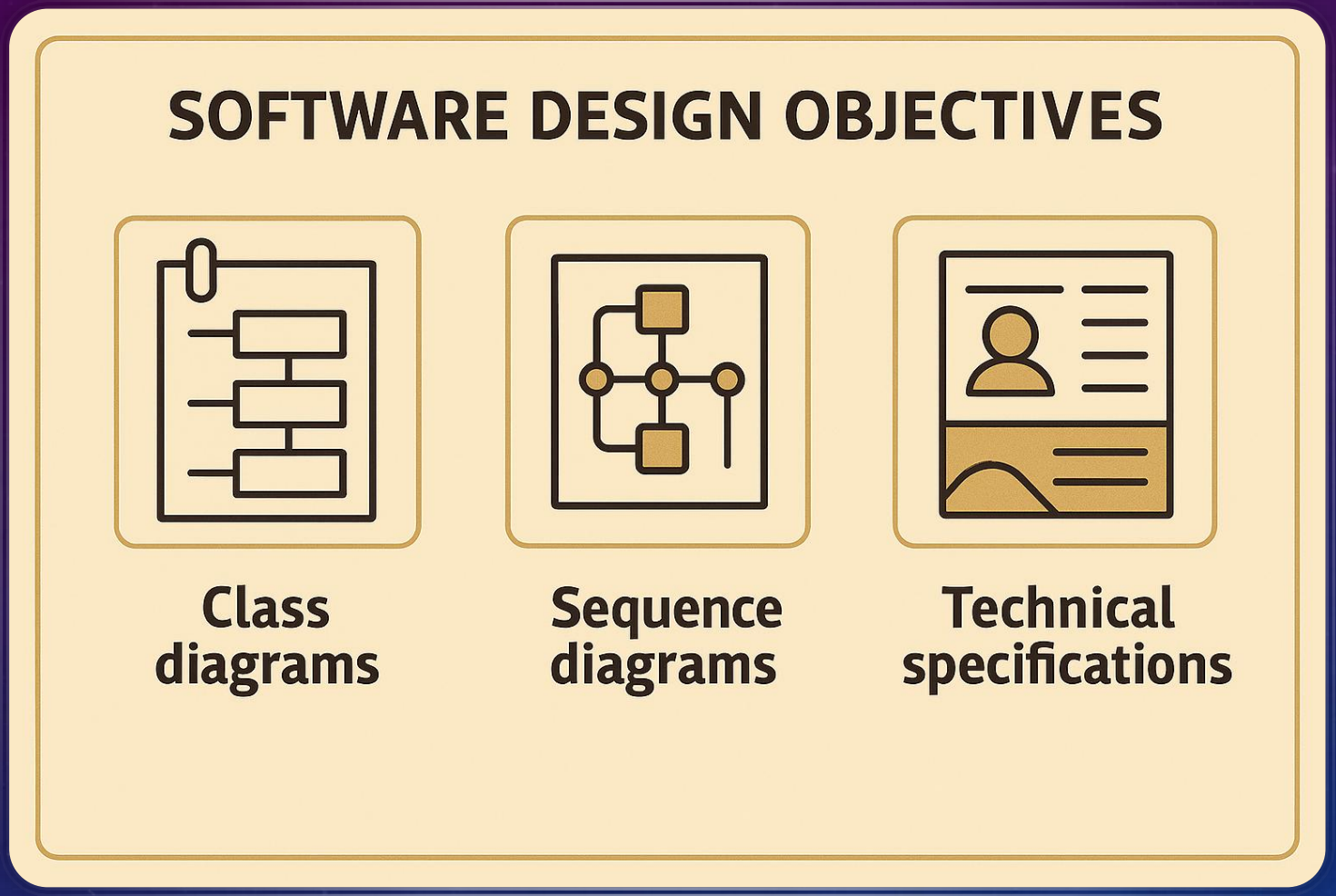
Software Design

- **What is it?** Planning and creating the architecture and internal components of the program.
- **Objective:** To define *how* the software will be built, its structure, modules, and relationships.
- **Key aspects:**
  - Architecture Design: System overview.
  - Detailed Design: Modules, algorithms, databases, interfaces.
  - Design Patterns: Reusable solutions.
  - Human-Center Interaction (HCI) Design: User experience.
  - Modeling: Use of notations such as UML (Class Diagrams, Sequence Diagrams) for visualization.

WHAT ARE THE RESULTS OF SOFTWARE ANALYSIS?

- Requirements document.
- Use case diagrams.
- Actor and scenario descriptions.





WHAT ARE THE RESULTS OF SOFTWARE DESIGN?

- Class diagrams.
- Sequence diagrams.
- System architecture.
- Technical specifications.

Aspect	Analysis	Design
Focus	Understand the problem	Create the solution
Key question	What is needed?	How will it be done?
Level	Conceptual	Technical
Outcome	Requirements	Architecture and models

IMPORTANCE OF ANALYSIS AND DESIGN

- They prevent costly errors in later stages.
- They improve communication between the team and the client.
- They allow for better development planning.
- They increase the quality of the final software.
- It ensures reliable, usable, secure, and maintainable software.
- It creates robust and scalable solutions for the future.
- It is a fundamental part of the Software Development Life Cycle (SDLC).

WHAT IS THE JOB OF A SYSTEMS ANALYST?

1. Requirements and technical specifications analysis
2. Systems design and development
3. Testing and debugging
4. Implementation and training
5. Maintenance and technical support

KEY SKILLS FOR BEING A SUCCESSFUL SYSTEMS ANALYST

Being a successful systems analyst requires a unique combination of technical and interpersonal skills. Below are some of the most important skills for excelling in this profession:

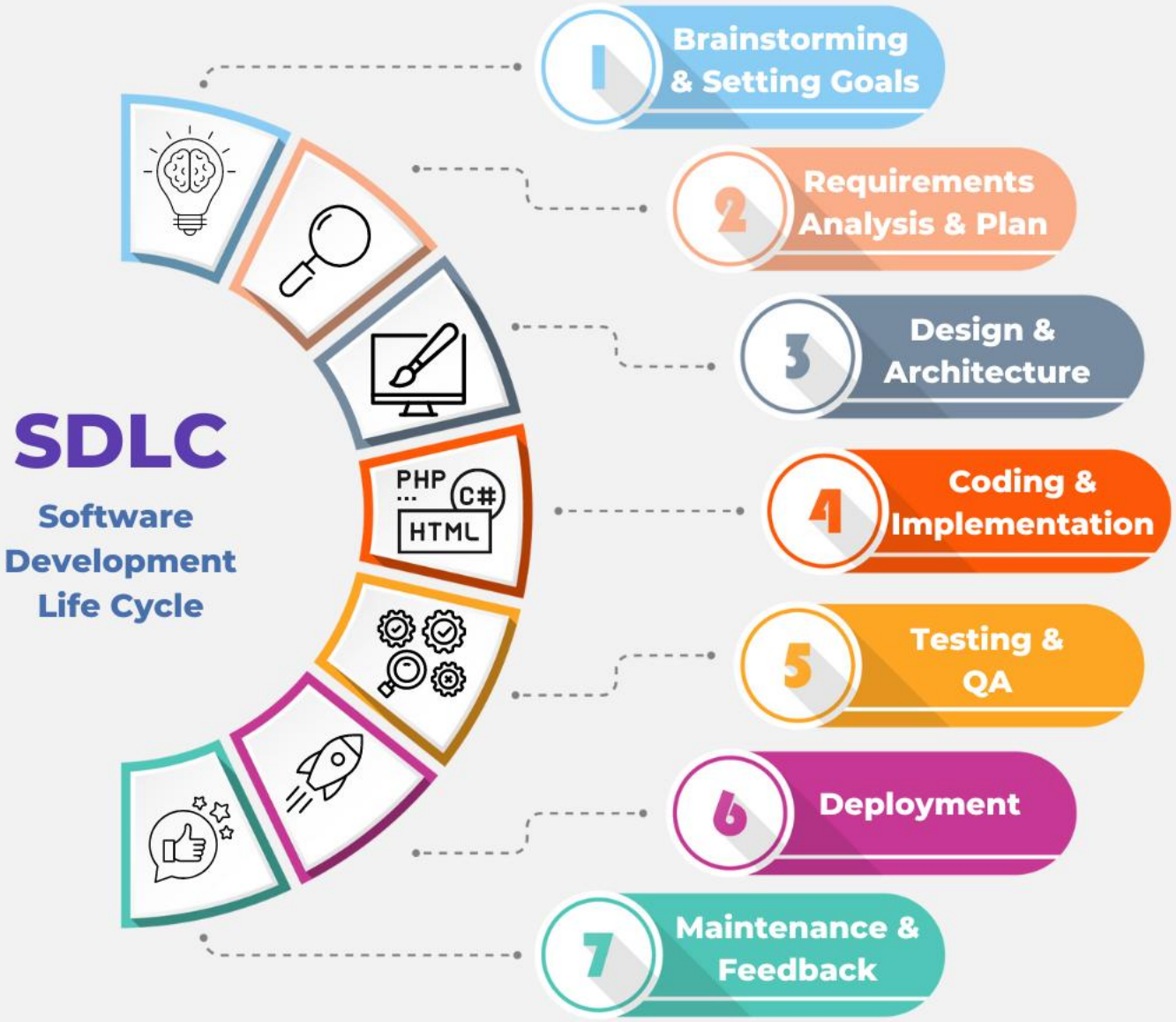
Solid technical knowledge: Systems analysts must have a deep understanding of software development principles and techniques, as well as the technologies and tools used in the design and development of computer systems.

Problem-solving skills: Analysis and problem-solving are fundamental parts of a systems analyst's job. They must be able to quickly identify and resolve problems that arise during systems development or implementation.

Effective communication skills: Systems analysts interact with users, development teams, and other stakeholders. Therefore, effective communication skills are essential for understanding and conveying technical information clearly and concisely.

Teamwork skills: Collaboration with other IT professionals, such as developers, designers, and database administrators, is essential for the success of a systems analyst. The ability to work effectively as part of a team is key to achieving common goals.

Customer focus: A good systems analyst must have a strong customer focus and understand the needs and expectations of end users. This allows them to design IT solutions that meet the organization's actual needs.



WHAT IS THE SOFTWARE LIFE CYCLE?

The software development life cycle (SDLC) is a structured process that guides software development from initial idea to maintenance, including key phases such as planning, requirements analysis, design, development (coding), testing, implementation (deployment), and ongoing maintenance, with the goal of creating high-quality software efficiently and minimizing risks.

BRAINSTORMING AND SETTING GOALS

Brainstorming marks the outset of the software development life cycle, when teams join together to explore possibilities and conceive inventive solutions. In this early stage, business objectives and goals are identified along with any requirements or specifications that may be needed for a successful project outcome. Potential risks in achieving these goals must also be considered at this juncture.

REQUIREMENTS ANALYSIS & PLAN DEVELOPMENT

The requirements analysis stage is when the client's expectations for the software are explored in depth. Here, teams create an action plan that outlines all necessary tasks to be completed and their order of importance. The plan will also consider any potential roadblocks, such as staff availability or budget constraints, that may hinder the project from progressing smoothly.

Software Development Requirement Specification (SRS) is often used in this stage to provide more clarity on the project's requirements.

DESIGN & ARCHITECTURE

During this phase of product creation, product architects and developers work together to bring the concept into reality. After researching solutions for anticipated issues and risks previously identified in stage 2's SRS document, several design approaches are documented within a Design Document Specification (DDS). This DDS will provide developers with an essential source during production's stage 4 as they code based on its content.

CODING & IMPLEMENTATION

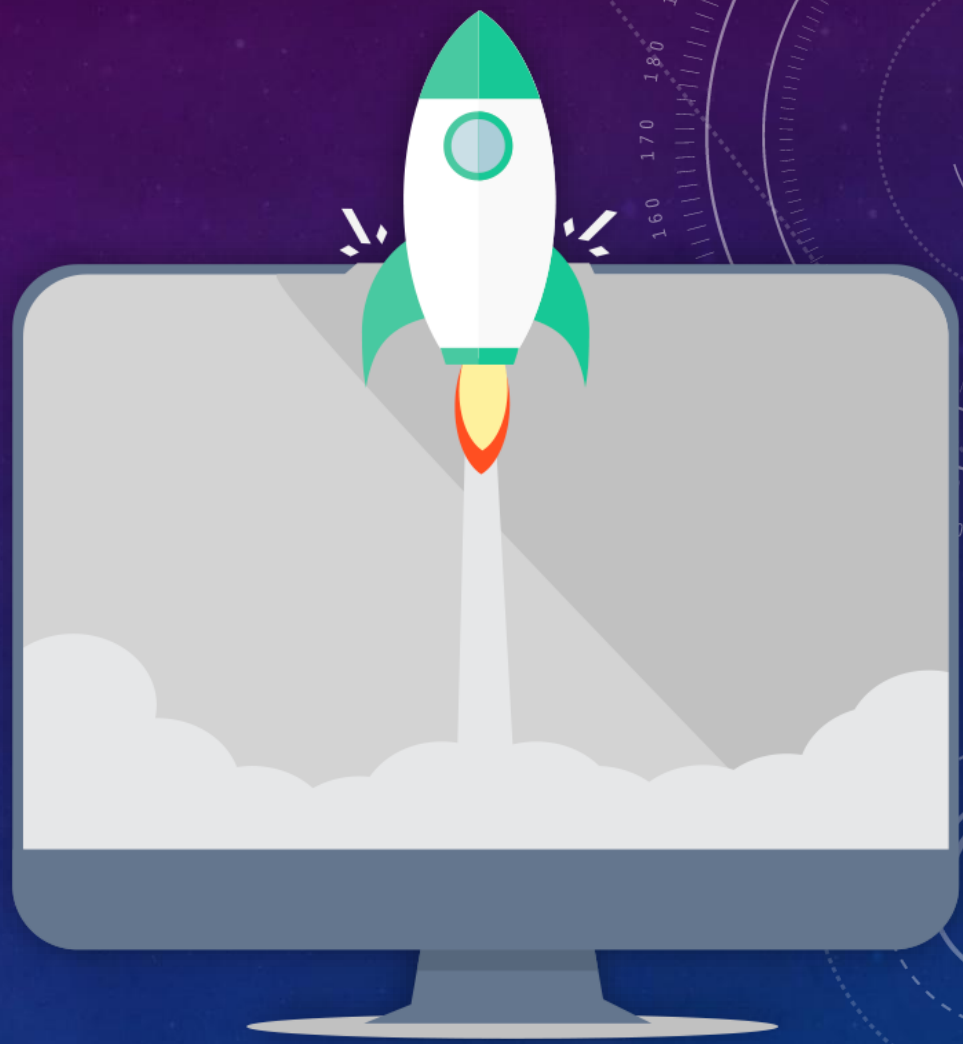
The implementation phase is where developers turn the design into a tangible product through coding. This requires teams to be familiar with programming languages and frameworks, ensuring that the source code written meets all project requirements. This might be the most time-consuming phase of the SDLC, so you can outsource this stage to other **software outsourcing companies** to save both time and money.

TESTING & QUALITY ASSURANCE

Testing is an essential part of the software development life cycle ensuring quality coding and a seamless user experience. A modern approach to this involves testing at various stages throughout, offering improved risk management while maintaining excellent code integrity. This emphasizes the importance of testers reviewing each component as well as the end-to-end integration for a polished product free from bugs or other undesirable issues.

DEPLOYMENT

Once the software passes all tests and QA checks, it’s time to deploy and release the product. Depending on the type of software, deployment can be a simple process or require more intricate steps such as setting up databases or configuring servers for an optimal environment. All these measures must be taken into account when working out a timeline for successful completion.



MAINTENANCE & FEEDBACK

Maintenance is an important final step, as software products require consistent updates, security patches and bug fixes. This stage also involves monitoring the system's performance to ensure it is running at optimum levels for users.

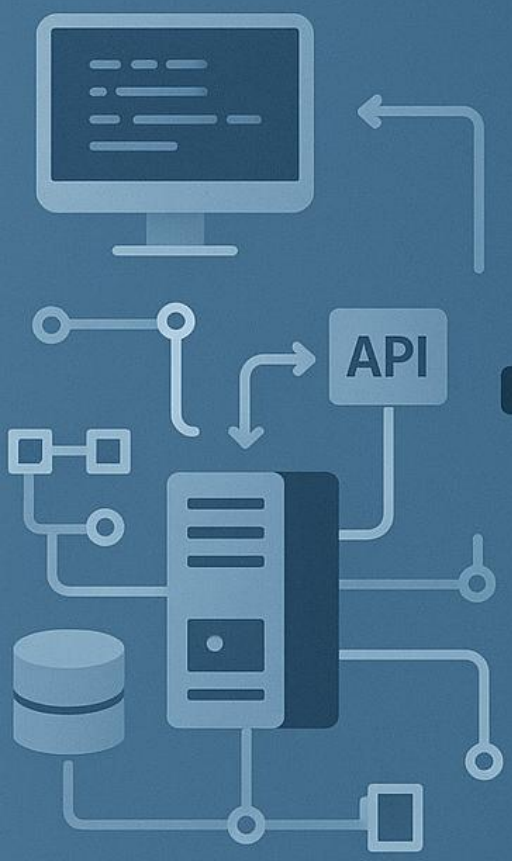
In addition, feedback from users must be taken into account and used to assess user satisfaction. Collecting this data can help inform future plans and releases, ensuring a high-quality product that is tailored to the needs of its users. By following these steps, teams can guarantee a successful software development life cycle from start to finish.

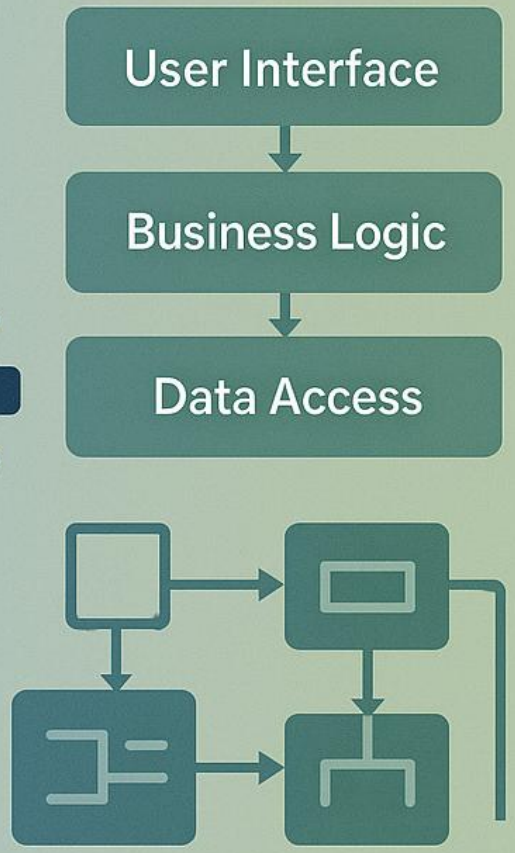
SOFTWARE DEVELOPMENT LIFE CYCLE MODELS

The software life cycle is the complete process a software product goes through—from the moment someone has the idea to create it, all the way until the software is retired or replaced. It provides a structured, repeatable way to build high-quality software.

The concept is widely defined in industry sources as a structured methodology used to build, deliver, and maintain software systems, breaking the work into clear phases with specific objectives.

System Design vs Software Architecture





System Design

WHAT IS SOFTWARE ARCHITECTURE?

It seems simple: software should be intuitive, easy to use, and designed to help you meet your technical and business goals. But as soon as you dig deeper on the backend, you'll likely uncover a complex web of components and processes, all working together to keep things running. To navigate the challenges of software design—and to reap the most benefits—it's important to plan ahead.

That’s where software architecture comes in. Software architecture refers to a set of fundamental structures that developers use as an overarching visual guide when planning for and building out software solutions. These frameworks also ensure that a project meets its technical and business needs by planning for important functionality such as scalability, reusability, and security. These plans are then broken down into individual components and code during the design phase.

BENEFITS OF GOOD SOFTWARE ARCHITECTURE

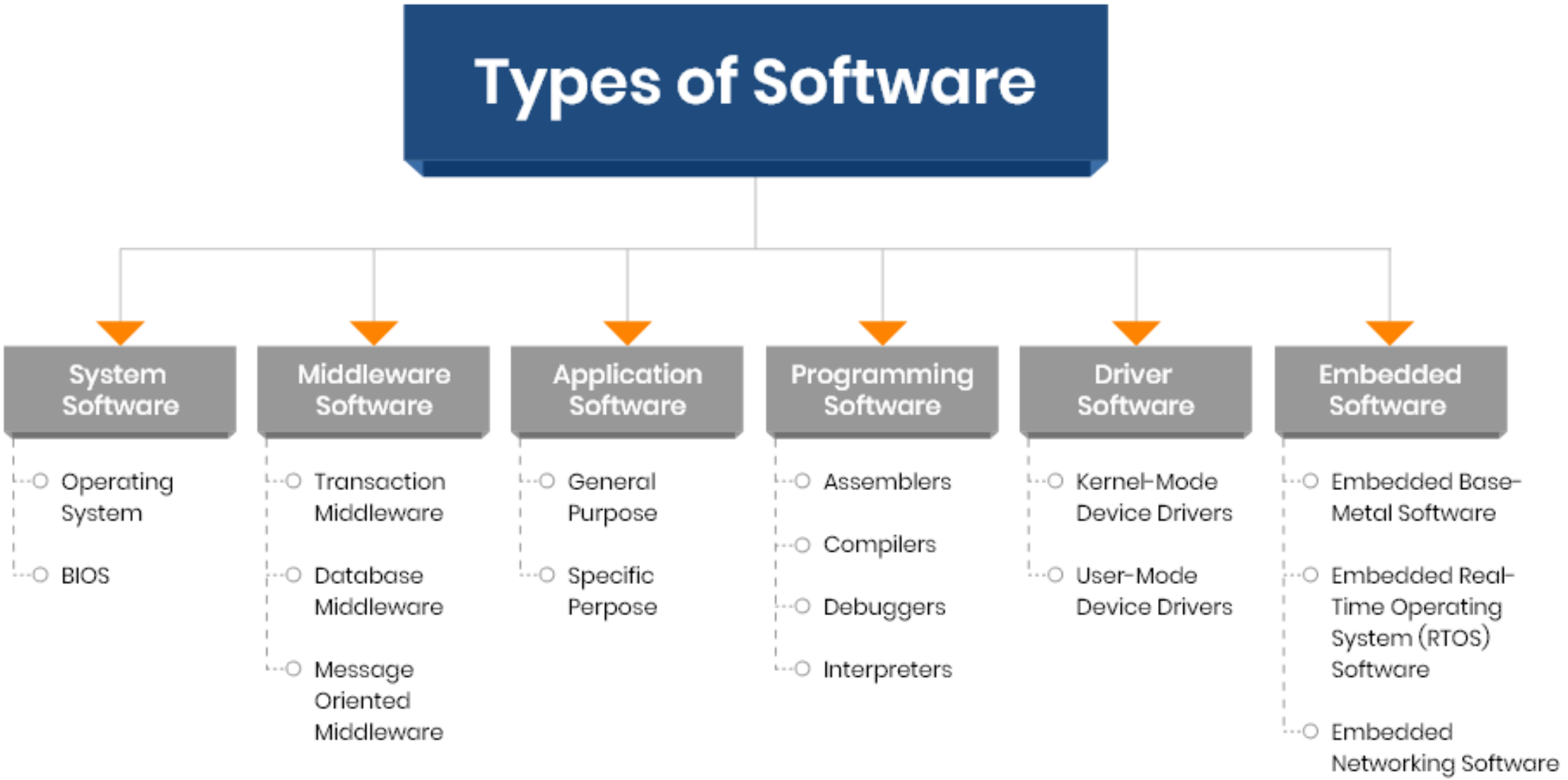
•Good software architecture design can help your organization:

- Easily scale up or down through agile, iterative updates.
- Maintain code quality, reduce errors, and find bugs.
- Modernize legacy systems, reduce audit and IT costs, and build innovative new customer experiences.
- Streamline development through reusability and automation, in turn saving time and boosting productivity.
- Reduce costs, speed up time to market, and boost customer satisfaction.
- Optimize the overall performance of your software.

DESIGN PRINCIPLES AND BEST PRACTICES

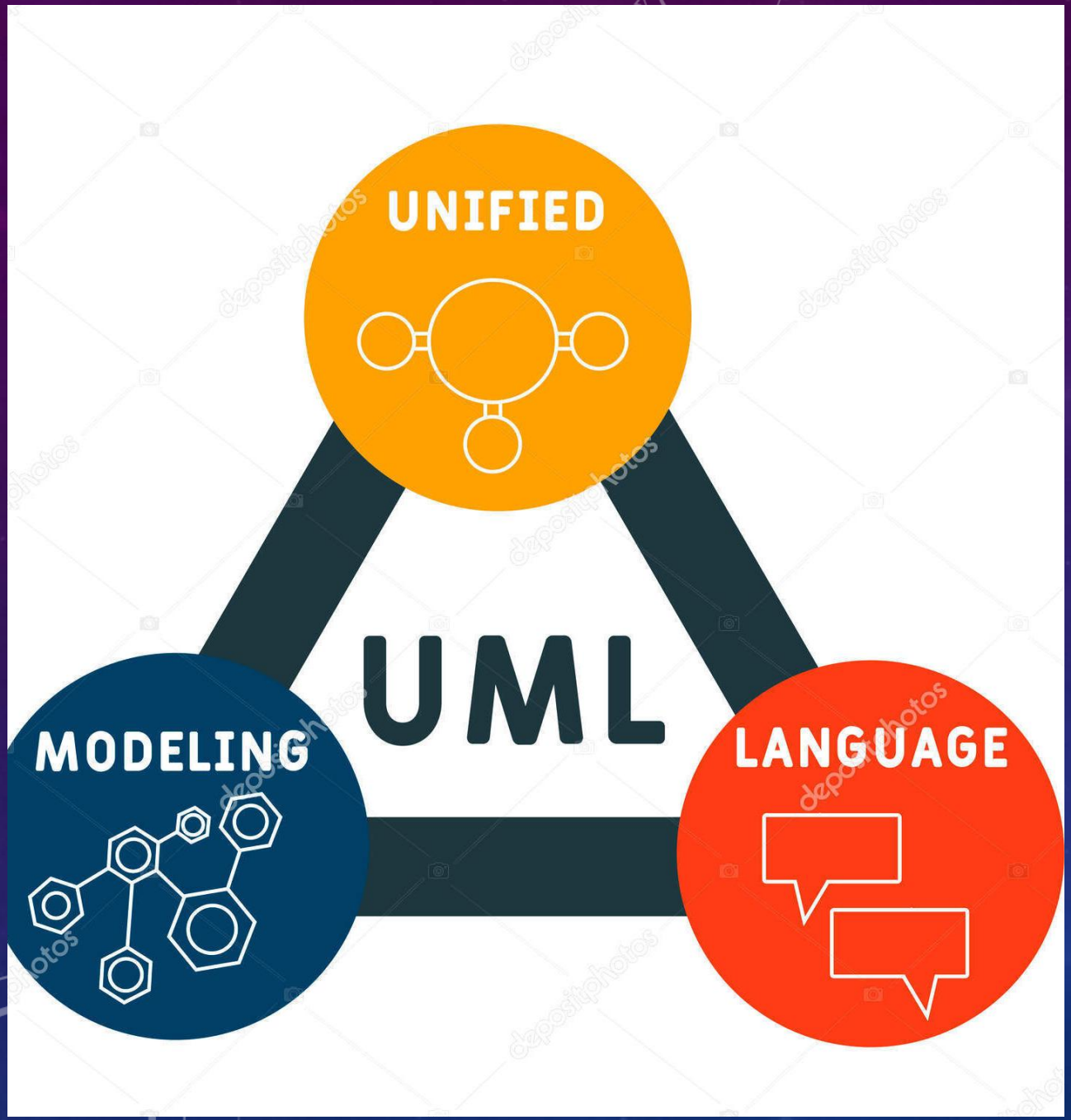
- SOLID principles
- Separation of concerns
- Modularity and encapsulation
- Design patterns

SOFTWARE AND IT'S TYPES



TOOLS AND TECHNOLOGIES FOR SOFTWARE ARCHITECTURE

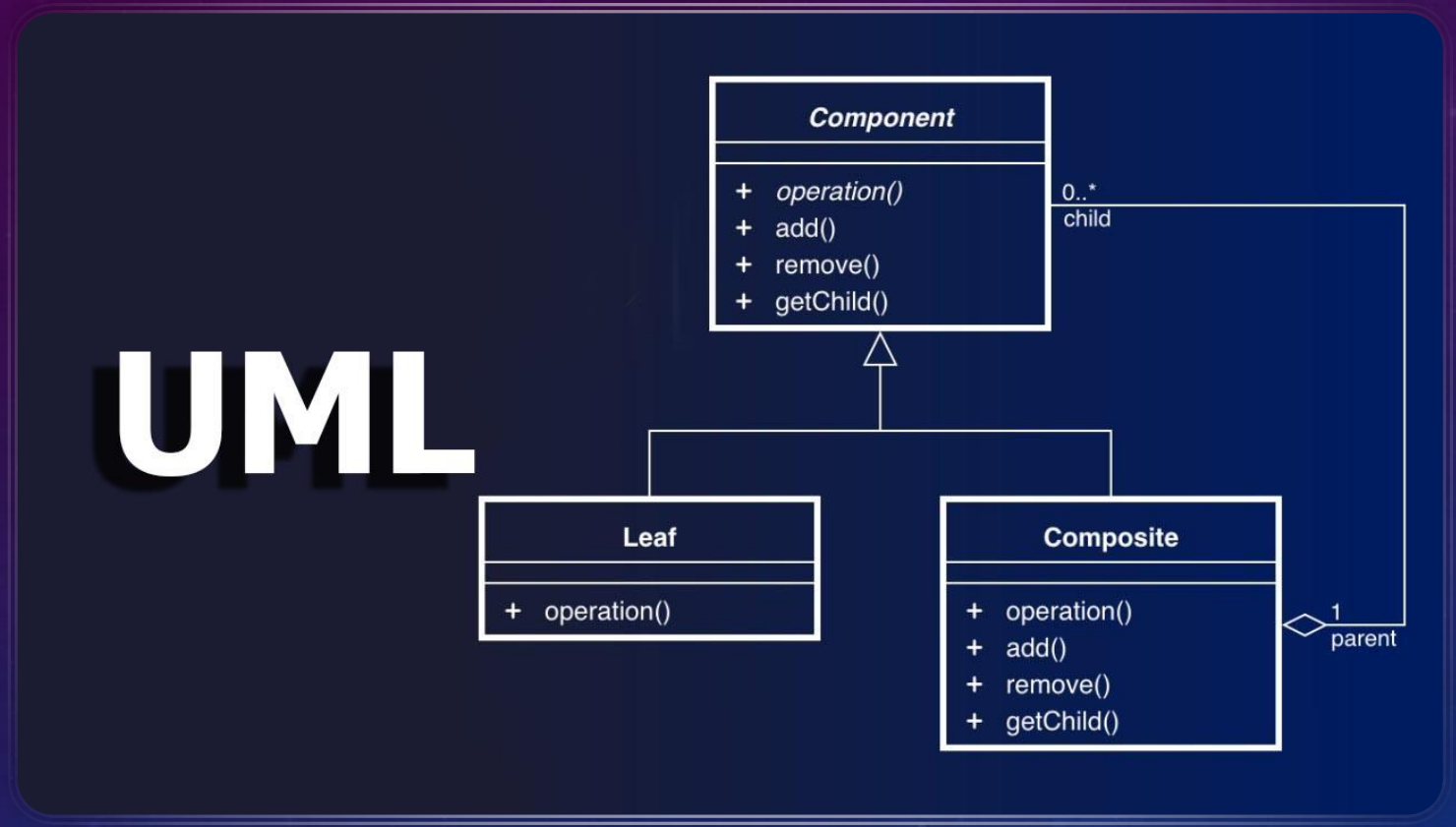
- Unified modeling language (UML)
- Architectural modeling tolos
- Version control and collaboration tools



UNIFIED MODELING LANGUAGE

UML, or Unified Modeling Language, is a st andardized graphical language for modeling , visualizing, specifying, constructing, and d ocumenting complex systems. It is widely u sed in software development to represent t he architecture, design, and behavior of sys tems. UML provides a common framework for modeling systems, which can be applied to various industries and projects. It is not a programming language but a modeling la nguage that can be used alongside any dev elopment methodology. UML has been a st andard since 2005 and is recognized by ISO as **ISO/IEC 19501:2005**.

.



UML is particularly useful for modeling complex systems, such as software applications, and is widely used in the software industry. It helps in simplifying and unifying the graphical notation used to describe the architecture and behavior of a system, making it easier for stakeholders to understand and communicate the system's design. UML is also applicable to modeling business processes and can be used in various programming paradigms, including object-oriented programming.

Software Design and Analysis

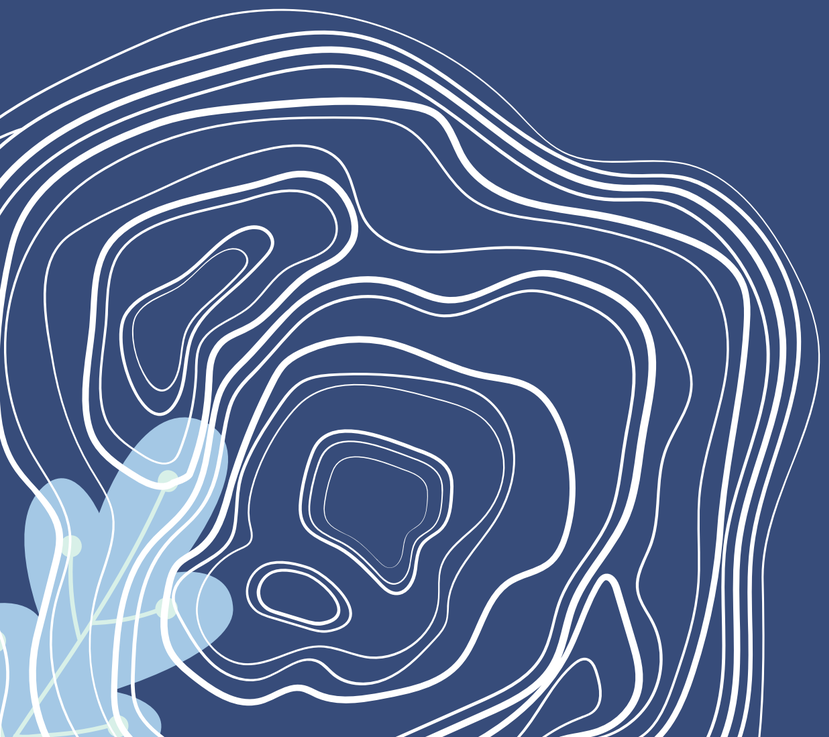
UML - USE CASE DIAGRAMS

ITIID-4

Eudaldo Ahumada
Juan Hernández
Sandra Perales

WHAT IS...?

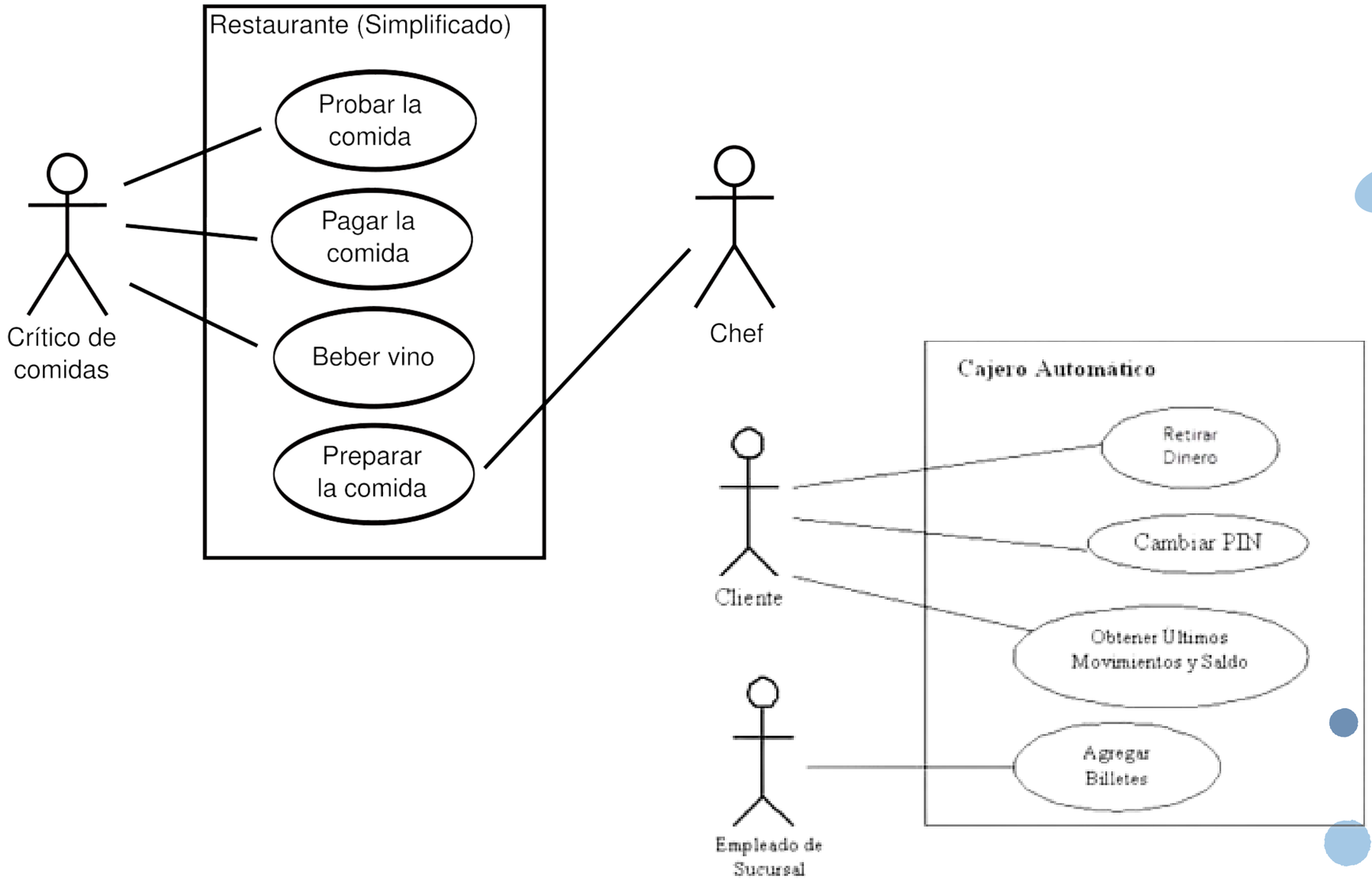
A use case diagram is a graphic representation that models the system's fucntional requirements from a user's perspective.
It shows what the system is doing and who are interacting with it (actors); it doesn't show what is doing internally.



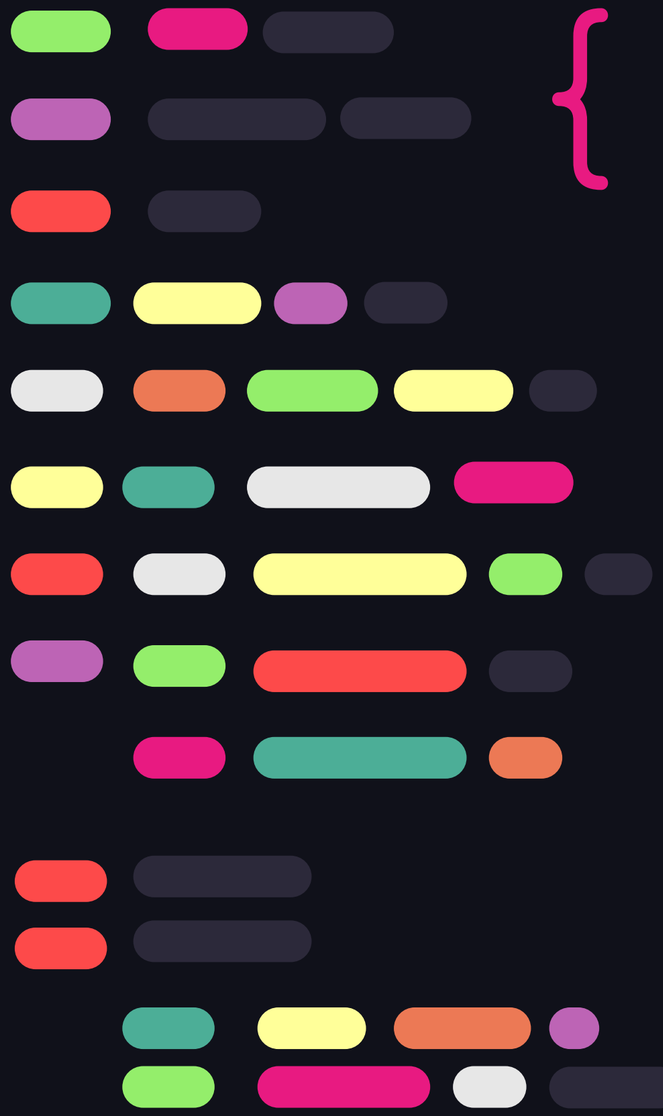
Key Components of a diagram:

- Actor(s)-. Represented by a human figure, an external entity that interacts with the system.
- System-. Represented by a square, contains the use cases.
- Use Case-. An ellipse that represents the funtionality or objective.
- Association-. The line that connects the actor to the case.

EXAMPLE


Restaurante (Simplificado)
Probar la comida
Pagar la comida
Beber vino
Preparar la comida
Crítico de comidas
Chef
Cajero Automático
Retirar Dinero
Cambiar PIN
Obtener Ultimos Movimientos y Saldo
Agregar Billetes
Cliente
Empleado de Sucursal

Thank You



SUB ESCENARIO

< Leonardo Ruben Hernandez Silva >
< Maximo Santiago Romero >
< Carlos Antonio Andrade Valles >

¿Qué es un sub-escenario?

Un subescenario es una ruta alternativa dentro de un caso de uso que describe qué sucede cuando el proceso no sigue el flujo principal o cuando ocurre una situación especial.

- **Permite anticipar condiciones alternativas o excepciones sin perder la coherencia con el escenario principal.**
- **Es como una “rama secundaria” que describe qué pasa si ocurre algo distinto a lo esperado.**

Funcion del sub-escenario:

La función de un sub-escenario es manejar situaciones diferentes al flujo normal del sistema. Permite que el sistema sepa qué hacer cuando ocurre un error, una decisión distinta del usuario o una condición especial, evitando que el proceso falle. Gracias al sub-escenario, el sistema puede mostrar mensajes, corregir acciones y continuar o finalizar el proceso de forma controlada.

Ejemplo: Iniciar sesion

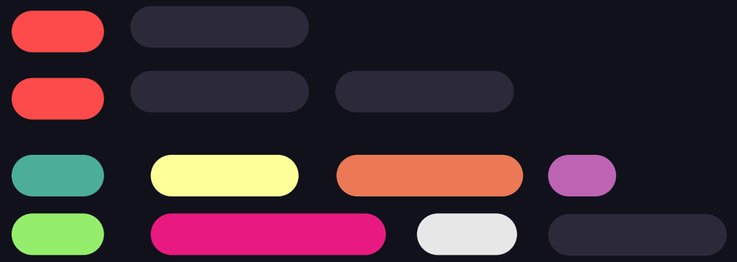
Caso de uso: Iniciar sesión

Escenario principal:

El usuario ingresa su correo y contraseña.

El sistema valida los datos.

El sistema permite el acceso.



Sub-escenario 1 (alternativo):

2.1 El usuario ingresa una contraseña incorrecta.

2.2 El sistema muestra un mensaje de error.

Sub-escenario 2 (alternativo):

2.1 El usuario deja campos vacíos.

2.2 El sistema solicita completar la información.

Diagramas de Secuencia

Our Member

Andrey Mares

Angel Serra

Níz Peña

¿Qué es?

Un diagrama de secuencia es un tipo de diagrama UML que muestra cómo interactúan los actores y objetos de un sistema, indicando el orden de los mensajes en el tiempo para realizar una acción.

Objetivo

- Representar la comunicación entre componentes
- Mostrar el flujo lógico de un proceso
- Entender cómo funciona una funcionalidad paso a paso

Elementos Fundamentales

01 actor

Entidad externa que inicia o participa en la interacción
Ejemplo: Usuario, Sistema externo

02 Objeto

Instancia de una clase que participa en la interacción
Ejemplo: Controlador, Servicio, Base de Datos

03 Línea de Vida

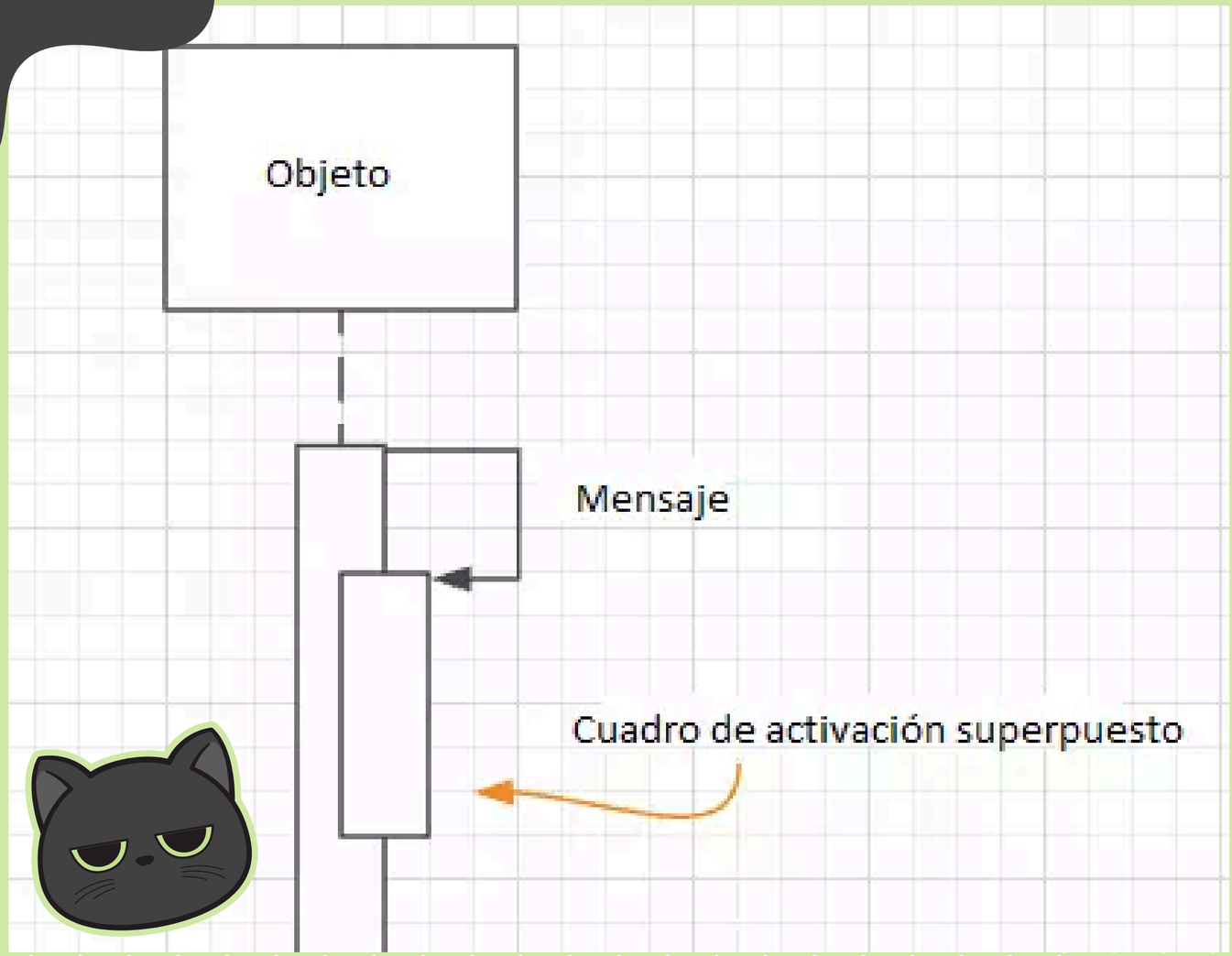
Línea vertical que representa el tiempo de existencia del actor u objeto durante la interacción

04 Mensajes

Comunicación entre actores u objetos
Tipos:

- Sincrónicos: el emisor espera respuesta
- Asincrónicos: el emisor no espera respuesta
- Retorno: respuesta a un mensaje previo

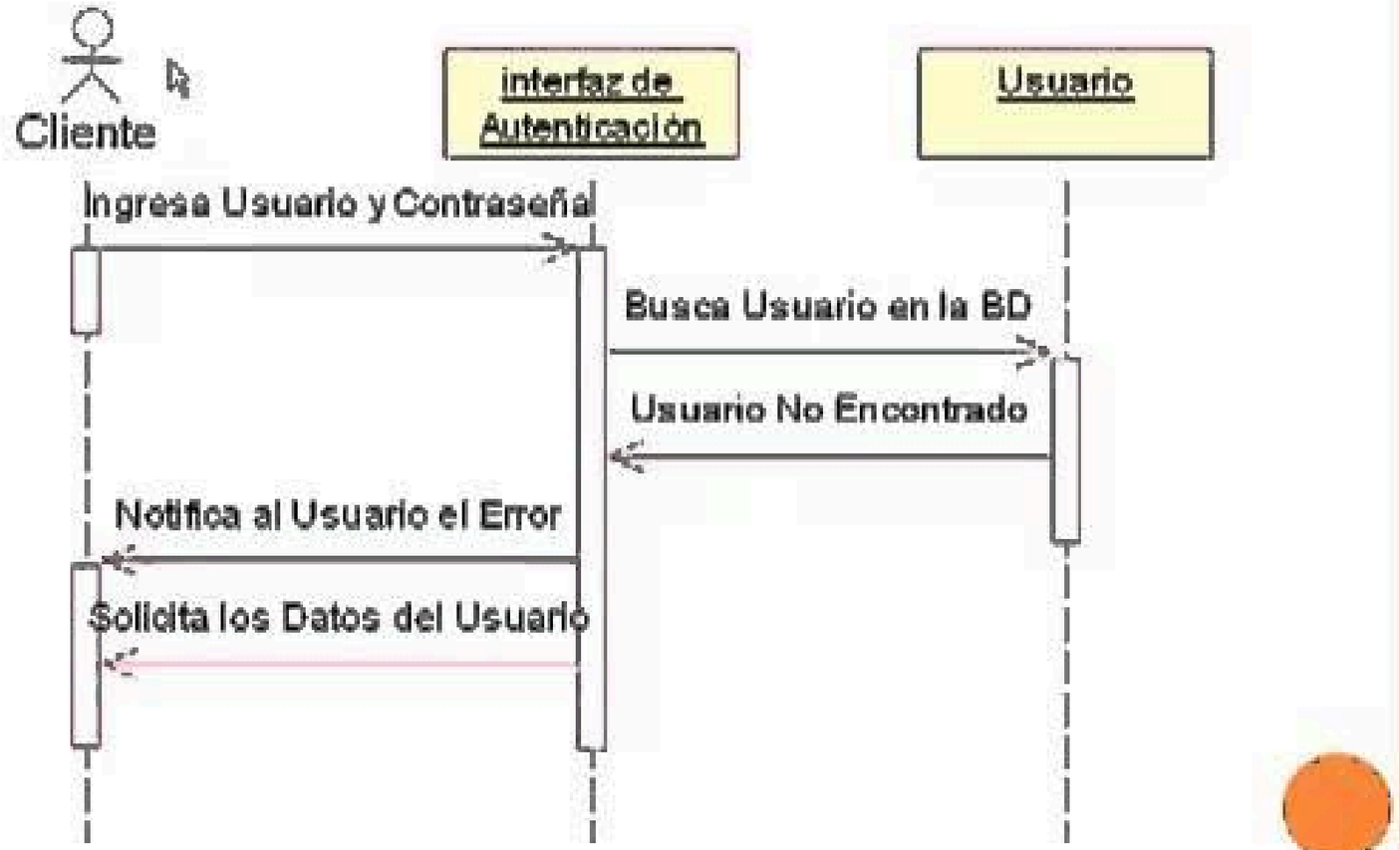
Elementos Fundamentales



05 Activación

La activación representa el tiempo durante el cual un objeto está ejecutando una operación como resultado de un mensaje recibido. Se dibuja como un rectángulo vertical delgado sobre la línea de vida del objeto.

Diagrama de Secuencia



Ejemplo

Process

01

El usuario envía una solicitud.

02

El sistema valida la información.

03

Se consulta la Base de Datos.

04

Se devuelve una respuesta.

Conclusion

El diagrama de secuencia representa la interacción temporal entre los componentes de un sistema, mostrando los procesos que se ejecutan, el intercambio de datos entre objetos y el flujo de control, lo que permite analizar y validar el funcionamiento del sistema antes de su implementación.

Thank You

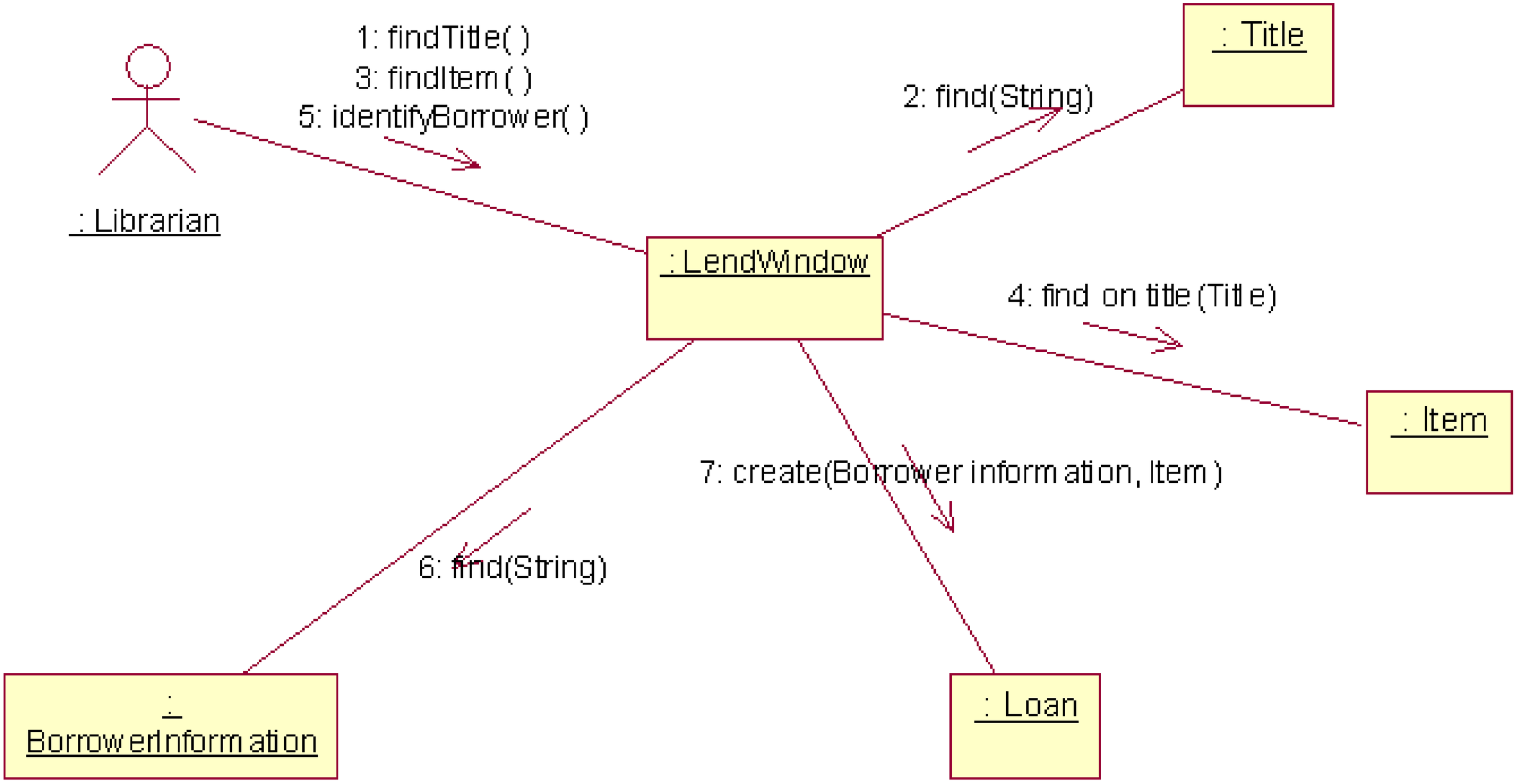
DIAGRAM COLLABORATION

IS A UML DIAGRAM THAT SHOWS HOW OBJECTS COLLABORATE BY EXCHANGING MESSAGES, EMPHASIZING THE STRUCTURAL RELATIONSHIPS AND LINKS BETWEEN THEM.

IT VISUALIZES MESSAGE FLOW USING OBJECTS, CONNECTORS, AND NUMBERED MESSAGES TO INDICATE ORDER, ILLUSTRATING THE DYNAMIC BEHAVIOR AND OBJECT ROLES WITHIN A SPECIFIC SCENARIO.

ITS MAIN PURPOSE IS TO SHOW HOW A SET OF OBJECTS COLLABORATES TO REALIZE A PARTICULAR BEHAVIOR OR OPERATION.

Abdiel Josue Pacheco Robles
George Mena Camacho
Maria José Vazquez Romano


1: findTitle( )
3: findItem( )
5: identifyBorrower( )
: Librarian
: Title
2: find(String)
: LendWindow
4: find on title(Title)
: Item
7: create(Borrower information, Item)
6: find(String)
:
BorrowerInformation
: Loan

DIAGRAMAS DE
CLASE PURA

INTRODUCCIÓN

En un diagrama de clases se muestran las relaciones entre clases para entender cómo interactúan.

Estas relaciones incluyen la asociación, donde una clase usa a otra; la agregación, que indica una relación de “tiene un” con independencia entre clases; la composición, que es una relación más fuerte donde una clase depende totalmente de otra; la herencia, en la que una clase hija obtiene atributos y métodos de una clase padre; y la dependencia, que ocurre cuando una clase usa temporalmente a otra.

Todas estas relaciones se representan mediante líneas y símbolos específicos en UML.

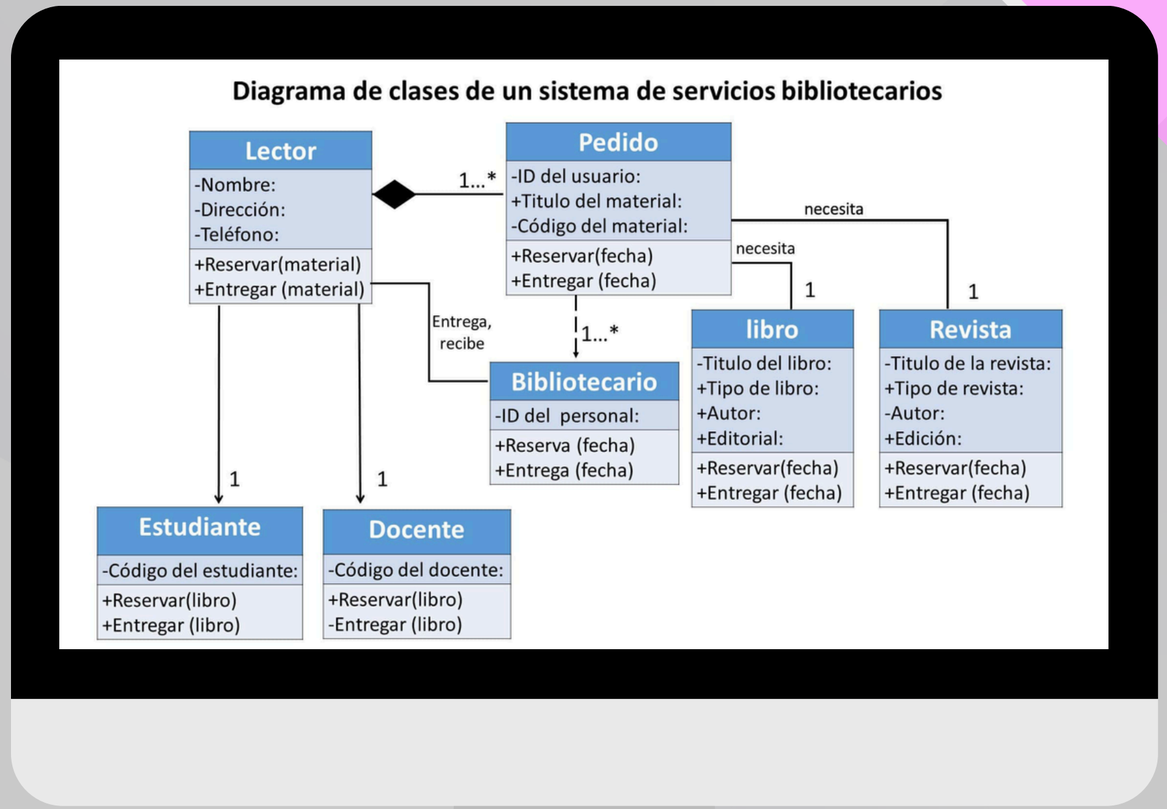
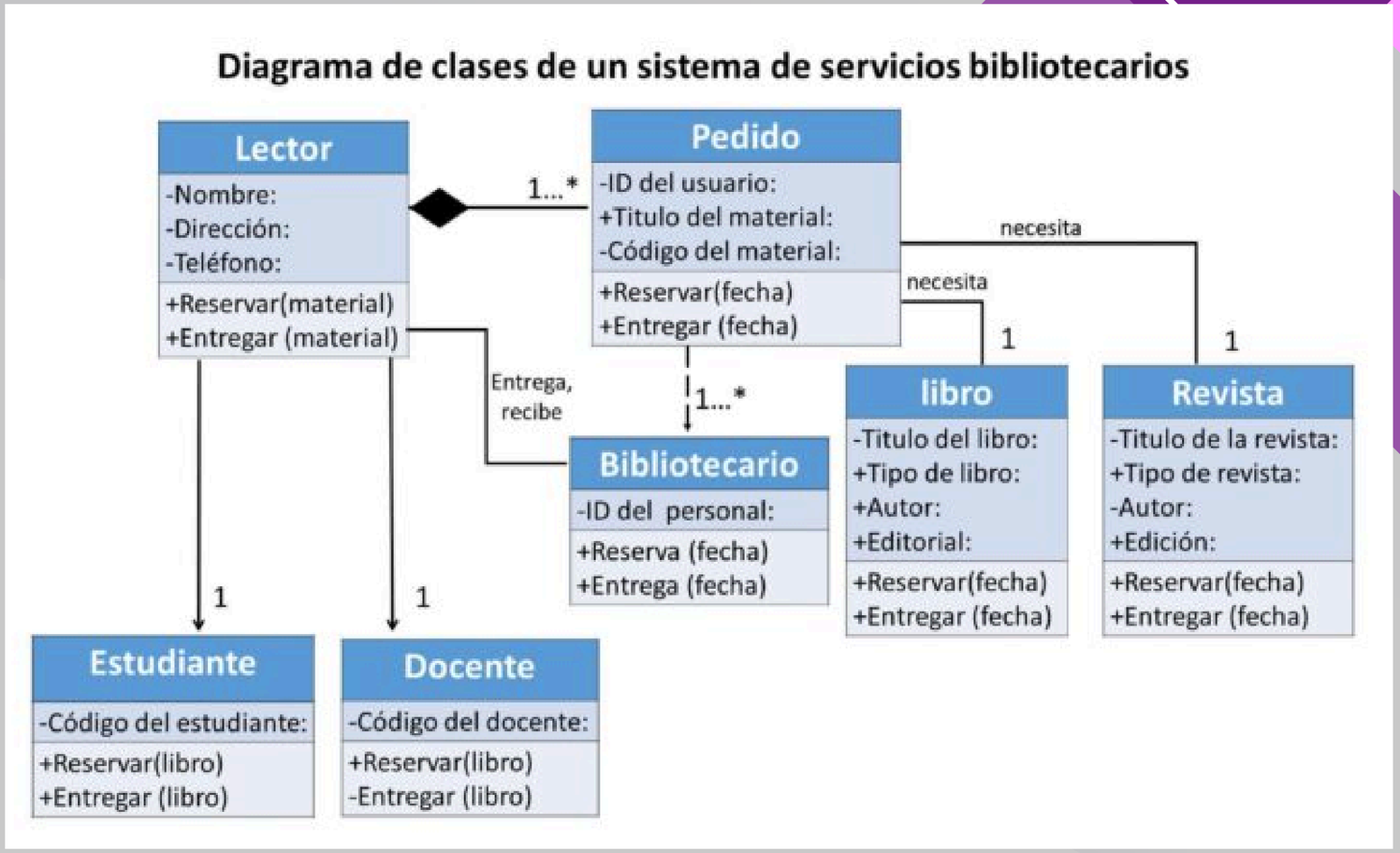


DIAGRAMA DE CLASE

Una clase es una plantilla que define las características y el comportamiento de los objetos. Las características se representan mediante atributos, que almacenan información, y el comportamiento mediante métodos, que son las acciones que la clase puede realizar.

En un diagrama de clases, una clase se dibuja como un rectángulo dividido en tres partes: nombre, atributos y métodos.



RELACIONES ENTRE CLASES

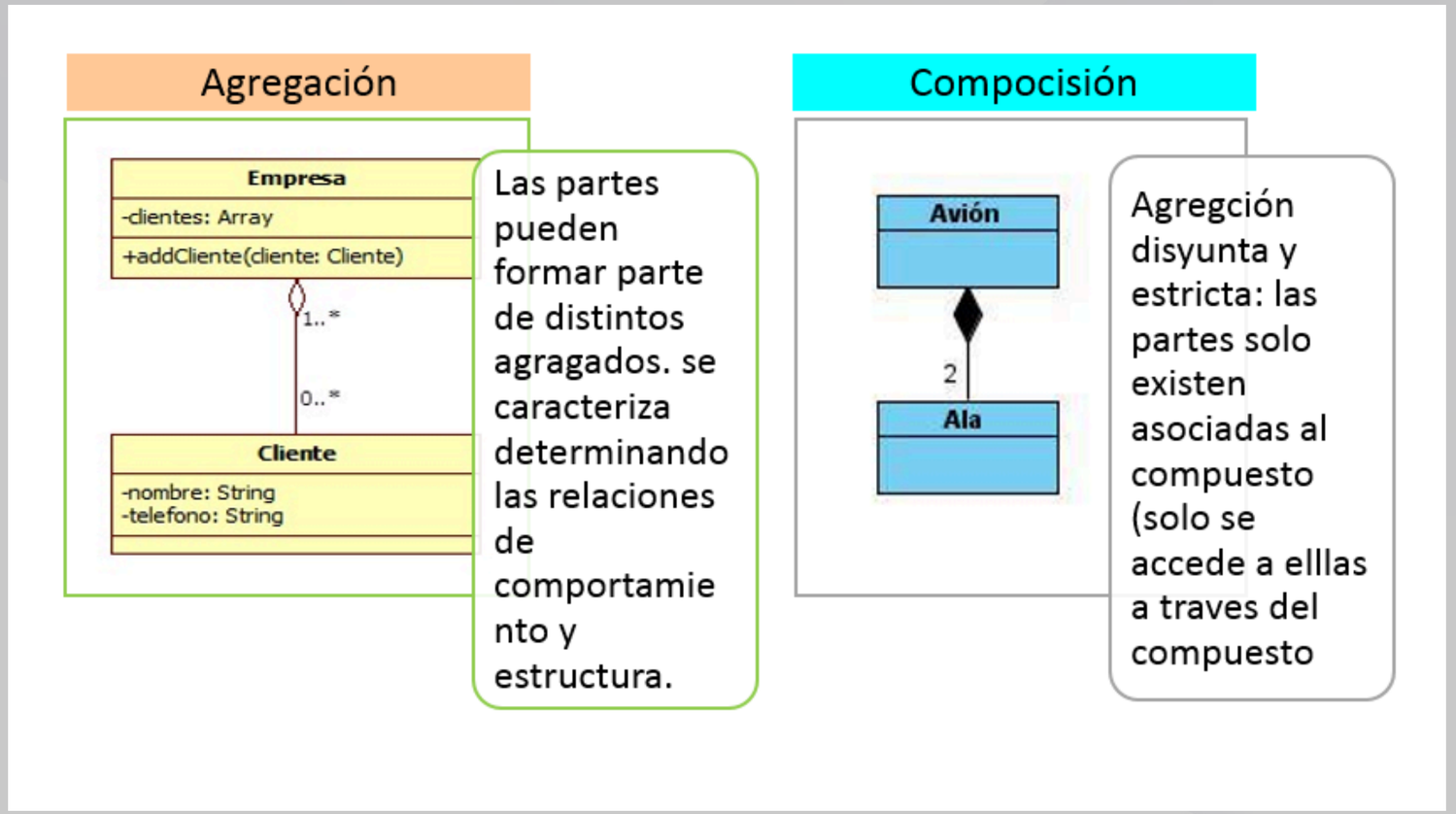
Las relaciones permiten comprender cómo interactúan las clases entre sí.

La asociación indica que una clase usa a otra. La agregación representa una relación de “tiene un”, donde las clases pueden existir de manera independiente.

La composición es una relación más fuerte, donde una clase depende completamente de otra para existir.

La dependencia ocurre cuando una clase necesita temporalmente a otra para realizar una acción.

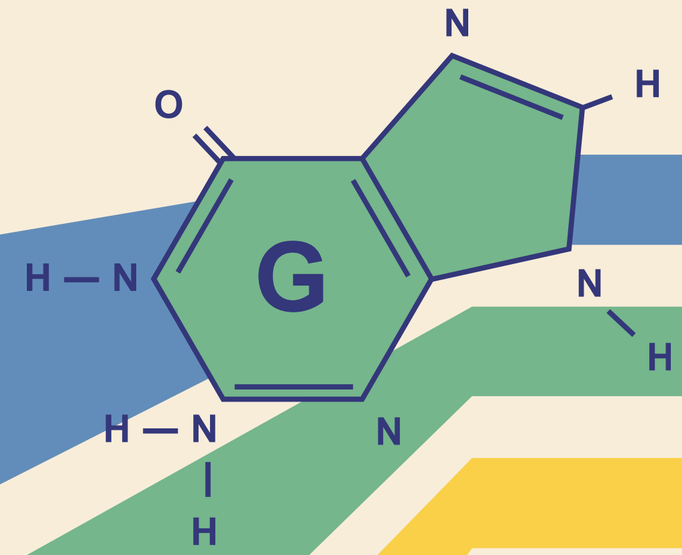
Estas relaciones se representan mediante líneas y símbolos propios del UML.

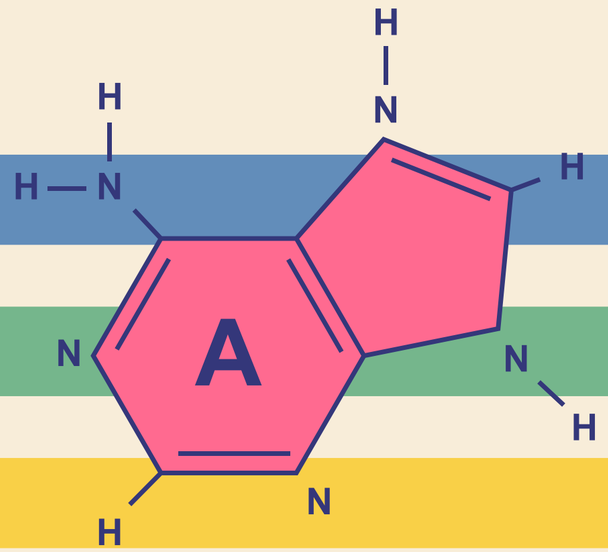


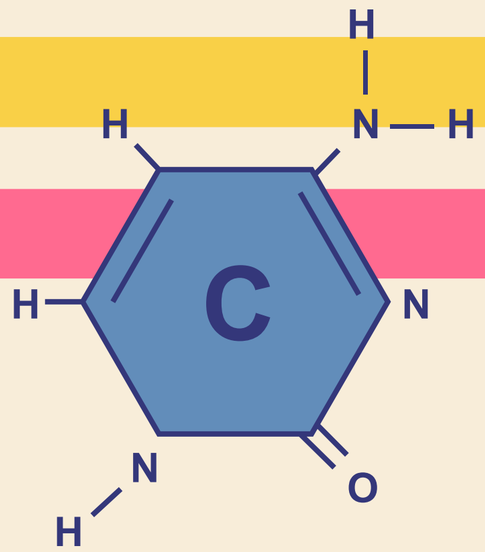
MUCHAS
GRACIAS

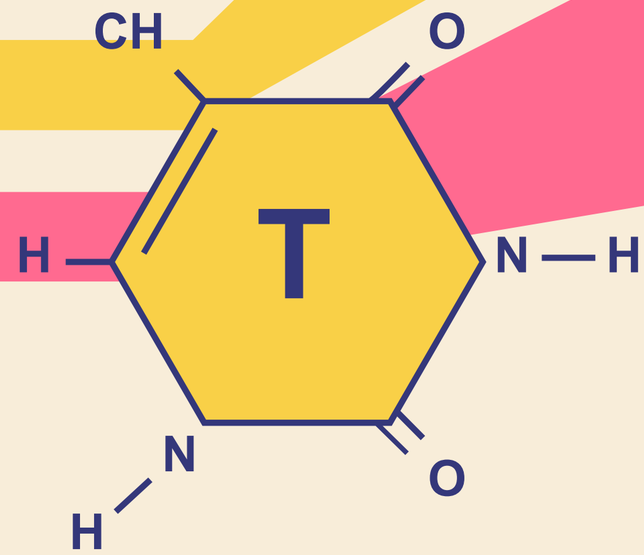
Diagramas de clase:

Herencia.









Temario

Diagramas de clase

Herencia

Equipo

Aguilar Uriel

Avalos Alexis

Valdivia Leopoldo

Diagrama de Clase

Es un tipo de diagrama del lenguaje UML que se utiliza para representar de manera gráfica la estructura de un sistema orientado a objetos.

En este diagrama se muestran las clases que forman parte del sistema, sus atributos, sus métodos y las relaciones que existen entre ellas. Su función principal es ayudar a comprender cómo está organizado un programa antes de comenzar a programarlo.

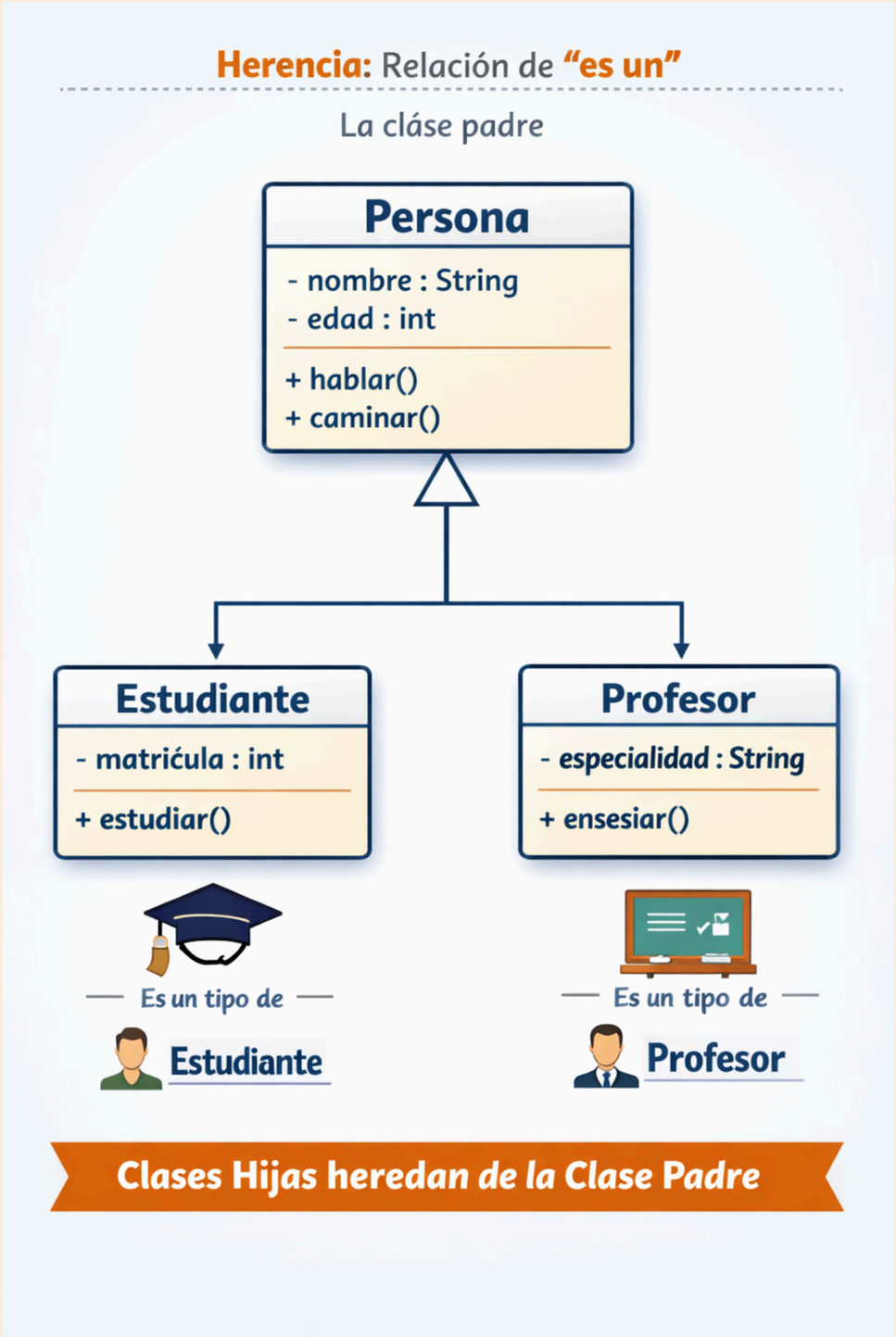
Herencia

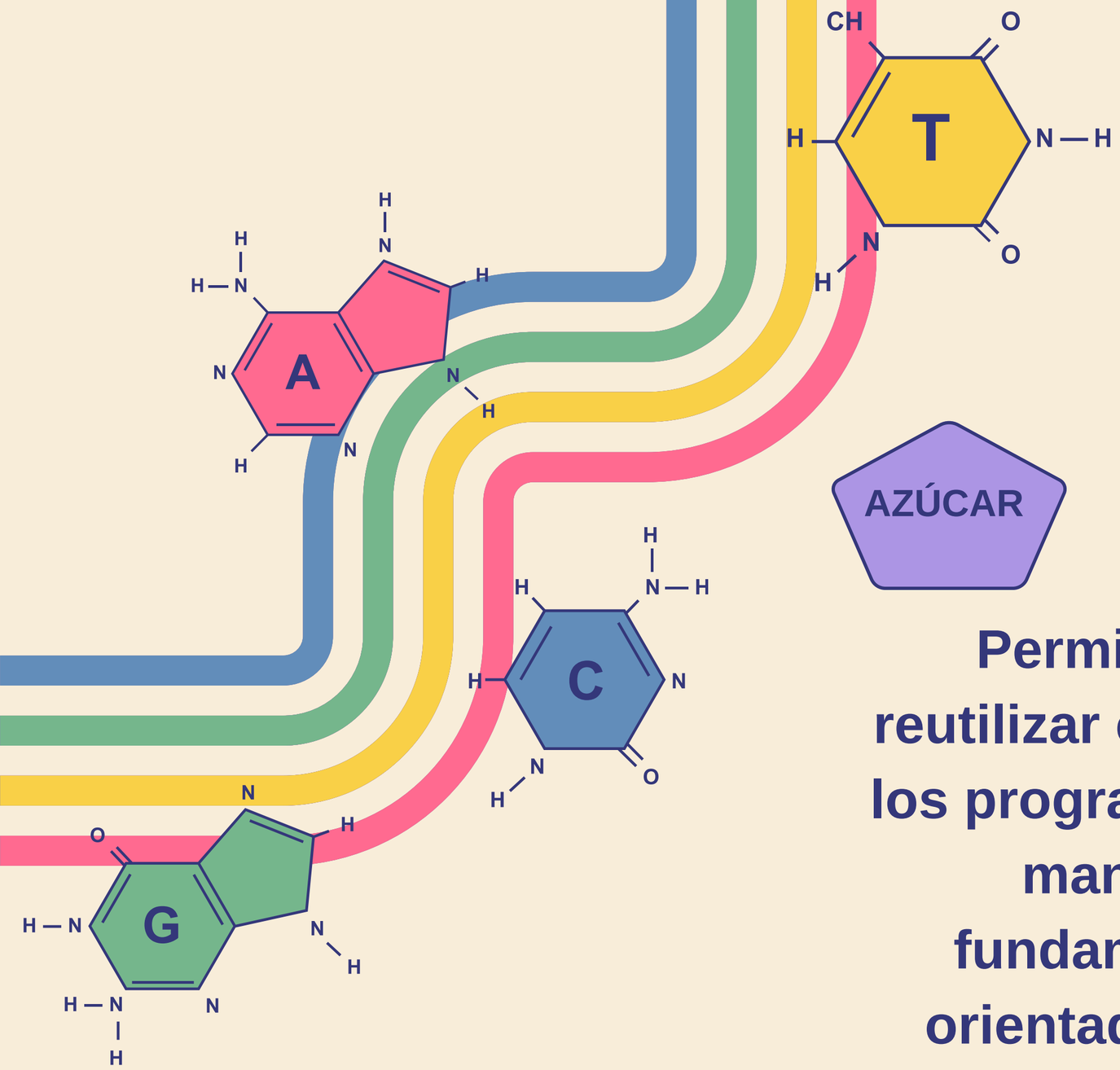
La herencia es una de las relaciones más importantes dentro de un diagrama de clases. Consiste en que una clase puede heredar los atributos y métodos de otra clase. La clase de la que se hereda se llama clase padre o superclase , y la clase que recibe esa herencia se llama clase hija o subclase . Gracias a la herencia, no es necesario repetir el código, ya que las clases hijas pueden reutilizar todo lo que está definido en la clase padre.

Herencia

En un diagrama de clases, la herencia se representa mediante una línea con una flecha triangular vacía que va desde la clase hija hacia la clase padre.

Esto indica que la clase hija es un tipo más específico de la clase padre. Por ejemplo, si existe una clase llamada “Persona”, de ella pueden heredar clases como “Estudiante” o “Profesor”, ya que ambos son tipos de persona y comparten características en común.





F

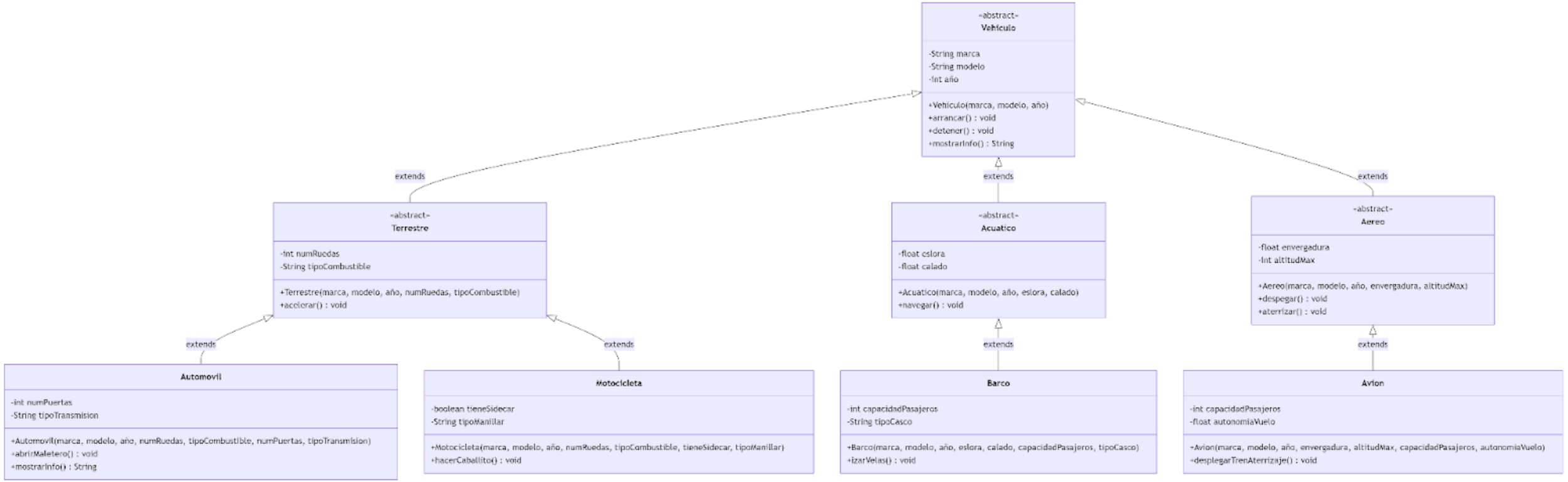
Herencia

Permite organizar mejor los sistemas, reutilizar códigos, reducir errores y hacer que los programas sean más fáciles de entender y mantener. Por eso es un concepto fundamental dentro de la programación orientada a objetos y del diseño mediante diagramas de clases.

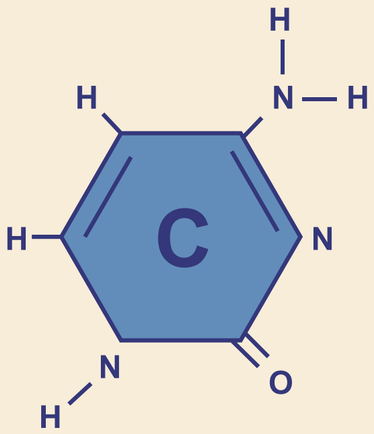
Uso de la Herencia

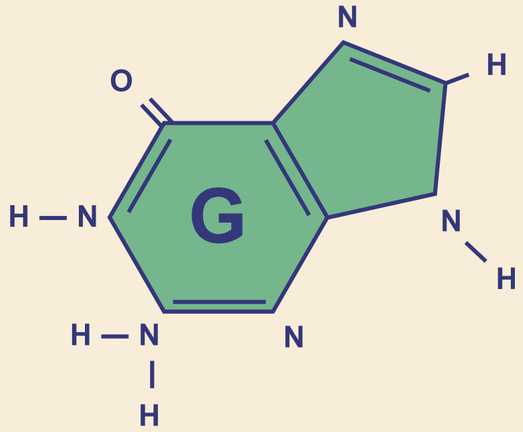
El uso de la herencia ofrece múltiples ventajas, como la reducción de código duplicado, la facilidad de mantenimiento y una mejor representación de la realidad en los sistemas informáticos. Existen diferentes tipos de herencia, como la simple, jerárquica y multinivel, dependiendo de cómo se relacionen las clases entre sí. Sin embargo, es importante utilizarla de forma adecuada y solo cuando exista una relación lógica de tipo “es un”, ya que un mal uso puede generar sistemas confusos y difíciles de mantener. En conclusión, los diagramas de clases y la herencia son elementos fundamentales para el diseño correcto de software orientado a objetos.

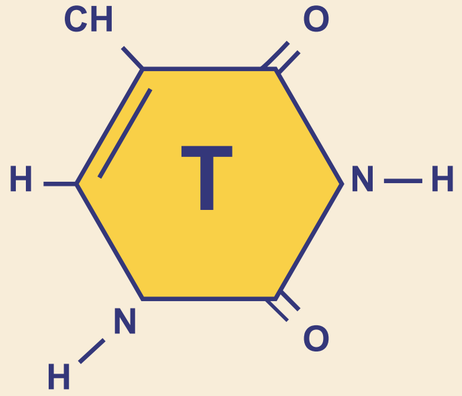
Ejemplo



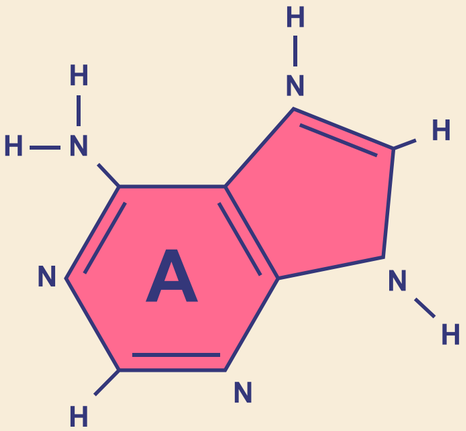
¡Y ahora un Video!







¡Gracias!



Diagrama

CLASES DE ATRIBUTOS

Santiago Pacheco - Hernández Cortes - Gutiérrez Hernandez

Diagrama de clases de un sistema de servicios bibliotecarios

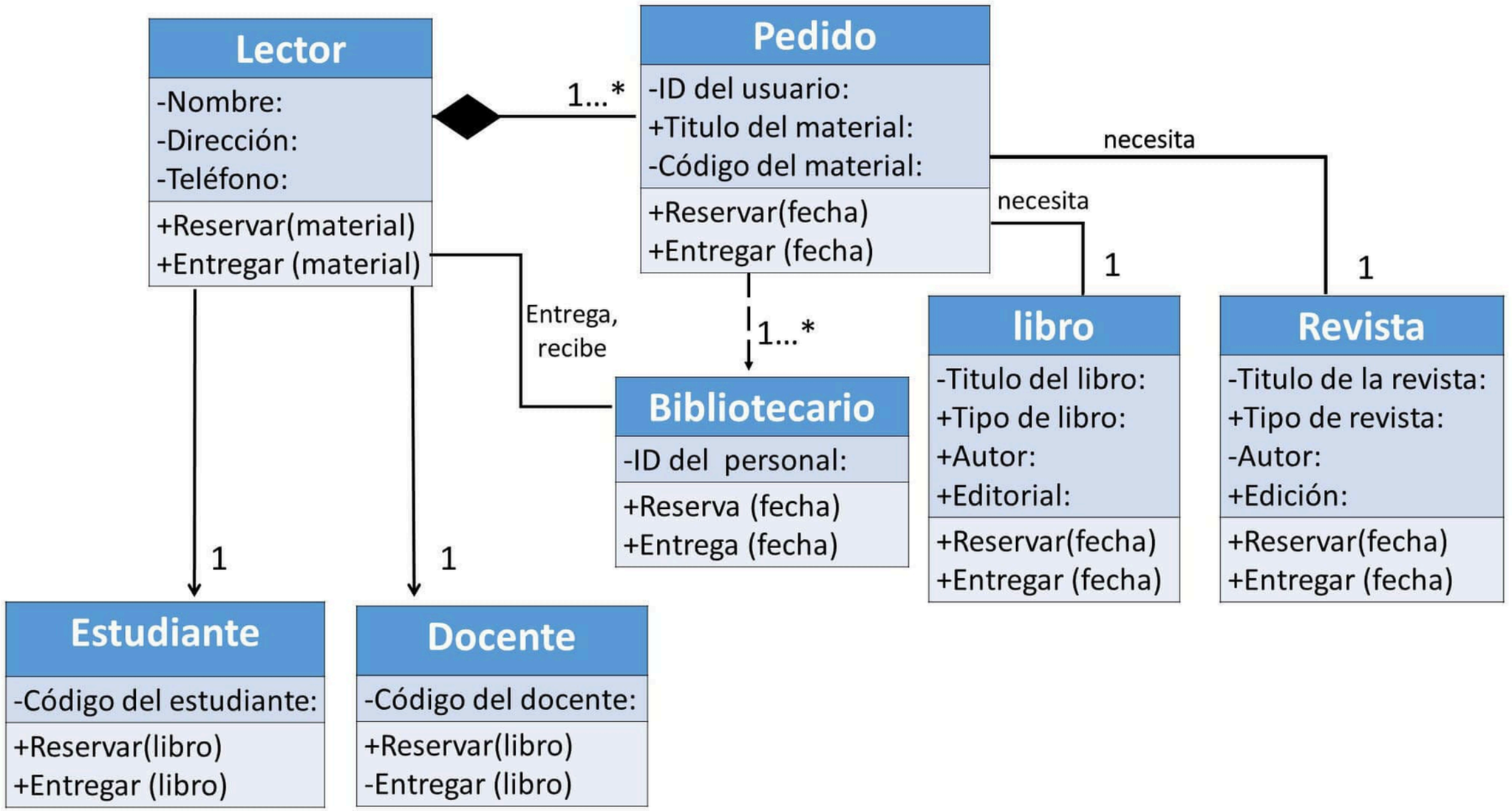


DIAGRAMA DE CLASE

2026

OPERACIONES

Diego Velasco
Fernando Plantillas
Luis Avila

¿Qué son?

Son los servicios o funciones que una clase proporciona, es decir, las acciones que sus objetos pueden realizar.
Se listan en la parte inferior de la representación rectangular de la clase, debajo de los atributos.

Como funcionan?

Las operaciones indican las acciones que una clase es capaz de realizar a través de sus métodos. Estas se muestran en la parte inferior del diagrama de la clase, debajo de los atributos como ya mencionado anteriormente.

Ejemplo