

# La fase de análisis

Antes de escribir una sola línea de código, es necesario comprender qué necesita el cliente o usuario. Esta tarea forma parte de la fase de análisis, en la que se estudian los requisitos del sistema y se definen sus comportamientos. Aquí se representa lo que debe hacer el sistema, no cómo se va a hacer.

En esta unidad aprenderemos a utilizar herramientas y técnicas de análisis, como:

- Casos de uso y diagramas de comportamiento, para describir cómo interactúan los usuarios con el sistema
- Diagramas de actividades, para representar flujos de trabajo
- Diagramas de estados, para mostrar cómo se comporta el sistema ante distintos eventos

## Actividad

Reflexiona sobre la siguiente pregunta:

¿Por qué crees que es un error empezar a programar sin haber analizado antes lo que se necesita?

Después, debate en clase los siguientes puntos:

- ¿Qué consecuencias puede tener un mal análisis de requisitos?
- ¿En qué se diferencia “saber lo que el cliente quiere” de “suponer lo que quiere”?
- ¿Qué ventajas tiene representar visualmente el comportamiento del sistema?
- ¿Crees que los diagramas ayudan solo al analista, o también al equipo de desarrollo? ¿Por qué?

# Especificación de requisitos del sistema

- Especificación de requisitos del sistema
  - Introducción al análisis de sistemas
  - Especificación de requisitos de sistemas
  - El estándar IEEE 830 para representar la especificación de requisitos de un sistema

## Introducción al análisis de sistemas

El análisis de sistemas es una fase del desarrollo de software que consiste en el proceso mediante el cual se estudian y descomponen los componentes de un sistema para entender su funcionamiento, identificar problemas y encontrar soluciones eficaces. Este proceso se utiliza en ingeniería, informática, administración y muchas otras áreas para desarrollar o mejorar sistemas complejos. Entre sus utilidades, encontramos las siguientes:

1. **Identificación de problemas:** Permite detectar cuellos de botella, errores o fallos en un sistema.
2. **Optimización de procesos:** Ayuda a rediseñar procesos para mejorar su eficiencia.
3. **Diseño de nuevos sistemas:** Se utiliza como paso inicial para construir sistemas nuevos y bien estructurados.

El análisis de sistemas es fundamental en el desarrollo de software, ya que proporciona una visión clara de los requisitos del sistema antes de pasar al diseño y la programación.

## Especificación de requisitos de sistemas

La **Especificación de Requisitos del Sistema (ERS)** es un documento que define **qué hará el sistema** y **cómo interactuará con su entorno**, sin entrar en detalles técnicos de implementación. Su propósito es establecer una visión compartida entre los stakeholders (clientes, desarrolladores, testers, etc.). Una ERS es esencial en el desarrollo de software debido a los siguientes puntos:

1. **Proporciona claridad:** Ayuda a evitar malentendidos durante el desarrollo.
2. **Facilita el diseño y las pruebas:** Los desarrolladores y testers lo usan como base para su trabajo.
3. **Control de alcance:** Ayuda a gestionar cambios en los requisitos.

Un buen SRS cumplir con las siguientes características:

- **Compleción:** Cubre todos los requisitos del sistema.
- **Consistencia:** Evita contradicciones internas.
- **Estructuración:** Fácil de entender y navegar.

# El estándar IEEE 830 para representar la especificación de requisitos de un sistema

Según la norma IEEE 830, la especificación de requisitos de un sistema consta de las siguientes partes:

- **1. Introducción**
  - **1.1. Propósito:** Se especifica el propósito del documento y a quién va dirigido.
  - **1.2. Ámbito del sistema:** Se da nombre al sistema, se explica qué hace y qué no hace, los beneficios, objetivos y metas que se espera alcanzar y se referencian todos los documentos de nivel superior del sistema (por ejemplo, si el sistema pertenece a un sistema más grande se referencia ese documento).
  - **1.3. Definiciones, acrónimos y abreviaturas**
  - **1.4. Referencias:** En esta sección se listan los documentos referenciados.
  - **1.5. Visión general del documento:** Se describen los contenidos de el documento de forma breve.
- **2. Descripción general:** En esta sección se describe el contexto del sistema.
  - **2.1. Perspectiva del producto:** Se relaciona el sistema con otros productos.
  - **2.2. Funciones del producto:** Se resume a grandes rasgos la funcionalidad del futuro sistema.
  - **2.3. Características de los usuarios:** Se describen las características de los usuarios del sistema.
  - **2.4. Restricciones:** Se listan las limitaciones sobre los desarrolladores, ya sean por parte de la empresa, limitaciones del hardware, etc.
  - **2.5. Suposiciones y dependencias:** Se describen factores que, si cambian, podrían afectar a los requisitos.
  - **2.6. Requisitos futuros:** Se esbozan posibles mejoras para el futuro.
- **3. Requisitos específicos:** Esta es la sección más importante, donde se explica en detalle cómo funciona el sistema. Es la sección que vamos a trabajar en clase.
  - **3.1. Interfaces externas:** Se describen los requisitos externos al sistema, por ejemplo, como se comunica con bases de datos, interfaces de usuarios, otros sistemas, etc.
  - **3.2. Funciones o Requisitos funcionales:** Es la parte más importante del documento y se detalla aquellas acciones que debe hacer el sistema.

- **3.3. Requisitos de Rendimiento o no funcionales:** Se detallan los requisitos relacionados con el rendimiento del sistema.
- **3.4. Restricciones de diseño:** Se describe todo aquello que restrinja las decisiones relativas al diseño de la aplicación.
- **3.5. Atributos de sistema:** Se detallan atributos de calidad del sistema, como la fiabilidad, mantenibilidad, seguridad, etc.
- **3.6. Otros requisitos:** Se incluye cualquier otro requisito que no tenga cabida en los puntos anteriores
- **4. Apéndices:** Se incluye cualquier información relevante para el ERS pero que no forme parte del ERS.

Los requisitos deben estar de escritos de forma correcta, no ser ambiguos y completos. Además, los requisitos deben de ser consistentes (no contradictorios) y estar clasificados por importancia. Deben de ser verificables y modificables de forma sencillo. Además, deben de ser trazables (es decir, saber el origen de cada requisito y saber a qué componentes del sistema afecta).

Por su parte, los requisitos funcionales se pueden organizar por tipos de usuario, por objetos del sistema, por objetivos, por estímulos (entradas de datos que recibe el sistema) o por jerarquía funcional.

El estándar permite representar los requisitos empleando diagramas para facilitar su comprensión.

### Actividad

Determina los requerimientos funcionales a partir de la siguiente definición:

Una máquina de café automática gestiona la preparación y entrega de bebidas calientes. En su funcionamiento, la máquina puede encontrarse en varios estados. Al inicio, la máquina está a la espera de que un usuario realice una selección. Cuando el usuario elige una bebida (como café, té o chocolate caliente), la máquina pasa a un proceso de preparación específico según la bebida seleccionada. Si el proceso de preparación concluye sin problemas, la máquina entrega el producto al usuario y regresa al estado inicial para una nueva solicitud.

Sin embargo, si durante la preparación ocurre un error, como falta de ingredientes o una avería técnica, la máquina debe entrar en un estado de error que bloquea nuevas solicitudes hasta que un técnico de mantenimiento intervenga. El técnico podrá reiniciar el sistema y devolver la máquina a su estado inicial. Además, el proceso de **preparación** debe incluir detalles específicos dependiendo del tipo de bebida elegida: preparar café, preparar té o preparar chocolate caliente.

# Diagramas de casos de uso

## Notas sobre los diagramas

Los diagramas de esta unidad y la siguiente están hechos usando plantuml, que es un lenguaje de marcado con una sintaxis similar a un lenguaje de programación. Extensiones de VS Code como Markdown Enhanced Preview permiten ver el código como diagramas directamente, pero github no, por lo que las imágenes están compiladas en el [servidor oficial de plantum](#). y [el futuro editor](#).

Algunos diagramas están hechos en mermaid.js, que sí se renderiza en github y no incluyen ejemplos de código de forma explícita, aunque puedes copiarlo directamente de la página usando el botón arriba a la derecha y copia en formato código.

- [Diagramas de casos de uso](#)
  - [Diagramas UML de Casos de Uso](#)
    - [Pasos para crear un diagrama de casos de uso](#)
  - [Relaciones avanzadas](#)
    - [Relaciones](#) <<extend>> y <<include>>
    - [Generalización](#)
  - [Historias de Usuario \(Scrum\) y Casos de Uso](#)

## Diagramas UML de Casos de Uso

Un diagrama de casos de uso especifica cómo funciona un sistema. Más concretamente, qué puede hacer cada usuario que interactúe con el sistema. Para ello, se requieren cuatro elementos principales: Los casos de uso, los actores, las relaciones y el sistema.

- El **sistema** es el programa que vamos a desarrollar, y engloba todos los casos de uso. Se representa con un recuadro con una etiqueta.
- Un **actor** es un usuario o un sistema externo que interactúa con el sistema.
- Un **caso de uso** describe cómo un actor interactúa con el sistema para lograr un objetivo. Es una forma de **especificar requisitos funcionales** de manera visual y comprensible.
- Por último, una **relación** es una conexión que se establece entre un actor y un caso de uso. También se pueden establecer relaciones especiales entre casos de uso.

En general, un diagrama de casos de uso describe qué hace nuestro sistema y quién hace las acciones del mismo, pero no cómo las hace. Gran parte de la información necesaria la podemos extraer del documento de Especificación de Requerimientos Funcionales (ERS, IEEE 830). Como norma general, cada requerimiento funcional del sistema puede traducirse como un caso de uso.

Los diagramas de casos de uso son prácticos, además, para la **planificación de las pruebas de software**, apartado que trabajaremos en la Unidad de Programación 5.

### Actividad

En los siguientes enunciados, determina qué son actores, qué son sistemas y qué son casos de uso:

1. Se nos pide desarrollar una aplicación para que los fruteros puedan indicar qué fruta necesitan comprar a los agricultores y estos puedan actualizar su inventario en función de su cosecha.
2. Para poder evaluar el retraso de los vuelos de una compañía, se nos pide desarrollar un programa donde almacenar información sobre estos, para que los directivos de la empresa puedan tener una estadística de la puntualidad.
3. Un estudiante quiere desarrollar una aplicación que le ayude a resolver los problemas de la asignatura de Entornos de Desarrollo con Inteligencia Artificial, de tal forma que le pueda indicar un enunciado y seleccionar qué tipo de información desea extraer, si la quiere visualizar en plantuml o mermaid.js y cómo de precisa es la respuesta que le ha dado la Inteligencia Artificial, para poder saber cuánto tiene que repasar los ejercicios. Se tarda más en desarrollar esto que en hacer directamente los ejercicios, pero le parecía entretenido.

## Pasos para crear un diagrama de casos de uso

Para crear un diagrama de casos de uso, nos basamos en los requisitos funcionales del SRS. De ellos, determinamos los siguientes aspectos:

### 1. Actores:

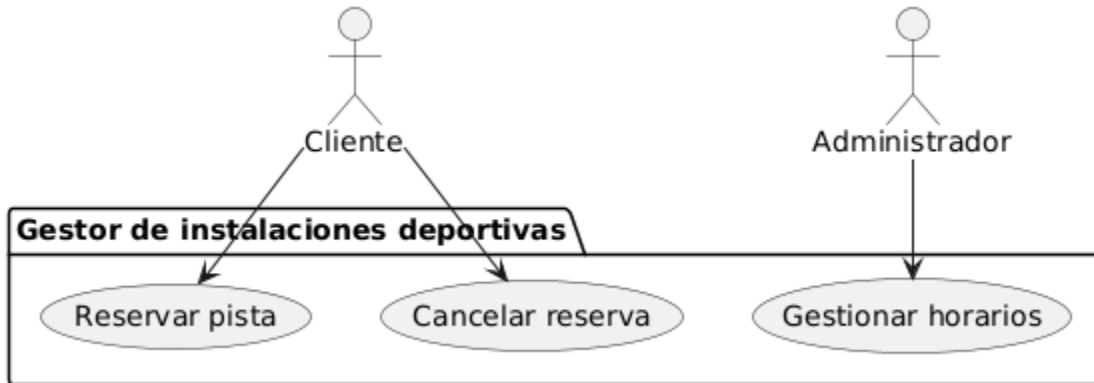
- Determinamos quién interactuará con el sistema.
- Ejemplo: En un sistema de reservas de pistas deportivas:
  - **Cliente:** Hace reservas.
  - **Administrador:** Gestiona horarios.

### 2. Casos de uso:

- Pensamos en las acciones principales que los actores quieren realizar.
- Ejemplo:

- "Reservar pista."
- "Cancelar reserva."
- "Gestionar horarios."

Con esta información, podemos finalmente crear su representación visual en forma de diagrama de casos de uso:



► [Ver el código en plantuml](#)

### Actividad

Dados los tres enunciados anteriores, crea un diagrama de casos de uso cada uno de ellos.

## Relaciones avanzadas

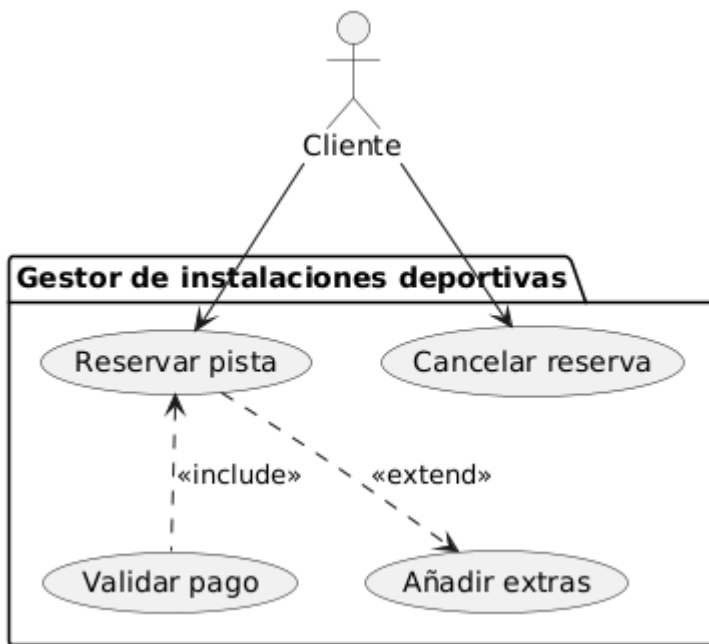
### Relaciones <<extend>> y <<include>>

Además de las relaciones básicas, los diagramas de casos de uso cuentan con relaciones avanzadas que se dan entre diferentes casos de uso. Nos sirven para dividir un comportamiento del sistema en distintos casos de uso, lo que ayuda a la modularidad y planificación del desarrollo.

La relación de tipo <<extend>> sirve para añadir una funcionalidad **opcional** que se puede lanzar después de un caso de uso. Por ejemplo, después del caso de uso `reservar pista` podríamos añadir una extensión que fuera `añadir extras` (como el alquiler de un balón). Este caso de uso podría darse o no, dependiendo de las elecciones del usuario.

La relación de tipo <<include>> sirve para añadir una funcionalidad **obligatoria** que se lanzará siempre después de un caso de uso. Por ejemplo, después del caso de uso `reservar pista` podríamos añadir una extensión que fuera `validar pago`. Esta validación se tendría que dar siempre después de cada reserva para que esta se realizara con éxito.

Las relaciones <<extend>> y <<include>> se representan así:



### ► Ver el código en plantuml

#### Actividad

En un restaurante, tenemos clientes que pueden acceder a una aplicación de reservas. Un cliente normal puede hacer una reserva y, si es necesario, indicar posibles alergias. Cuando se hace una reserva, se procesa un método de pago para disuadir a los clientes de no ir a las reservas. Por otro lado, un cliente puede cancelar su reserva siempre y cuando lo haga con una antelación de al menos 6 horas. Al cancelar la reserva, puede añadir si quiere los motivos.

#### Actividad

Un funcionario de administración de hospitales necesita una aplicación para que los médicos indiquen en qué habitaciones tienen pacientes. En caso de tener un paciente, deben indicar el tratamiento y también pueden indicar si hace falta alguna medida especial. Por otro lado, los enfermeros usan la aplicación para indicar si han entrado en una habitación y, en ese caso, indican a qué enfermo han atendido y cómo lo han hecho.

#### Actividad

Un desarrollador de videojuegos quiere hacer un diagrama de casos de uso para organizar las acciones que pueden realizar los personajes de su juego. Los personajes pueden saltar, andar, correr, usar objetos y atacar. Si atacan y golpean algo, pueden rebotar. Por otro lado, los personajes también pueden morir. Un personaje, siempre que salta, cae, pero también puede tirarse por un borde y caer.

#### Actividad

Desarrolla un diagrama de casos de uso que represente, de forma aproximada, lo que puedes

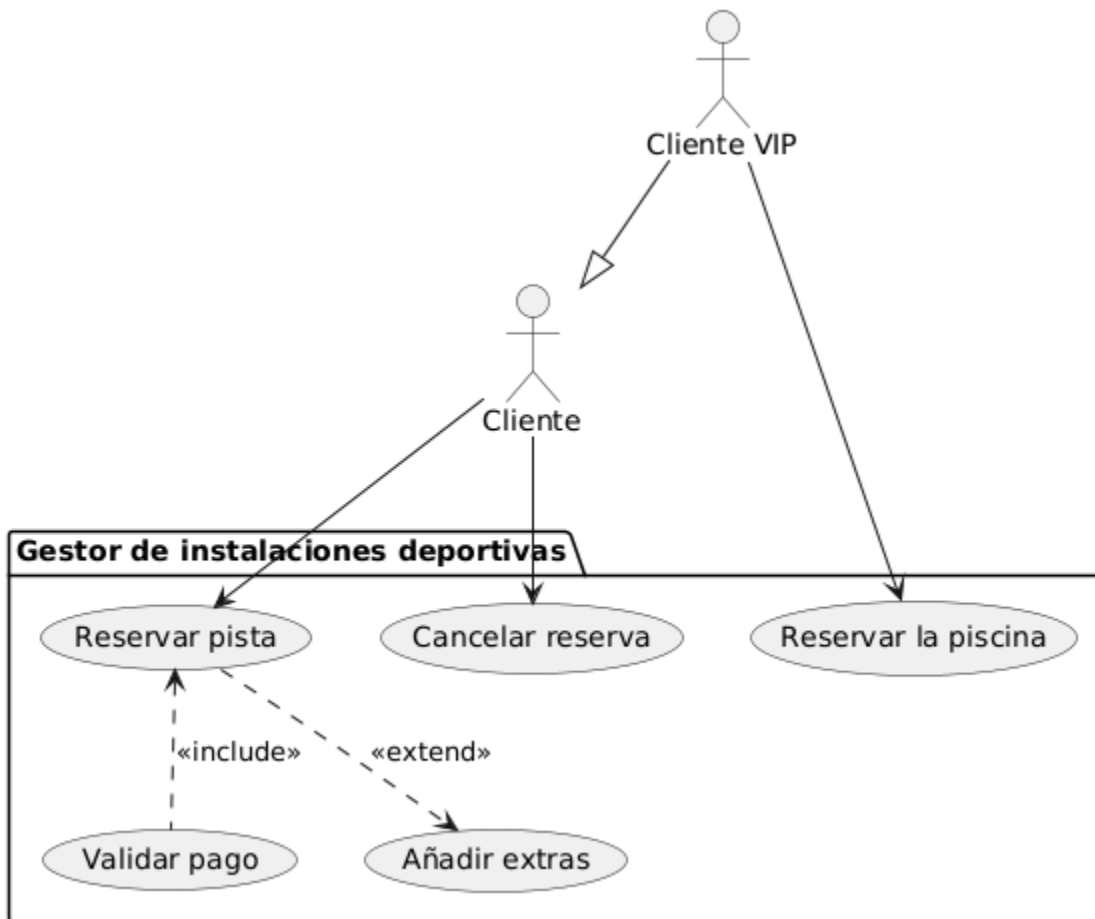


hacer en una aplicación como Dropbox, según el tipo de usuario que tienes contratado. Métete en los planes de empresa y compara los tipos de usuario.

## Generalización

Aparte de las relaciones `extend` y `include` contamos con relaciones de generalización entre actores, que funcionan de manera similar a lo ya visto en los diagramas de clases. Si unimos un actor **A** a un actor **B** mediante una flecha de generalización que apunta de A a B, lo que estamos diciendo es que el actor A es una especificación del actor B, es decir, cuenta con todos los casos de uso del actor B más los suyos propios.

Continuando con el ejemplo de las pistas deportivas, vamos a añadir un subtipo de cliente, el cliente VIP que, además de los casos de uso del cliente, también puede reservar la piscina.



### ► Ver el código en plantuml

#### Actividad

En la aplicación anterior del **restaurante**, se quiere añadir un tipo de clientes especiales que pueda hacer lo mismo que el cliente normal, pero que además pueda reservar en la zona VIP un menú especial.

## Actividad

En la aplicación anterior del **videojuego**, se quiere añadir también a los enemigos. Los enemigos son como los personajes, pero no pueden saltar. En cambio, un enemigo puede dejar llamar a otros enemigos y también puede escapar del personaje.

# Historias de Usuario (Scrum) y Casos de Uso

Las **historias de usuario (Scrum)** y los **casos de uso (UML)** se pueden relacionar, y hacerlo es una forma de integrar enfoques ágiles con técnicas de modelado más formales y tradicionales.

Para ello, hay que tener en cuenta que una **historia de usuario** es una descripción breve y general de una funcionalidad desde la perspectiva del usuario final. Suelen seguir el formato  
Como \[tipo de usuario], quiero \[acción o funcionalidad], para \[objetivo o beneficio]  
. Por ejemplo:

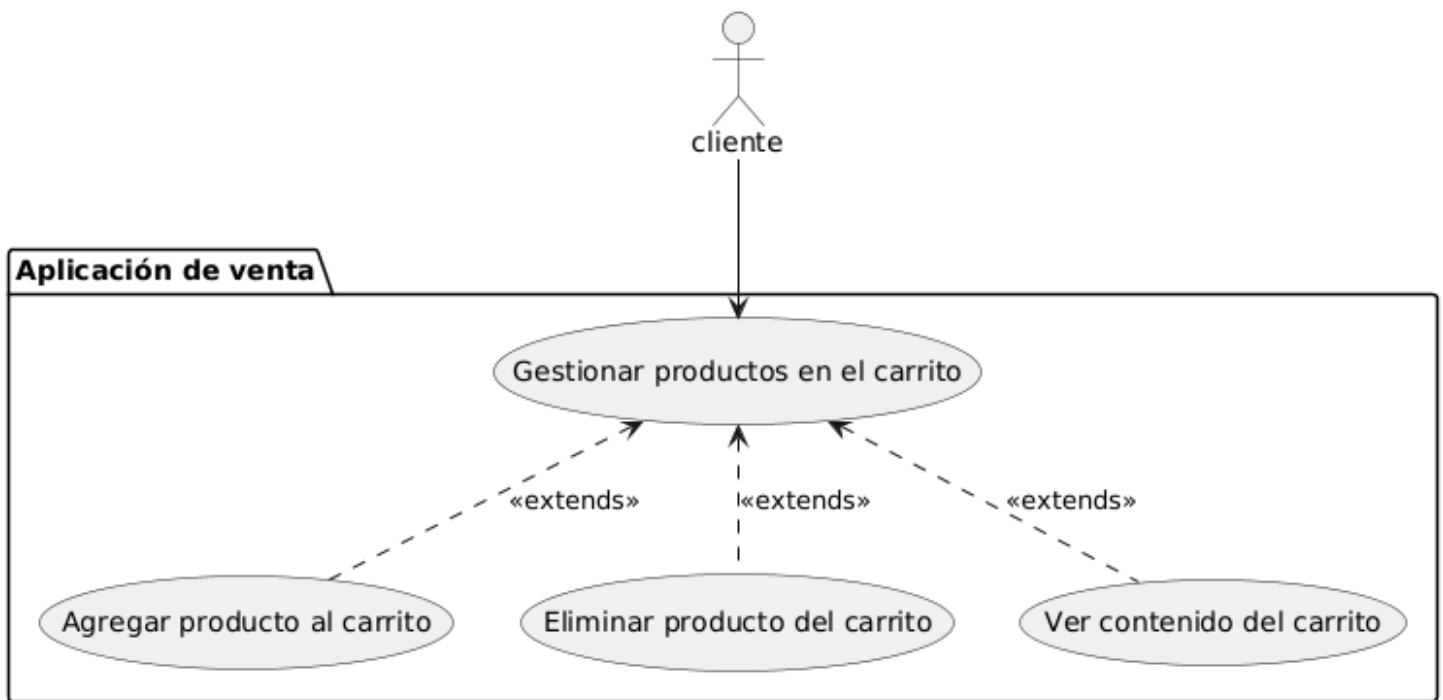
Como cliente, quiero agregar productos a un carrito de compras, para comprarlos después.

Por otro lado, un caso de uso describe cómo un usuario interactúa con un sistema para lograr un objetivo específico. Es decir, es más general ya que puede, o no integrar más de una función a través de extensiones o inclusiones, mientras que una historia de usuario necesita ser breve y concisa. Por lo tanto, conviene tener en cuenta lo siguiente:

### 1. Una historia de usuario puede corresponder a un caso de uso completo.

- Historia: "*Como usuario, quiero iniciar sesión para acceder a mi perfil.*"
- Caso de uso: "*Iniciar sesión*" (con actor: usuario; flujo básico: ingresar usuario/contraseña, verificación, redirección).

### 2. Varias historias de usuario pueden derivarse de un caso de uso complejo.



### ► Ver el código en plantuml

- Historias:
  - Como usuario, quiero agregar productos al carrito
  - Como usuario, quiero eliminar productos del carrito
  - Como usuario, quiero ver el contenido del carrito

### Actividad

Vuelve a la aplicación del **restaurante** y transforma los casos de uso en historias de usuario.

### Actividad

Crea un diagrama de casos de uso a partir de la siguiente definición:

Una máquina de café automática gestiona la preparación y entrega de bebidas calientes. En su funcionamiento, la máquina puede encontrarse en varios estados. Al inicio, la máquina está a la espera de que un usuario realice una selección. Cuando el usuario elige una bebida (como café, té o chocolate caliente), la máquina pasa a un proceso de preparación específico según la bebida seleccionada. Si el proceso de preparación concluye sin problemas, la máquina entrega el producto al usuario y regresa al estado inicial para una nueva solicitud.

Sin embargo, si durante la preparación ocurre un error, como falta de ingredientes o una avería técnica, la máquina debe entrar en un estado de error que bloquea nuevas solicitudes hasta que un técnico de mantenimiento intervenga. El técnico podrá reiniciar el sistema y devolver la máquina a su estado inicial. Además, el proceso de **preparación** debe incluir detalles específicos dependiendo del tipo de bebida elegida: preparar café, preparar té o preparar chocolate caliente.

# Diagramas de Transición de Estados

- Diagramas de Transición de Estados
  - Introducción a los Diagramas de Estados
  - Partes de un Diagrama de Estados
    - Estado
      - Transición
      - Evento
      - Estado Inicial y Estado Final
      - Diagrama 1: Ciclo de vida de una tarea
      - Diagrama 2: Sistema de un semáforo
      - Diagrama 3: Proceso de pago en línea
      - Diagrama 4: Gestión de una cuenta de usuario
  - Máquinas de Estados Simples y Compuestas
    - Máquina de estados simple
    - Máquinas de estados compuestas

## Introducción a los Diagramas de Estados

Un diagrama de estados es una representación gráfica del **comportamiento dinámico** de un objeto o sistema, mostrando los **estados** en los que puede encontrarse y las **transiciones** entre ellos debido a eventos. Se trata de la representación visual de una Máquina de Estados Finita (FSM por sus siglas en inglés).

Una máquina finita de estados cuenta con un conjunto finito de estados posibles, así como eventos o entradas que provocan transiciones entre estados. Siempre se tiene que definir un solo estado inicial y cero o muchos estados finales.

Un diagrama de estados facilita el modelado de sistemas en los que un objeto cambia de estado a medida que ocurren **eventos**. Por lo tanto, es útil en sistemas secuenciales como **máquinas expendedoras, procesos de pedidos, autenticación de usuarios, etc..**

Imagina que un semáforo es como un sistema que cambia de estado (verde, amarillo, rojo) en función de eventos (el tiempo transcurrido o la llegada de un coche). Así funcionan las máquinas de estados.

Los diagramas de transición de estados se usan en multitud de circunstancias, y se emplean para describir el ciclo de vida de un objeto en un sistema orientado a objetos, aunque son más antiguos que el propio lenguaje UML.

### Actividad

Señala tres ejemplos de la vida cotidiana, como el del semáforo, donde se pudiera aplicar un diagrama de estados.

## Partes de un Diagrama de Estados

### Estado

Un **estado** representa una condición en la que se encuentra un objeto.

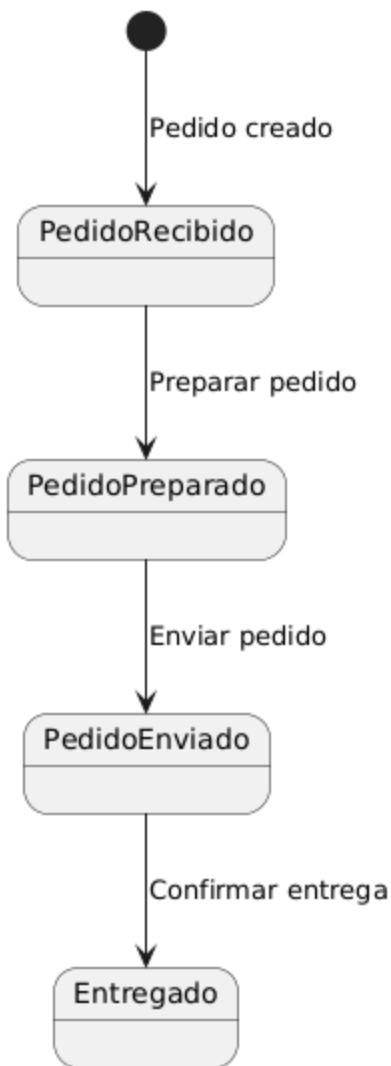


#### ► Ver el código en plantuml

- El punto negro (\*) representa el **estado inicial**.
- Estado1 y Estado2 son ejemplos de **estados**.

### Transición

Una **transición** conecta dos estados y ocurre cuando se dispara un **evento**. Las transiciones tienen un solo sentido.

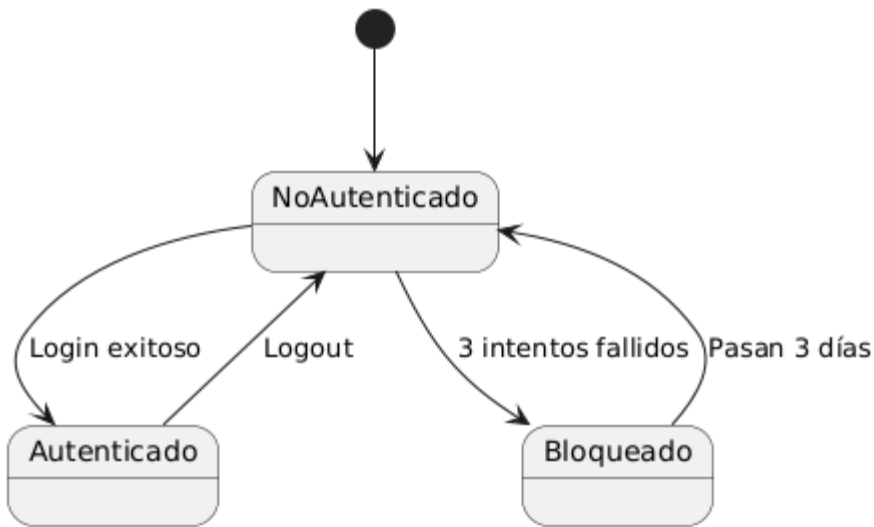


#### ► Ver el código en plantuml

- **Eventos:** "Pedido creado", "Preparar pedido", "Enviar pedido", etc.
- Cada flecha entre estados representa una **transición**. Una transición cambia desde el estado de donde nace la flecha hasta el estado donde apunta la flecha y ocurre cuando se dispara el evento.

## Evento

El **evento** es la acción que provoca una transición. Puede escribirse como texto en las flechas.

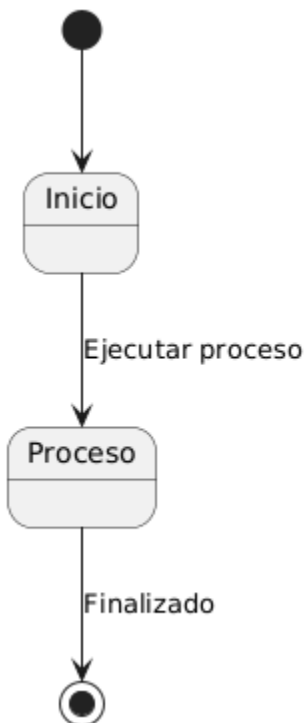


#### ► Ver el código en plantuml

- Estados: *NoAutenticado*, *Autenticado*, *Bloqueado*.
- Eventos: *Login exitoso*, *3 intentos fallidos*, *Logout*.

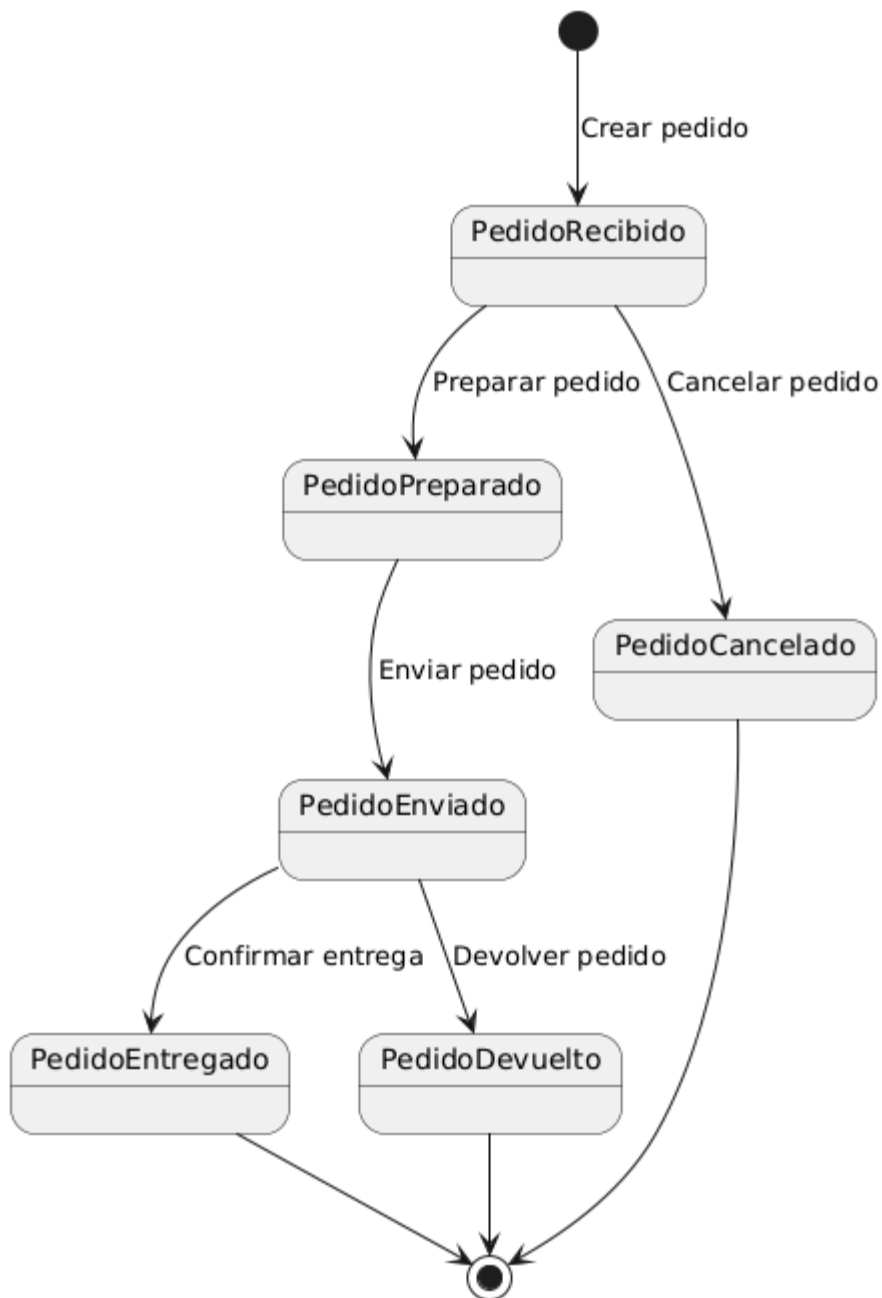
## Estado Inicial y Estado Final

- **Estado inicial:** Punto de partida del sistema (círculo relleno).
- **Estado final:** Punto donde el proceso termina (círculo doble). El proceso puede ser cíclico y no terminar.



#### ► Ver el código en plantuml

En el siguiente ejemplo, se muestra cómo un pedido pasa por varios estados durante su procesamiento.



### ► Ver el código en plantuml

En este diagrama podemos observar lo siguiente:

1. **Estados:** PedidoRecibido, PedidoPreparado, PedidoEnviado, PedidoEntregado, PedidoCancelado, PedidoDevuelto.
2. **Transiciones:**
  - *Crear pedido*: De [ \* ] a PedidoRecibido .
  - *Preparar pedido*: De PedidoRecibido a PedidoPreparado .
  - *Enviar pedido*: De PedidoPreparado a PedidoEnviado .
  - *Cancelar pedido*: Transición directa a PedidoCancelado .



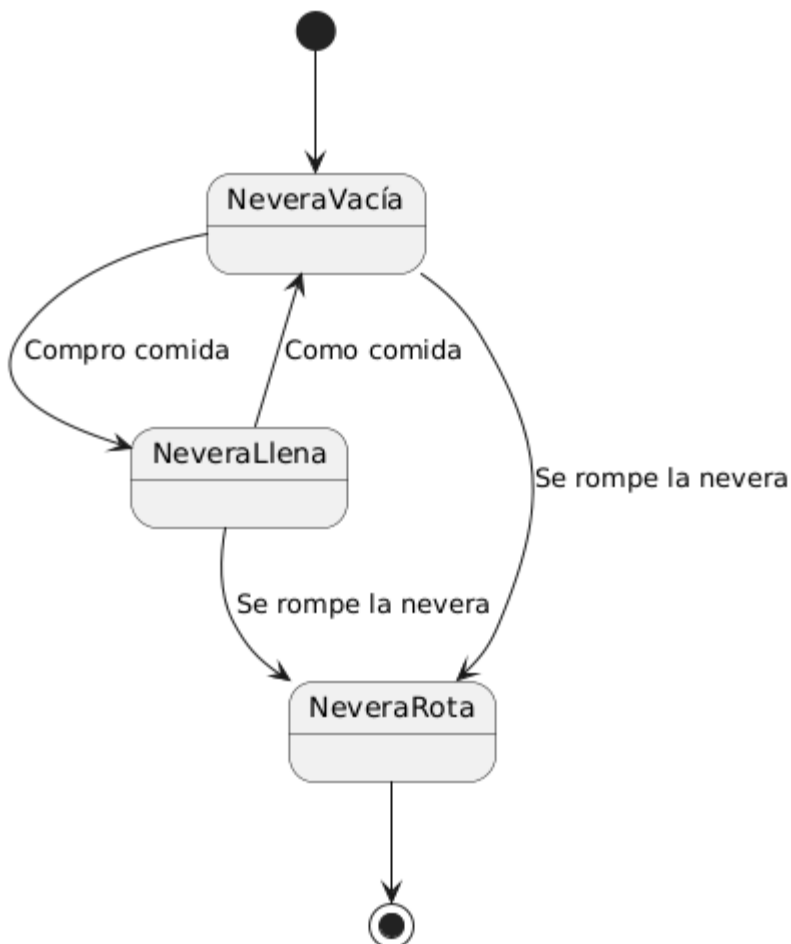
## Actividad

Crea diagramas de estados de los siguientes objetos:

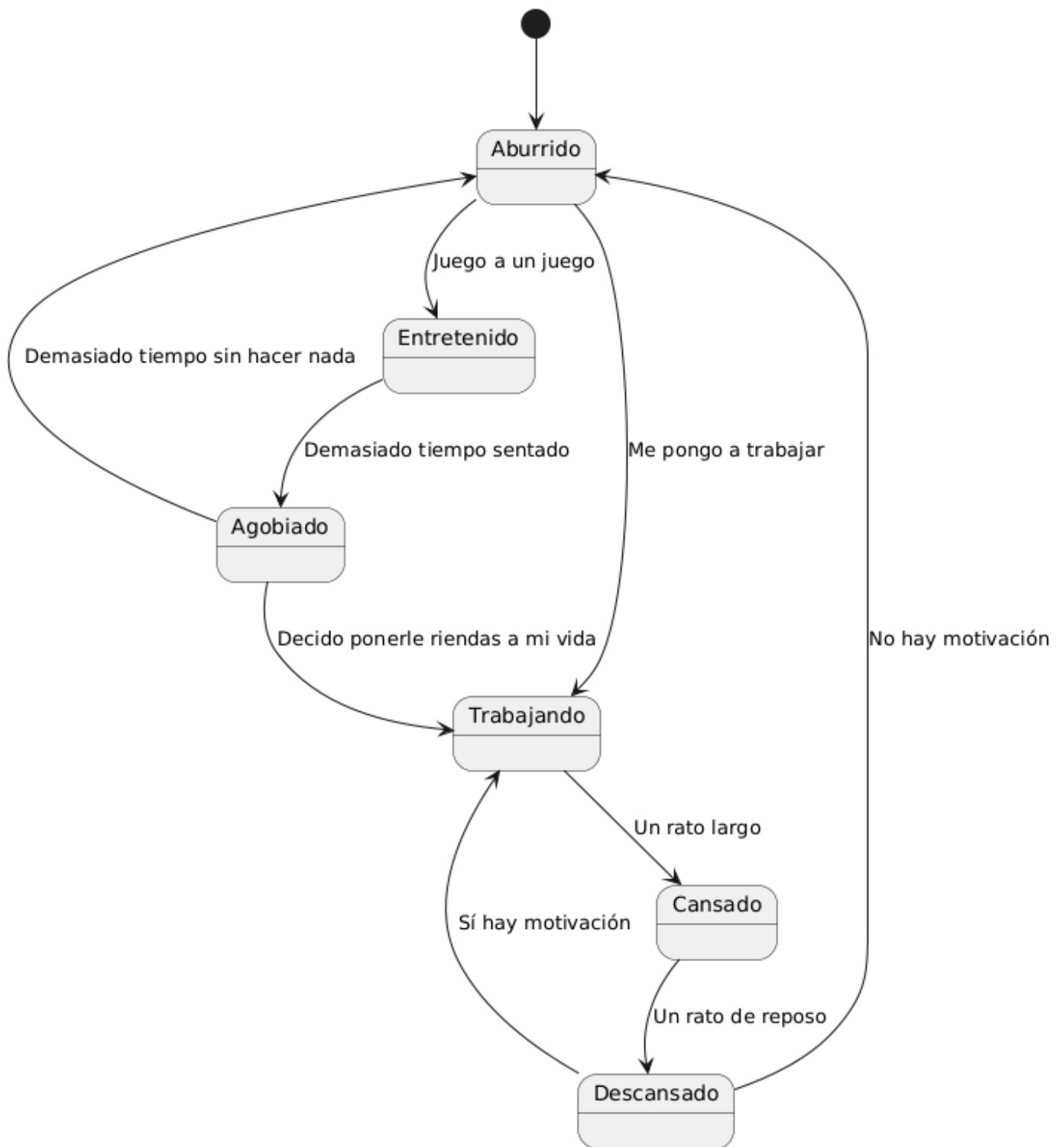
1. Una puerta que se puede abrir o cerrar, y puede estar cerrada, abierta o en proceso de apertura o cierre. Las puertas empiezan cerradas.
2. El ciclo de vida de una mariposa, que comienza como un huevo, eclosiona y se transforma en una larva, cuando se ha alimentado lo suficiente crea un capullo de seda y comienza una metamorfosis, cuando termina se transforma en mariposa, se aparea, pone huevos y muere.
3. Las animaciones de un personaje de un videojuego, donde el personaje puede andar, estar en reposo, saltar, caer y morir.

## Actividad

Interpreta los siguientes diagramas:



► [Ver el código en plantuml](#)



► Ver el código en plantuml

**Identifica el significado de los siguientes diagramas de estados.** Para ello, sigue los siguientes pasos:

- ¿Qué representa cada estado?
- ¿Qué significado tienen las transiciones?
- ¿Qué proceso o sistema modela el diagrama?

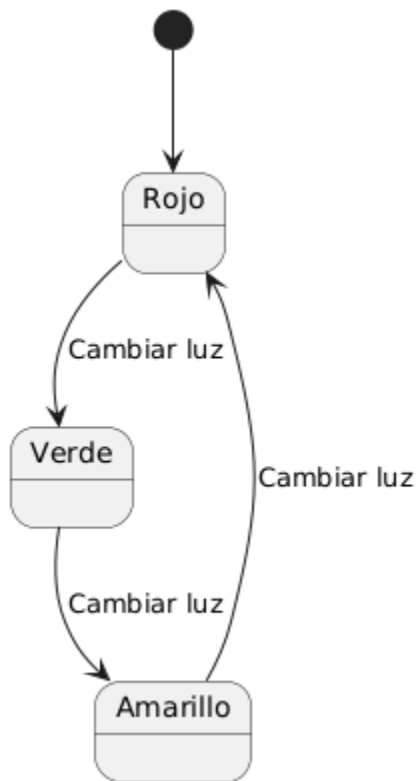
Finalmente, escribe una descripción breve (de 3-5 líneas) para cada diagrama, explicando su **función** y **aplicación en el mundo real**.

### Diagrama 1: Ciclo de vida de una tarea



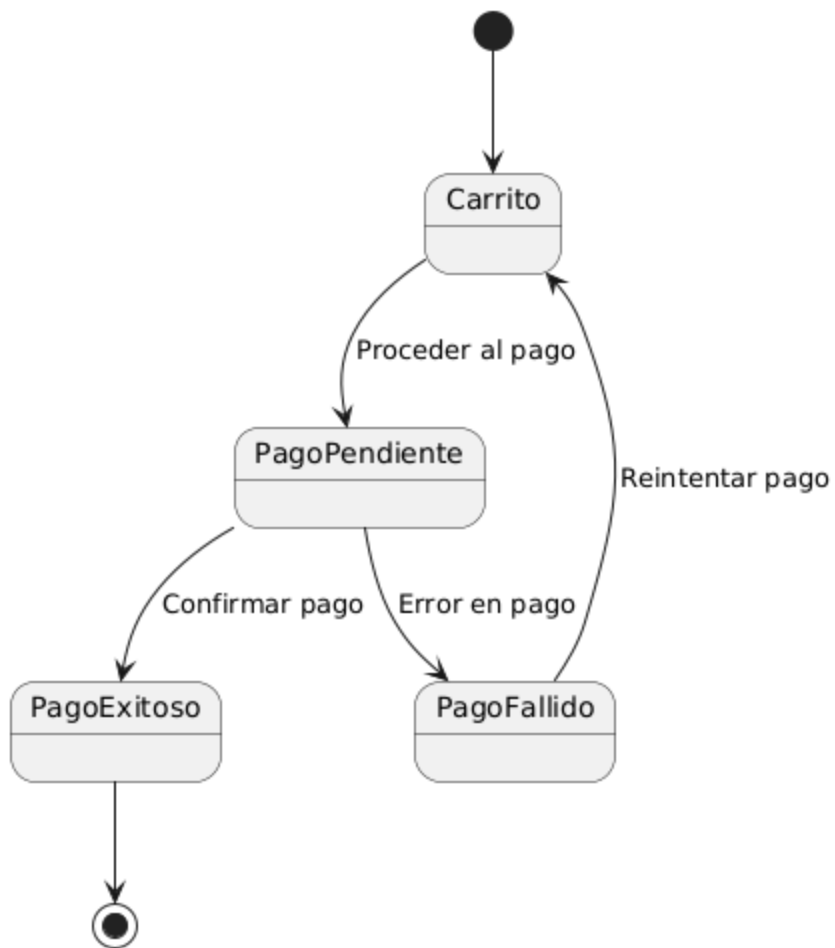
► [Ver el código en plantuml](#)

**Diagrama 2: Sistema de un semáforo**



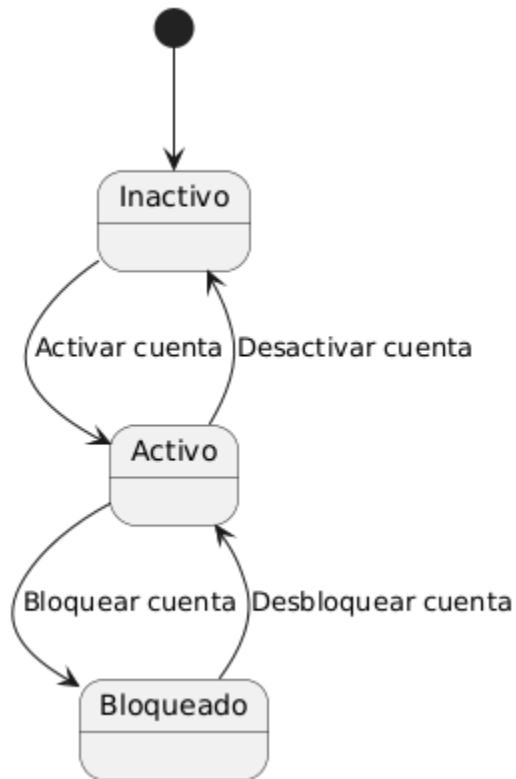
► [Ver el código en plantuml](#)

### Diagrama 3: Proceso de pago en línea



► [Ver el código en plantuml](#)

## Diagrama 4: Gestión de una cuenta de usuario



► [Ver el código en plantuml](#)

## Máquinas de Estados Simples y Compuestas

### Máquina de estados simple

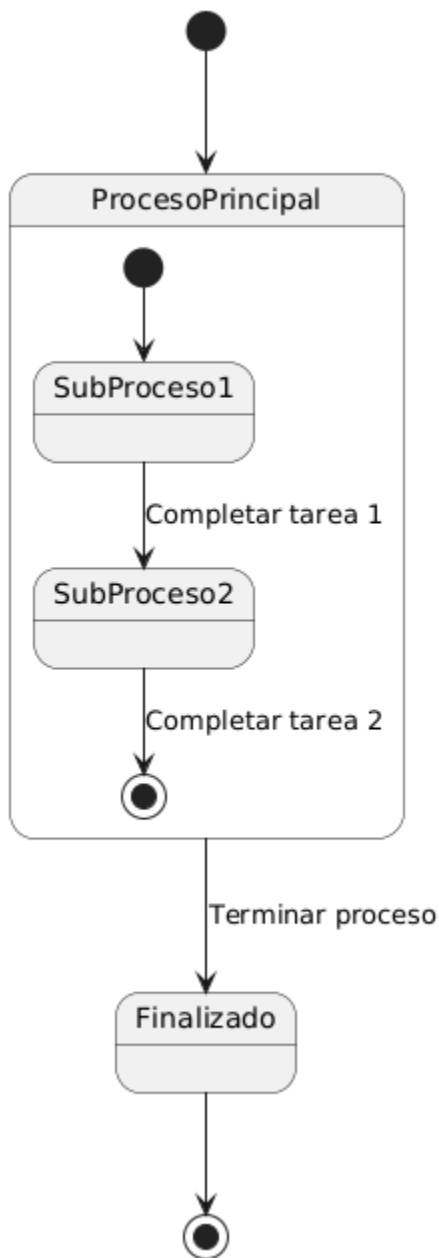
Una máquina de estados simple representa de forma lineal los estados y las transiciones.



► [Ver el código en plantuml](#)

## Máquinas de estados compuestas

Una máquina de estados compuesta representa estados que pueden contener **subestados**.



## ► Ver el código en plantuml

### Actividad

Añade al diagrama del personaje del videojuego el estado correr, que a su vez se compone de los estados de arrancada, carrera y parada. De la parada se vuelve al estado de reposo, pero al estado salto solo se puede llegar desde el estado carrera (o desde el estado reposo).

### Actividad

Crea un diagrama de transición de estados a partir de la siguiente definición:

Una máquina de café automática gestiona la preparación y entrega de bebidas calientes. En su funcionamiento, la máquina puede encontrarse en varios estados. Al inicio, la máquina está a la espera de que un usuario realice una selección. Cuando el usuario elige una bebida (como café, té o chocolate caliente), la máquina pasa a un proceso de preparación específico según la



bebida seleccionada. Si el proceso de preparación concluye sin problemas, la máquina entrega el producto al usuario y regresa al estado inicial para una nueva solicitud.

Sin embargo, si durante la preparación ocurre un error, como falta de ingredientes o una avería técnica, la máquina debe entrar en un estado de error que bloquea nuevas solicitudes hasta que un técnico de mantenimiento intervenga. El técnico podrá reiniciar el sistema y devolver la máquina a su estado inicial. Además, el proceso de **preparación** debe incluir detalles específicos dependiendo del tipo de bebida elegida: preparar café, preparar té o preparar chocolate caliente.

# Diagramas de actividad

- [Diagramas de actividad](#)
  - [Diagramas actividad](#)
    - [Componentes básicos de un diagrama de actividad](#)
  - [Estructuras comunes](#)
    - [Estructuras if-else, if y switch](#)
    - [Estructuras while, do-while y for](#)
    - [Programación modular](#)
  - [Concurrencia en diagramas de actividad](#)
    - [Representación de los hilos](#)
    - [Uso de carriles en los diagramas de actividad](#)

## Diagramas actividad

Los diagramas de actividad son representaciones gráficas de algoritmos o procesos, en las que se usan símbolos específicos para mostrar los pasos y decisiones. Son una herramienta esencial en la planificación, análisis y comunicación de sistemas.

Los diagramas de actividad son equivalentes a los diagramas de flujo que se emplean fuera de UML.

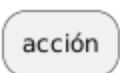
## Componentes básicos de un diagrama de actividad

1. **Flechas:** Conectan los diferentes símbolos y muestran el flujo del proceso.
2. **Círculo:** Representa el inicio (simple) o fin del proceso (doble).



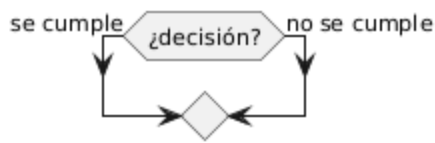
► [Ver el código en plantuml](#)

3. **Rectángulo:** Indica una acción o proceso.



► [Ver el código en plantuml](#)

4. **Rombo o hexágono:** Representa una decisión, como una bifurcación. Se usan siempre dos: una para abrir y otra para cerrar.



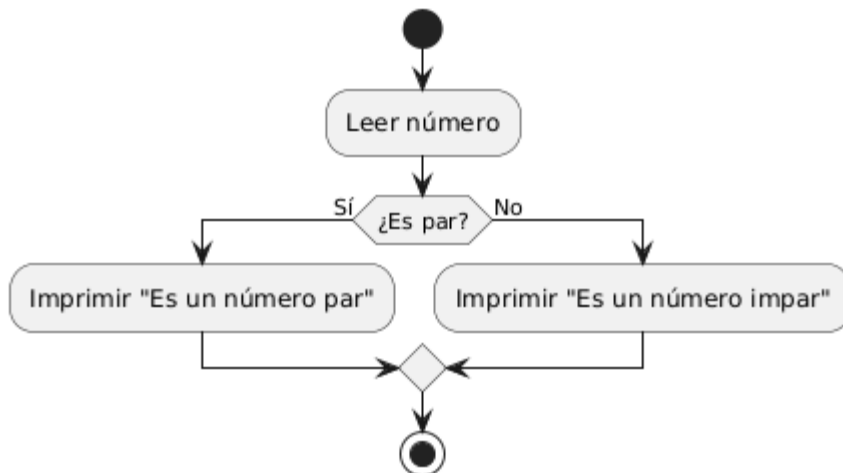
► [Ver el código en plantuml](#)

## Estructuras comunes

### Estructuras if-else, if y switch

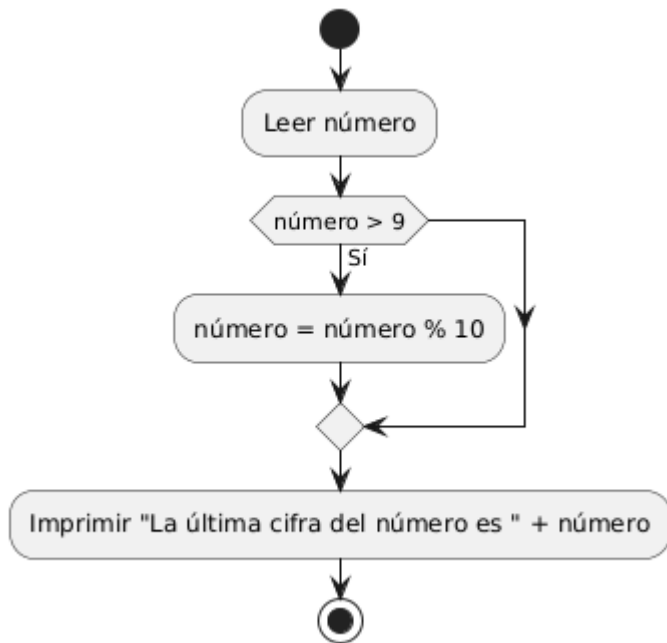
Estas estructuras se utilizan para representar decisiones lógicas.

**Ejemplo de if-else:**



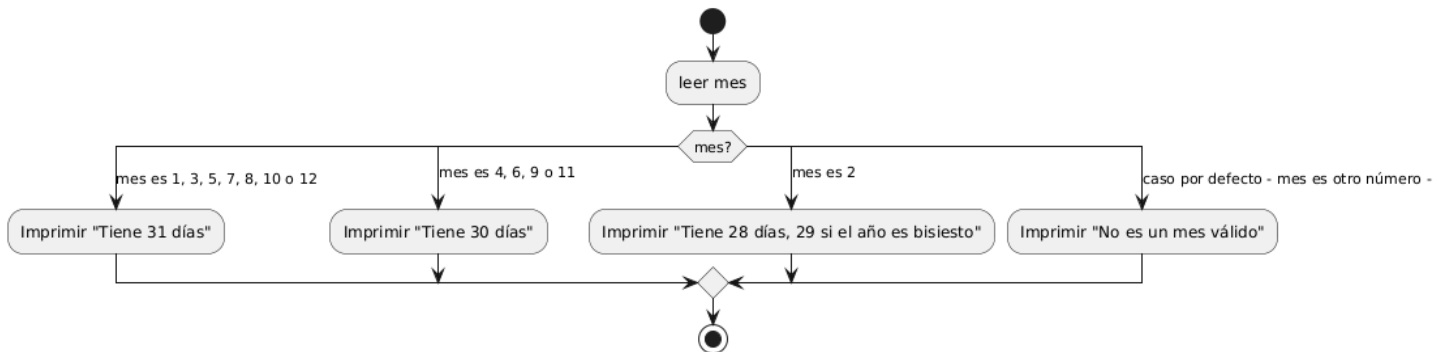
► [Ver el código en plantuml](#)

**Ejemplo de if:**



► [Ver el código en plantuml](#)

### Ejemplo de Switch-case

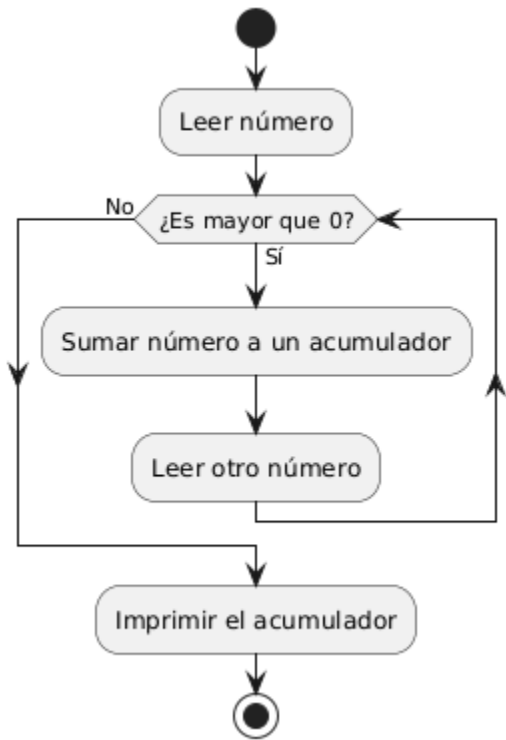


► [Ver el código en plantuml](#)

## Estructuras while, do-while y for

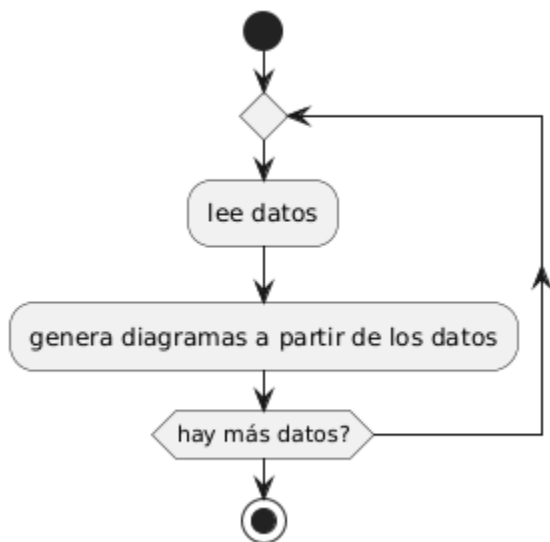
Los bucles permiten repetir una acción varias veces.

**Ejemplo de bucle while que pide suma números hasta que introduces 0:**



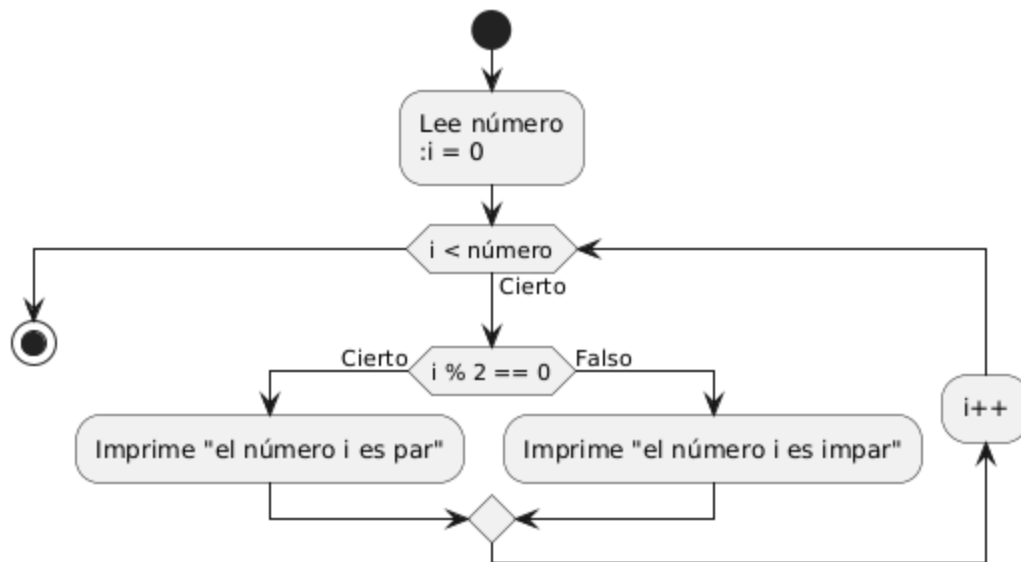
► [Ver el código en plantuml](#)

Ejemplo de bucle do-while para hacer diagramas:



► [Ver el código en plantuml](#)

Ejemplo de bucle for para separar los números pares e impares desde 0 hasta una entrada dada por el usuario:



► [Ver el código en plantuml](#)

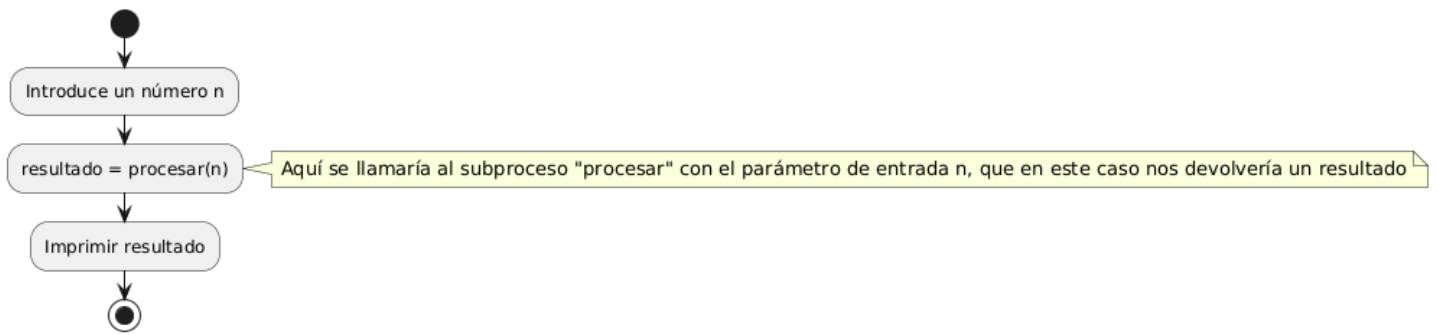
## Programación modular

Dentro del diseño de programas, existe un paradigma que es el de la programación modular. En él, se adopta un enfoque de diseño descendente en el cual un problema grande se divide en subproblemas más pequeños que permiten simplificar el problema general.

Estos subproblemas se conocen como módulos y tienen la ventaja de ser más fáciles de probar, mantener y corregir, además de poder ser reutilizados en diferentes partes del problema principal o incluso en otros problemas distintos. Los módulos se pueden invocar desde cualquier otro módulo. Un módulo tiene un grado de entrada igual al número de veces que es invocado y un grado de salida igual al número de módulos que invoca.

La dependencia entre módulos se llama **acoplamiento**, y puede ser peligrosa si va más allá de que un módulo emplee el resultado de otro. Por ello, es conveniente diseñar los módulos como si fueran **cajas negras**, es decir, de tal forma que solo nos importe los parámetros que le pasamos como entrada y el resultado que nos devuelve.

Podemos representar la invocación a módulos en los diagramas de actividad como una instrucción o condición cualquiera, indicando entre paréntesis los parámetros que le pasamos.



► [Ver el código en plantuml](#)

## Concurrencia en diagramas de actividad

Aunque no se trabaja programación concurrente en el primer curso del ciclo, es apropiado conocer cómo representar flujos paralelos (hilos) en un diagrama de actividad. Esta técnica es útil cuando se modelan procesos que pueden ejecutarse simultáneamente, algo común en sistemas donde las operaciones independientes pueden ocurrir al mismo tiempo (como cuando se consulta una base de datos externa o se trabaja con interfaces gráficas).

### Representación de los hilos

En un diagrama de actividad, los hilos (o actividades concurrentes) se representan mediante **ramificaciones paralelas**, utilizando nodos de bifurcación (*fork*) y unión (*join*):

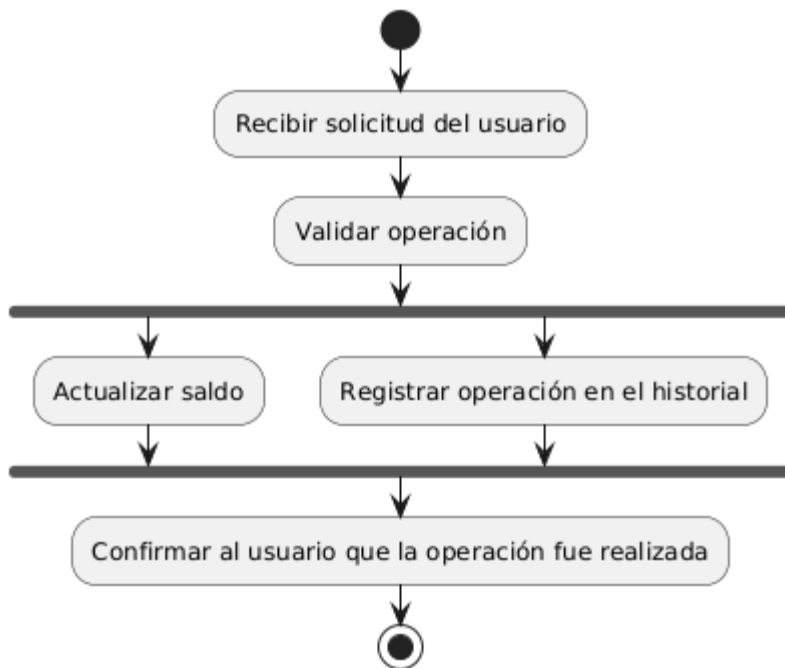
- **Fork (Bifurcación):** Representado por una línea horizontal o vertical gruesa. Indica que el flujo se divide en varios caminos paralelos que se ejecutan simultáneamente.
- **Join (Unión):** También representado por una línea gruesa, marca el punto donde los flujos paralelos se sincronizan para continuar con un único flujo.

Por ejemplo, Supongamos un sistema de cajero automático que debe realizar dos tareas en paralelo:

1. **Actualizar el saldo de la cuenta.**
2. **Registrar la operación en un historial de transacciones.**

Ambas tareas pueden ejecutarse de forma concurrente, ya que no dependen entre sí, pero deben completarse antes de que el sistema pueda finalizar la operación.

**Representación en un diagrama de actividad:**



### ► Ver el código en plantuml

En este caso, se puede observar que el diagrama representa las siguientes acciones:

1. **Inicio del flujo:** El sistema recibe la solicitud del usuario y valida que la operación sea posible.
2. **Bifurcación:** El flujo se divide en dos hilos: uno para actualizar el saldo y otro para registrar la operación en el historial.
3. **Unión:** Ambos hilos se sincronizan para continuar con el flujo principal.
4. **Fin del flujo:** El sistema confirma al usuario que la operación fue realizada.

Es importante no confundir los hilos con las diferentes opciones que nos proporciona un `if-else` o un `switch`. En el caso de los hilos, **todas** las acciones desde el **fork** se llevan a cabo de manera simultánea y el programa espera en el punto de encuentro, el **join**. Como curiosidad, el nombre de **fork** viene de tenedor (**fork** en inglés es tenedor), pues su representación se asemeja a un tenedor (en realidad no, viene de bifurcación, que es el otro significado de fork, pero sirve como regla mnemotécnica).

Aunque no vayamos a hacer programación concurrente este curso, la división en hilos sí que nos permite representar las siguientes cosas:

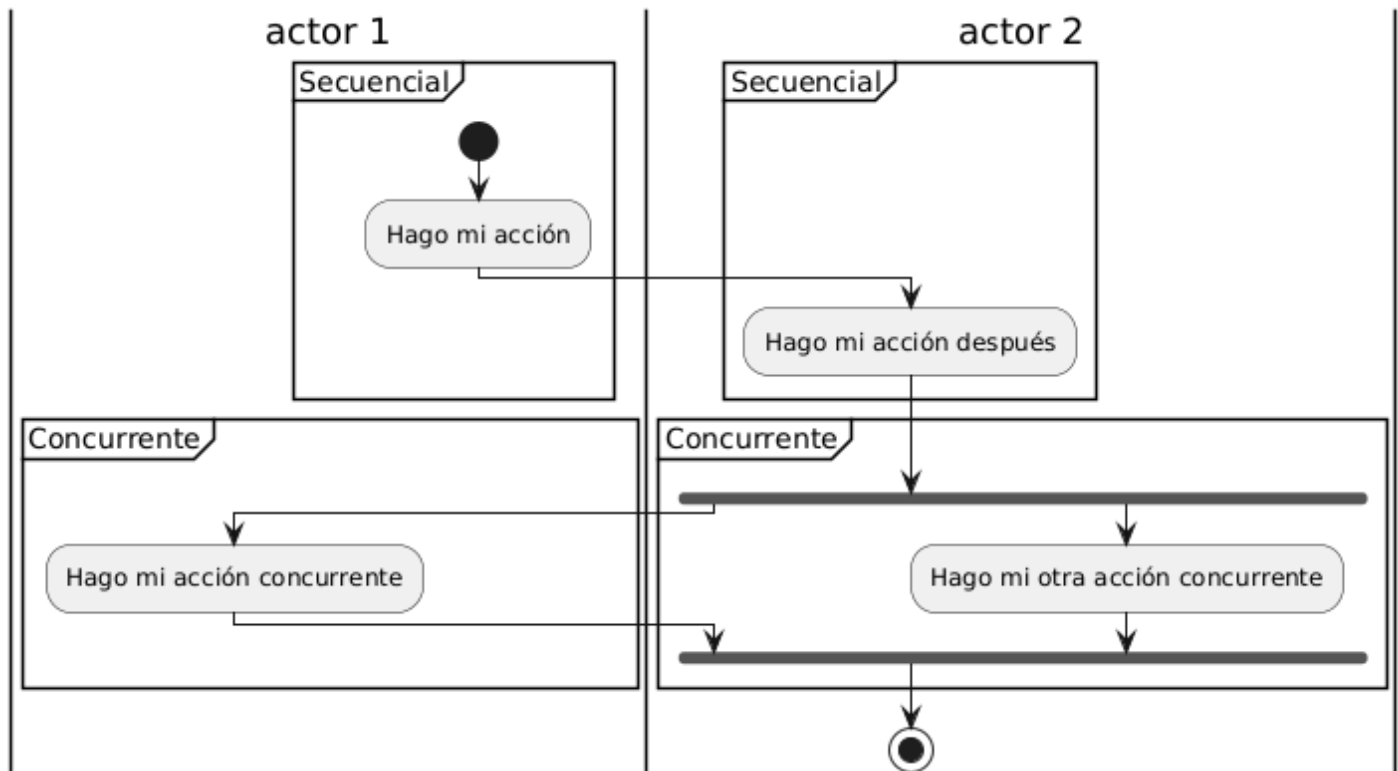
- Visualizar procesos que pueden ejecutarse en paralelo, optimizando tiempos.
- Detectar posibles dependencias entre actividades que podrían causar errores.
- Representar sistemas más complejos con mayor precisión.
- Planificar los sprints de un equipo de desarrollo.



# Uso de carriles en los diagramas de actividad

Para representar la intervención de varios actores dentro de una misma actividad, se puede dividir el diagrama de actividad en carriles, reservando cada carril para cada actor.

Aunque los carriles son especialmente prácticos para describir las actividades concurrentes, se pueden emplear en cualquier momento. Asimismo, también podemos agrupar diferentes bloques de acciones para facilitar la interpretación del diagrama.



## ► Ver el código en plantuml

**Nota:** En Plantuml, los carriles dividen las particiones por la mitad. Sin embargo, al diseñarlos con otros editores, como Visual Paradigm o en papel y lápiz, puedes agruparlos como te resulte más conveniente

### Actividad

Crea un diagrama de actividad a partir de la siguiente definición:

Una máquina de café automática gestiona la preparación y entrega de bebidas calientes. En su funcionamiento, la máquina puede encontrarse en varios estados. Al inicio, la máquina está a la espera de que un usuario realice una selección. Cuando el usuario elige una bebida (como café, té o chocolate caliente), la máquina pasa a un proceso de preparación específico según la bebida seleccionada. Si el proceso de preparación concluye sin problemas, la máquina entrega el producto al usuario y regresa al estado inicial para una nueva solicitud.

Sin embargo, si durante la preparación ocurre un error, como falta de ingredientes o una avería técnica, la máquina debe entrar en un estado de error que bloquea nuevas solicitudes hasta que un técnico de mantenimiento intervenga. El técnico podrá reiniciar el sistema y devolver la máquina a su estado inicial. Además, el proceso de **preparación** debe incluir detalles específicos dependiendo del tipo de bebida elegida: preparar café, preparar té o preparar chocolate caliente.

# Diagramas de interacción

Existen varios tipos de diagramas de interacción.

- Diagramas de secuencia
- Diagramas de comunicación
- Diagramas de tiempos
- Diagramas de interacción global

En este apartado, vamos a trabajar los diagramas de secuencia por ser los más relevantes y, al mismo tiempo, los más complejos de entender. Al final, introduciremos los diagramas de comunicación.

- Diagramas de interacción
  - ¿Qué es un diagrama de secuencia y para qué se usa?
  - Componentes básicos de un diagrama de secuencia y cómo construirlo
    - Actores y Objetos
    - Líneas de vida
      - Líneas de vida y bloques de activación
      - Fragmentos combinados: condicionales y bucles
      - Condicional (alt)
      - Bucle (loop)
      - Combinación de activación y fragmentos
      - Añadir estados al diagrama de secuencia
    - Mensajes
    - Construcción de diagramas de secuencia
  - Diagramas de comunicación

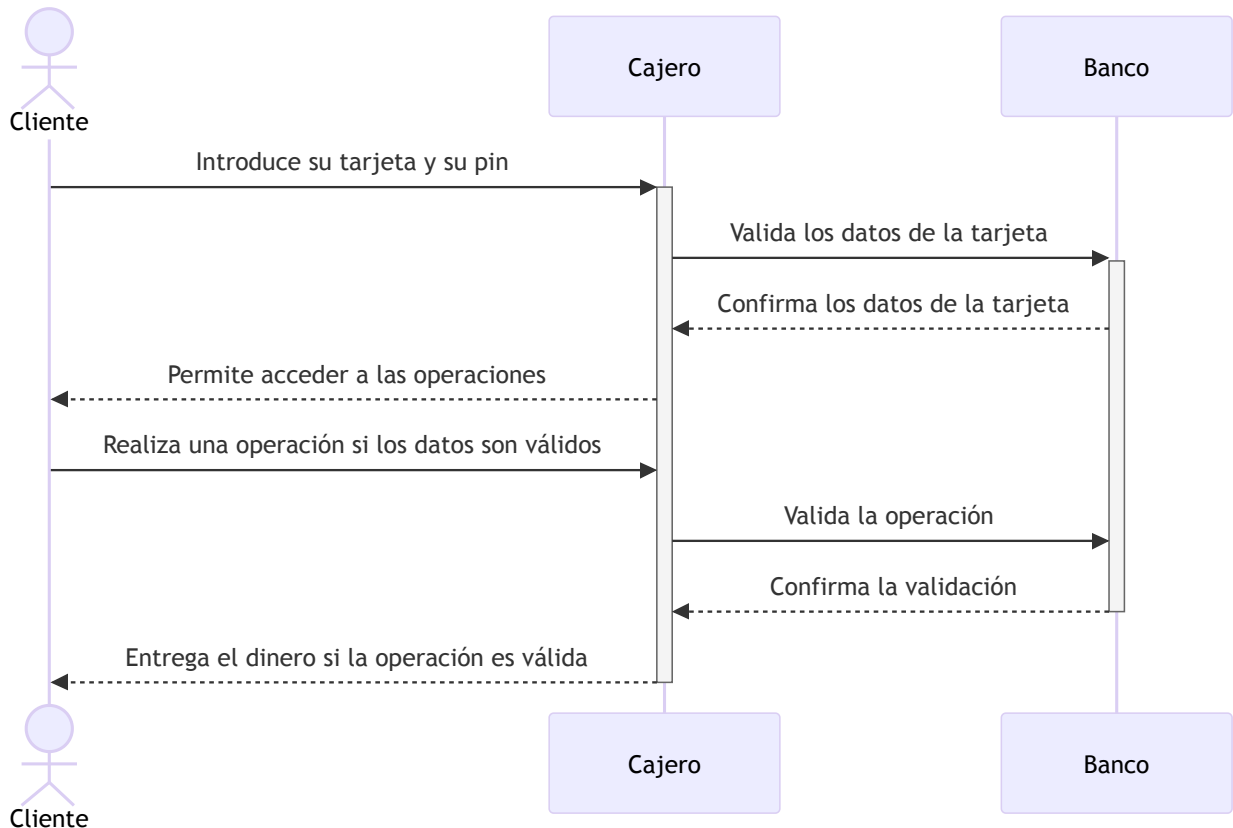
## ¿Qué es un diagrama de secuencia y para qué se usa?

Un **diagrama de secuencia** es un tipo de diagrama de comportamiento en UML que modela la interacción entre actores (usuarios u otros sistemas) y los objetos de un sistema en un orden temporal. Su principal objetivo es mostrar el flujo de mensajes o eventos que ocurren entre diferentes participantes a lo largo del tiempo, destacando el **orden cronológico** de las interacciones.

Se utiliza para:

- **Especificar** cómo se desarrolla un proceso dentro del sistema (por ejemplo, cómo se valida un pedido o cómo un usuario realiza una compra).
- **Diseñar o analizar escenarios** concretos de interacción entre componentes del sistema.
- **Comunicar ideas entre desarrolladores**, analistas y diseñadores.

Un ejemplo típico sería la interacción entre un cliente, un cajero automático y un banco para retirar dinero. El cliente introduce su tarjeta, el cajero se conecta al banco para validar los datos, el cliente realiza la operación y finalmente, si todo es correcto, entrega el dinero (En el caso de que la operación fuera sacar el dinero).



## Componentes básicos de un diagrama de secuencia y cómo construirlo

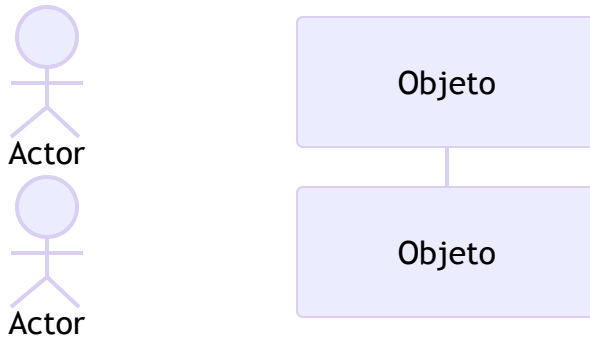
Un diagrama de secuencia se compone de los siguientes elementos:

### Actores y Objetos

Representan las entidades participantes:

- Los actores suelen ser personas o sistemas externos.
- Los objetos son instancias del sistema que participan en la interacción.

Se representan como una línea y el nombre de la entidad aparece tanto al principio como al final de esta. Es común acompañar los nombres de símbolos representativos:



## Líneas de vida

Son líneas verticales asociadas a cada actor/objeto. Representan el tiempo durante el cual estos están activos. Dentro de ellas se puede incluir determinada información:

1. **Bloques de activación:**

Rectángulos sobre las líneas de vida que indican que el actor/objeto está realizando una tarea. Pueden contener incluso estados como los trabajados en los diagramas de estados.

2. **Fragmentos combinados:**

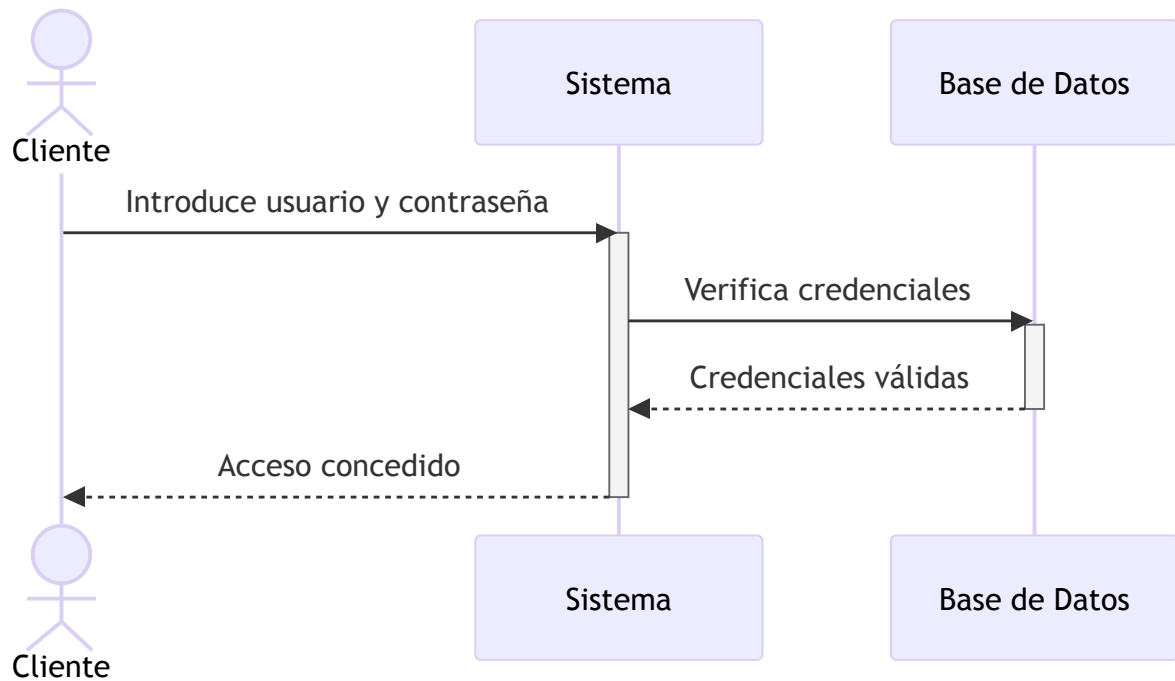
Estructuras condicionales o repetitivas que representan lógica compleja (por ejemplo, bucles o decisiones).

A continuación, vamos a profundizar un poco más en las líneas de vida:

## Líneas de vida y bloques de activación

Las líneas de vida son las líneas verticales asociadas a actores y objetos, y los bloques de activación son los rectángulos sobre estas líneas que indican que un actor/objeto está realizando una tarea específica.

En el siguiente ejemplo se representa una conversación entre un cliente y un sistema, donde se activa un bloque durante la validación de credenciales.



En este caso:

- El **Sistema** está activo mientras procesa la solicitud del cliente.
- La **Base de Datos** está activa mientras verifica las credenciales.

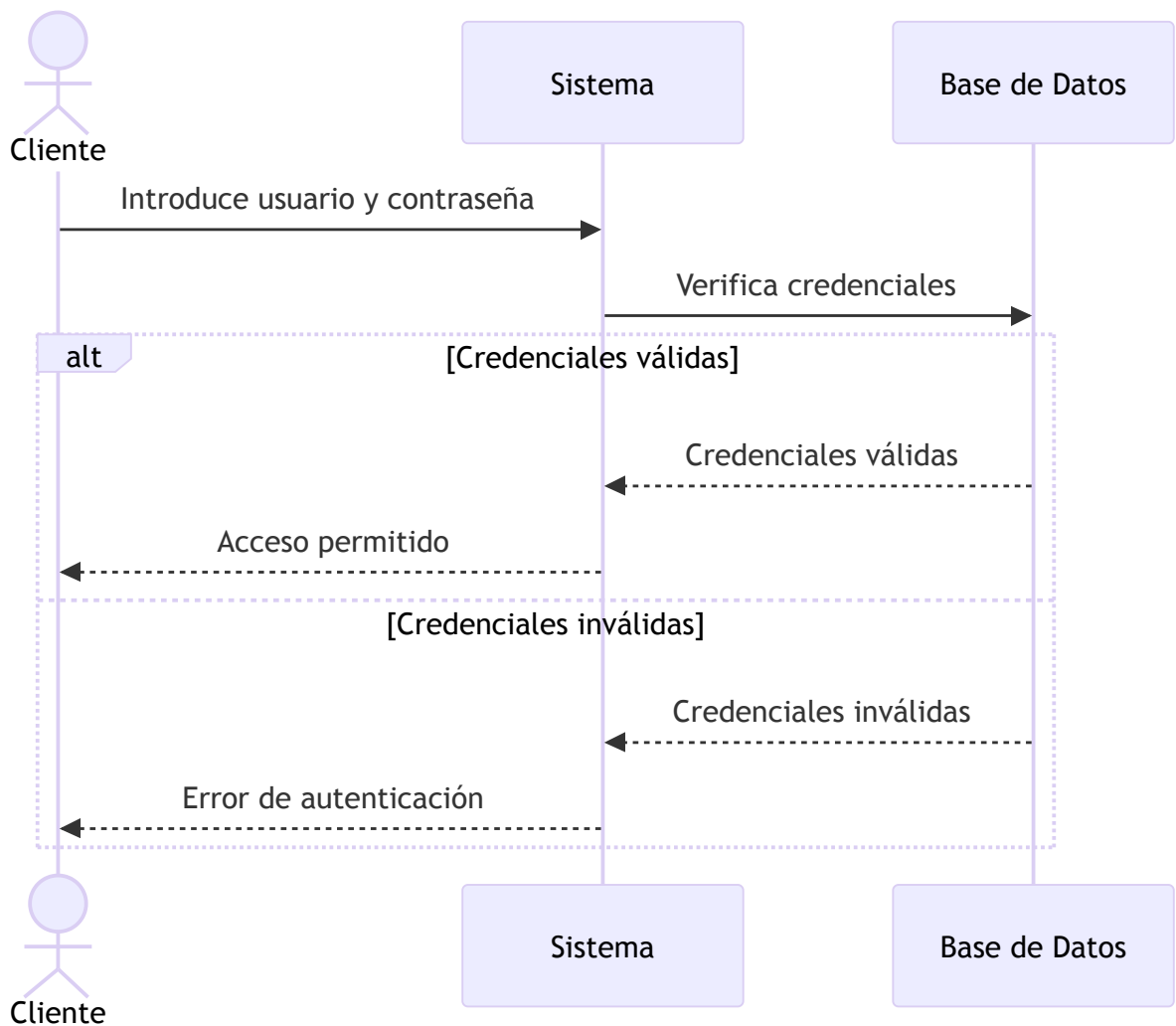
## Fragmentos combinados: condicionales y bucles

Los fragmentos combinados son estructuras que muestran flujos condicionales ( `alt` ) o repetitivos ( `loop` ).

### Condicional (alt)

El **fragmento condicional** permite modelar flujos donde existen diferentes caminos dependiendo de una condición.

**Ejemplo: Validación de credenciales con diferentes respuestas:**



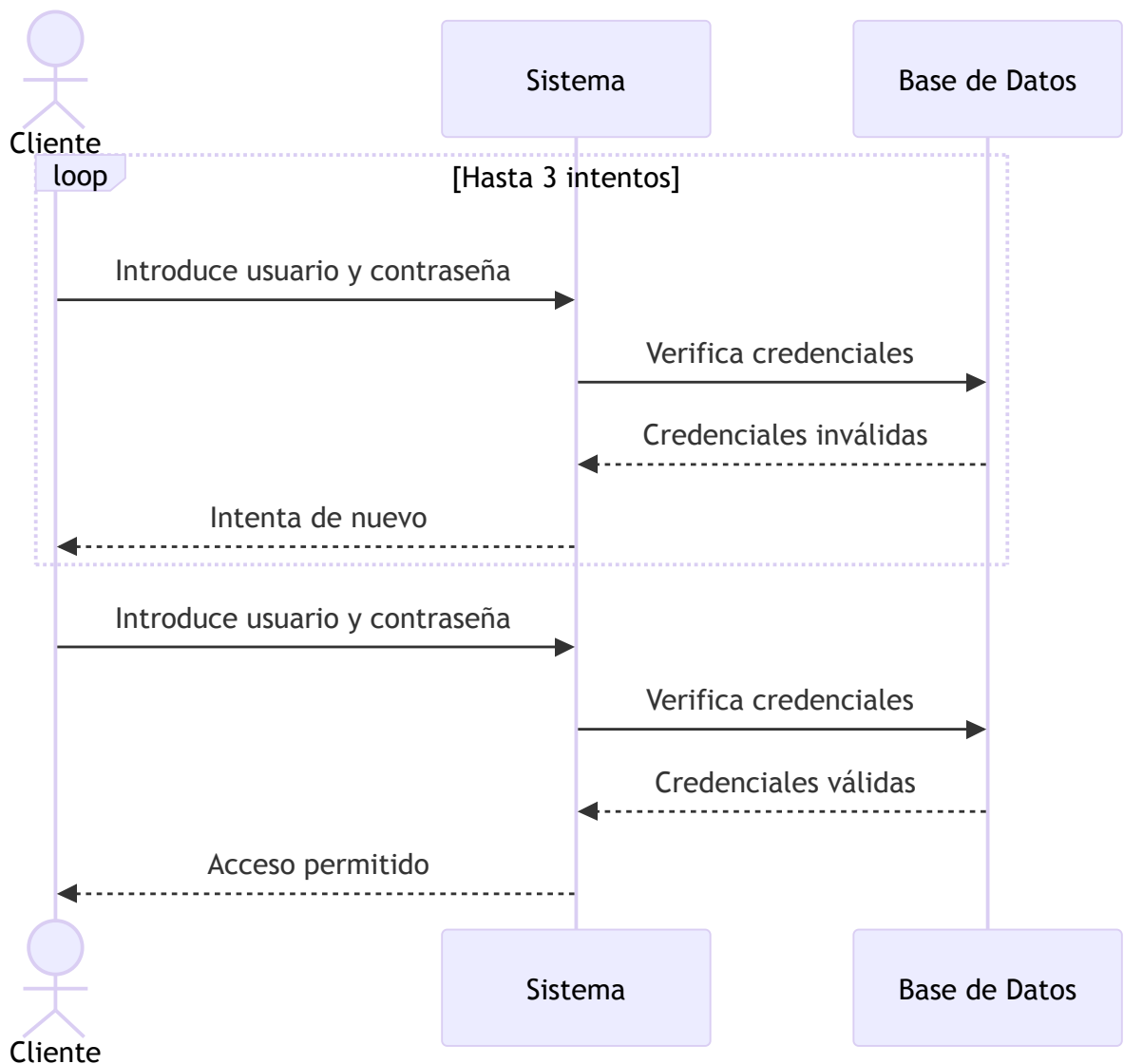
En este caso:

- Si las credenciales son válidas, el cliente recibe acceso.
- Si las credenciales son inválidas, se le notifica un error.

## Bucle (loop)

El **fragmento de bucle** representa procesos que se repiten hasta cumplir una condición.

**Ejemplo: Reintento de inicio de sesión (máximo 3 intentos):**



En este caso:

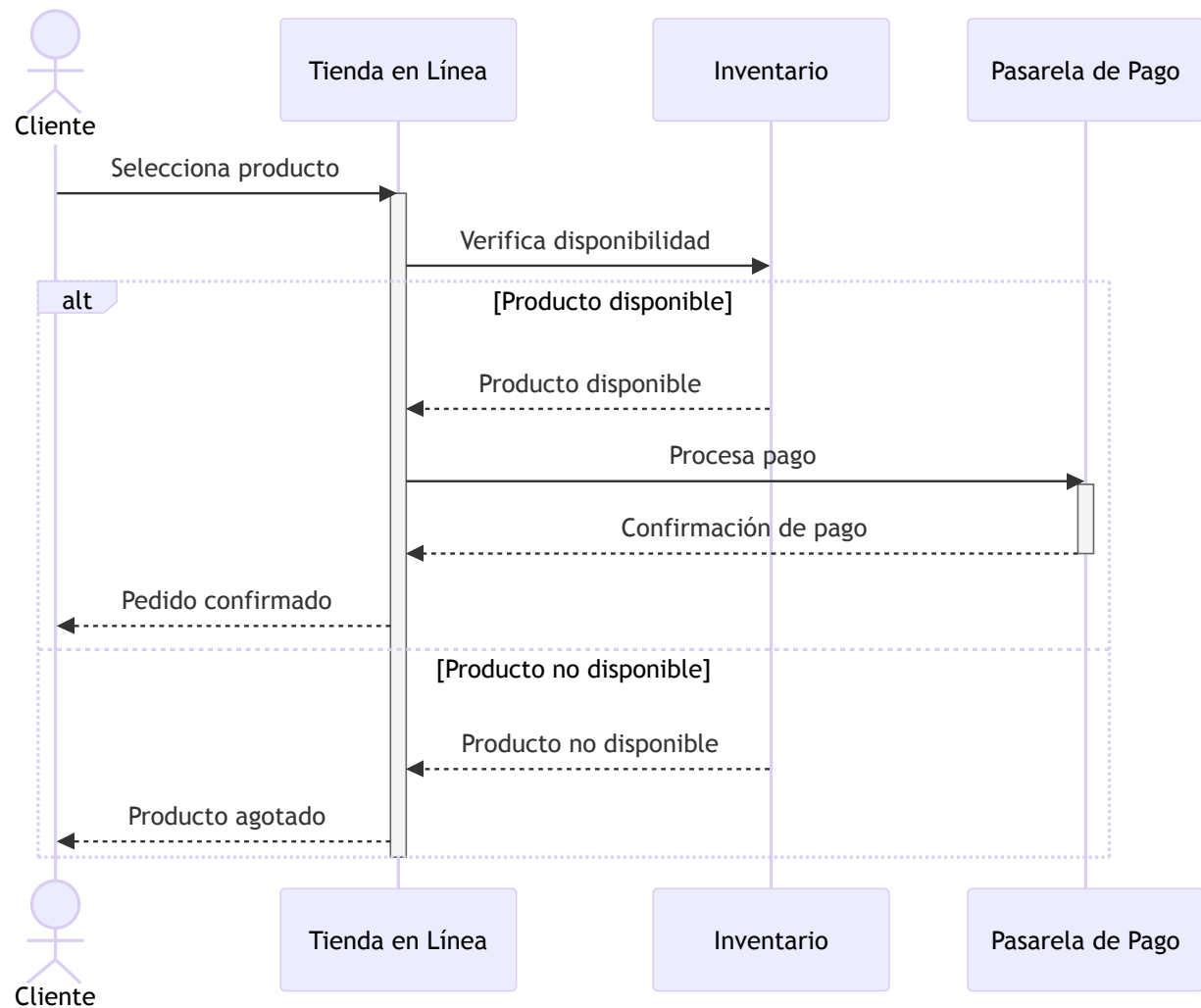
- El cliente puede intentar ingresar sus credenciales hasta tres veces. Si en algún intento las credenciales son válidas, el flujo termina con acceso permitido.

## Combinación de activación y fragmentos

En muchos escenarios, los **bloques de activación** y los **fragmentos combinados** se combinan para modelar interacciones más complejas. Por ejemplo:

**Ejemplo: Compra en línea con verificación de inventario y pago:**





En este ejemplo:

### 1. Bloques de activación:

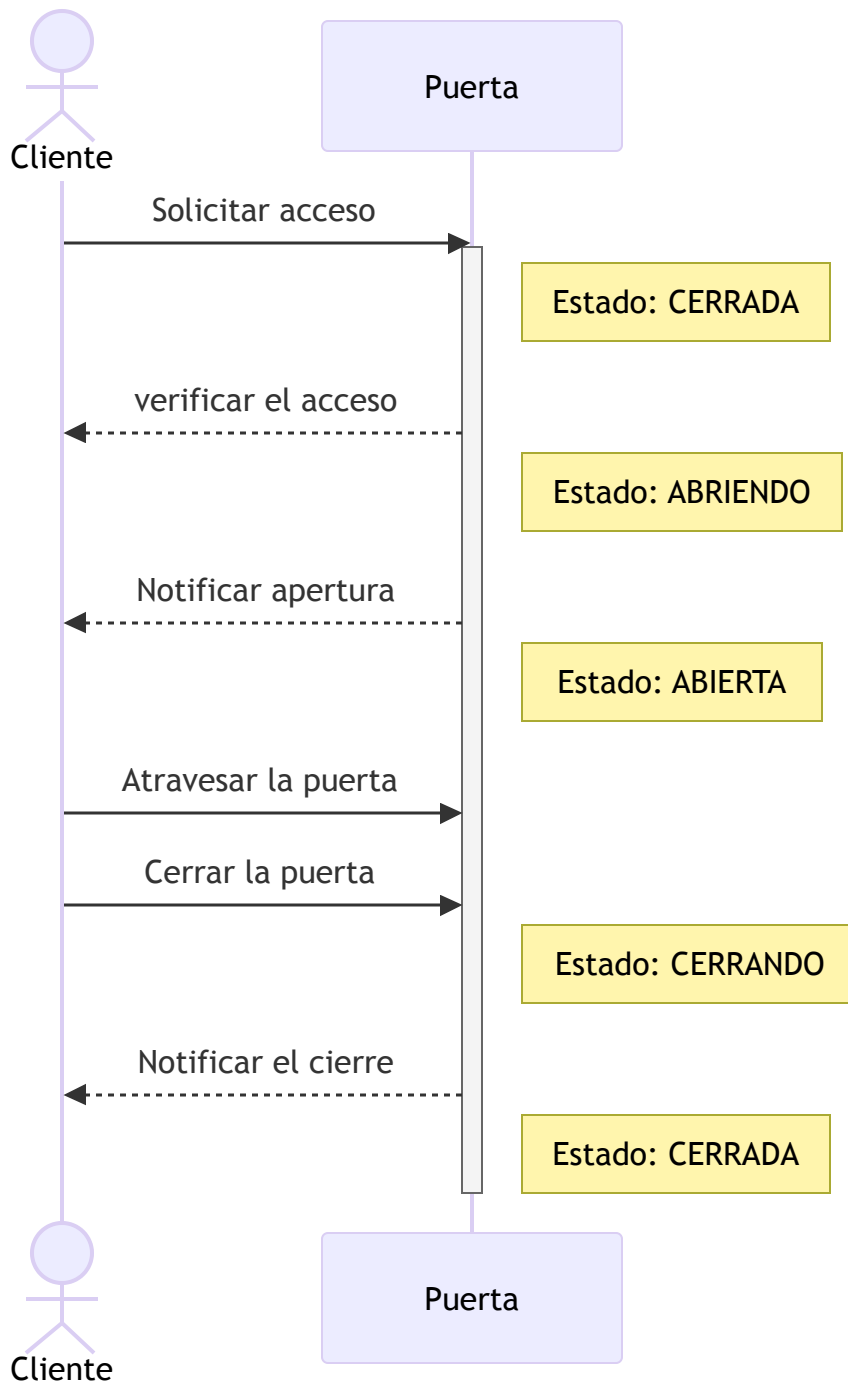
- La **Tienda** está activa mientras verifica el inventario y procesa el pago.
- El **Inventario** y la **Pasarela de Pago** están activos mientras realizan sus tareas específicas.

### 2. Fragmento condicional:

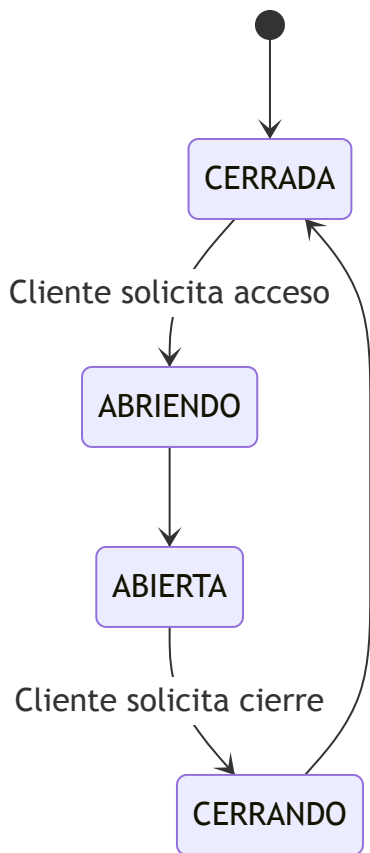
- Si el producto está disponible, se procesa el pago.
- Si no está disponible, se informa al cliente.

## Añadir estados al diagrama de secuencia

Otra práctica que puede ser de utilidad es indicar los cambios de estado que se producen en la línea de vida de un proceso. Estos cambios se pueden indicar dentro de la línea de vida o, como en el siguiente ejemplo, en una nota aparte.



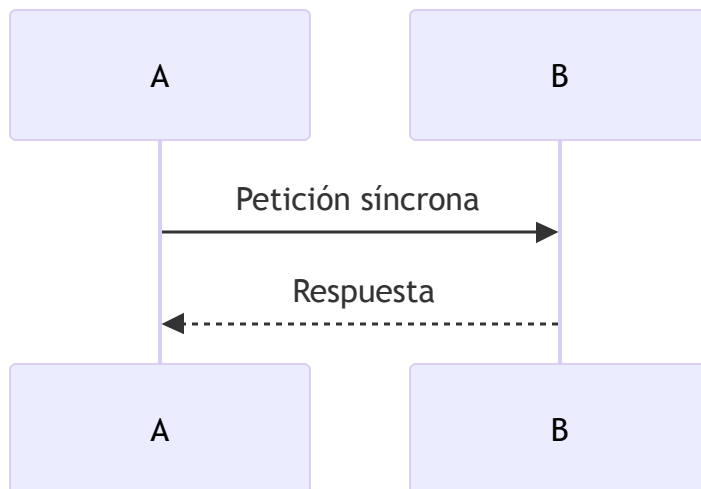
Este sería el diagrama de estados correspondiente de la puerta:



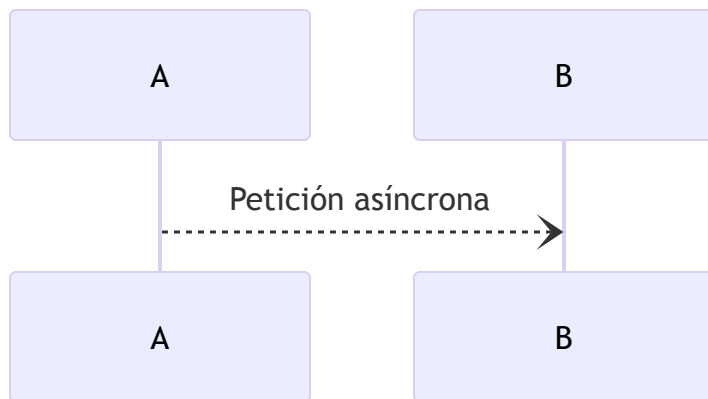
## Mensajes

Las flechas indican la comunicación entre actores/objetos. Los tipos principales son:

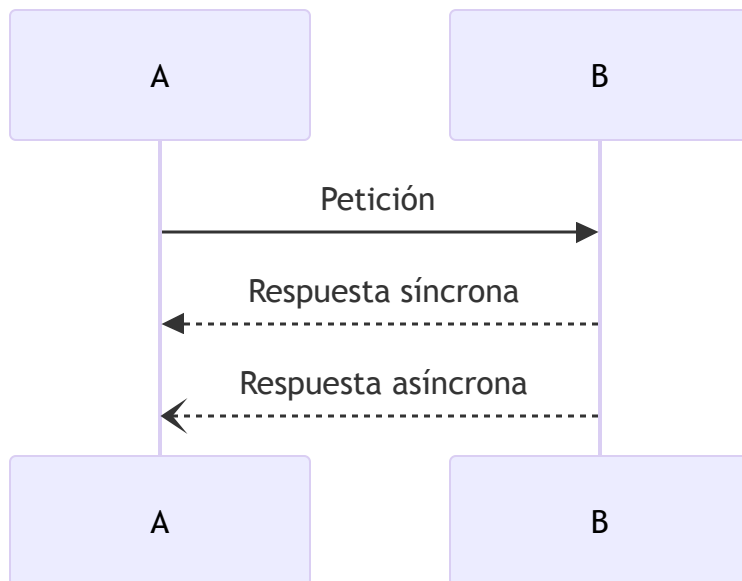
- **Mensajes síncronos:** llamadas o peticiones que esperan una respuesta. Se representan con una flecha normal y una línea continua.



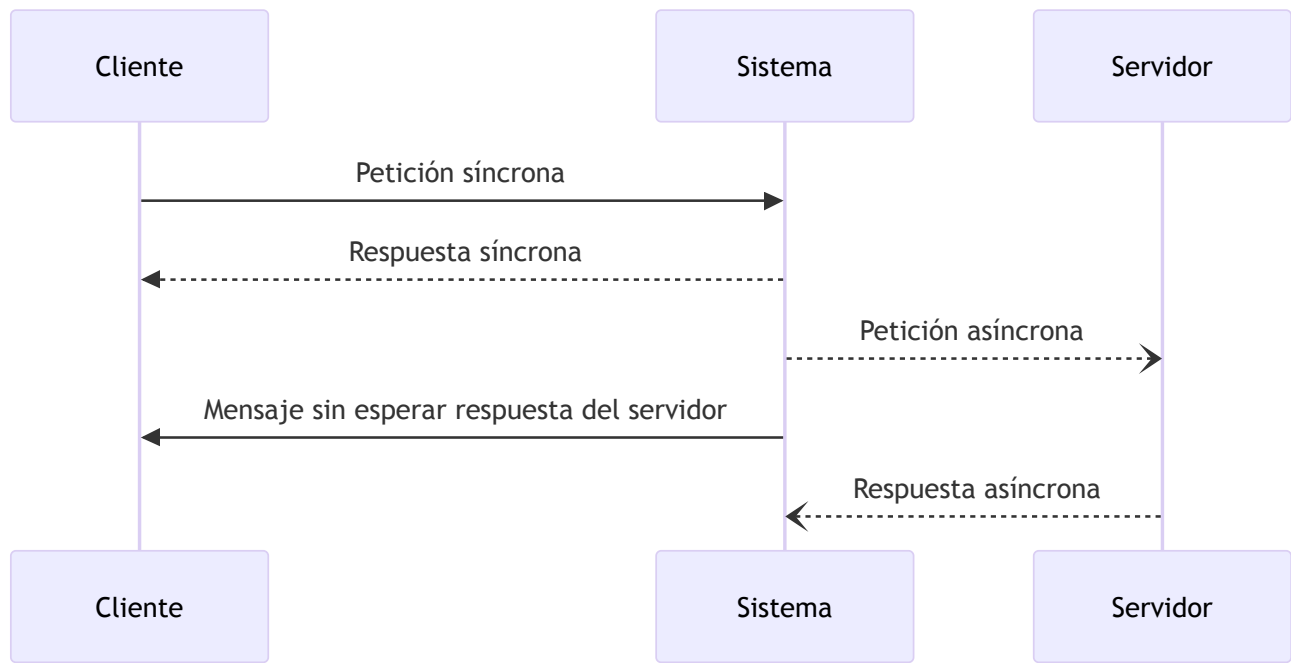
- **Mensajes asíncronos:** peticiones que no esperan una respuesta inmediata. Se representan con una flecha abierta y una línea discontinua.



- **Mensajes de retorno:** respuestas enviadas tras una petición. Se representan con una línea discontinua. La flecha será abierta o cerrada dependiendo de si son retornos asíncronos o síncronos respectivamente



A continuación, un ejemplo con los tres tipos de mensajes:

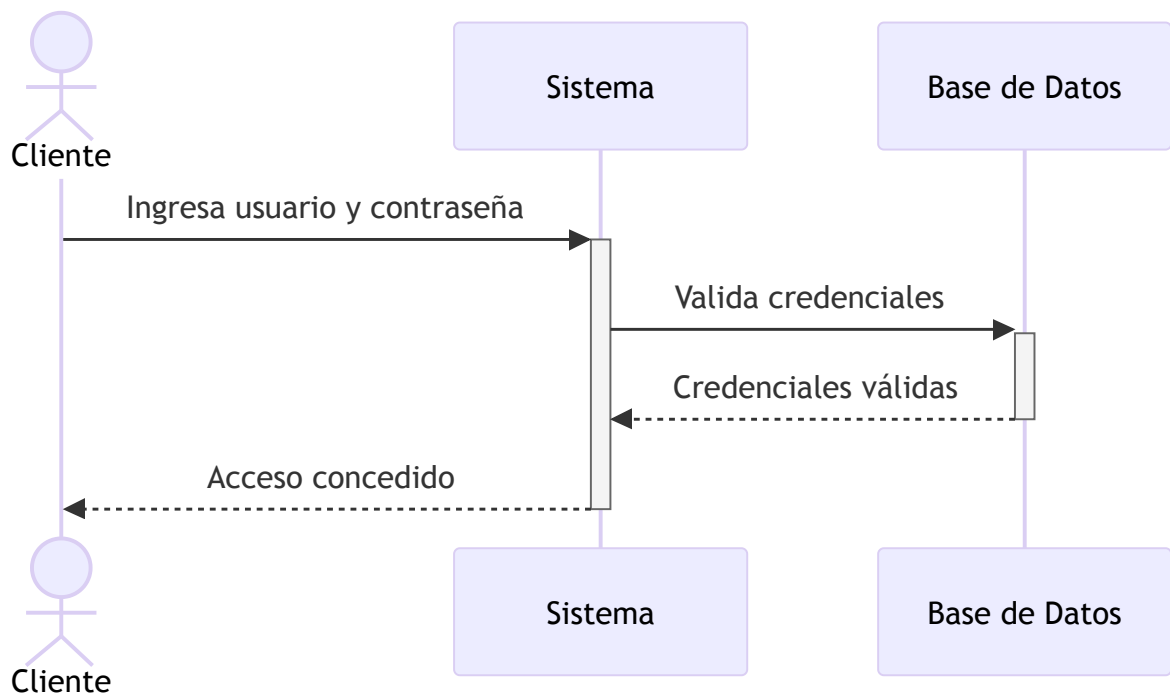


## Construcción de diagramas de secuencia

Para construir un diagrama de secuencia, debemos seguir los siguientes pasos:

1. **Identificar los actores y objetos** que participan en el escenario.
2. **Establecer el flujo de mensajes** entre ellos, respetando el orden cronológico.
3. Agregar activaciones, retornos y fragmentos combinados según sea necesario.

El siguiente ejemplo muestra cómo un cliente interactúa con un sistema de autenticación.



Los diagramas de secuencia son especialmente útiles en casos donde es necesario modelar **escenarios específicos y ordenados de interacción**.

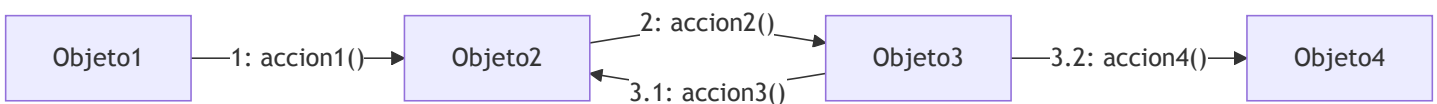
## Diagramas de comunicación

Los diagramas de comunicación modelan las interacciones entre objetos o partes empleando mensajes secuenciados. Representan una combinación de información que puede ser obtenida a través de los diagramas de clase, secuencia y casos de uso, que describen tanto la estructura estática como el comportamiento de un sistema. Asimismo, los diagramas de comunicación son una simplificación del diagrama de colaboración, que era estándar en UML 1.x y fue descartado en UML 2.x.

Los diagramas de comunicación emplean una codificación gráfica similar a la que se usa en los diagramas de objetos. Los elementos se elanzan con líneas y en cada enlace se etiqueta con un número, que representa el orden cronológico del mensaje y el mensaje que se comunica. Los diagramas de comunicación empiezan desde el mensaje 1 y siguen los mensajes de objeto a objeto.

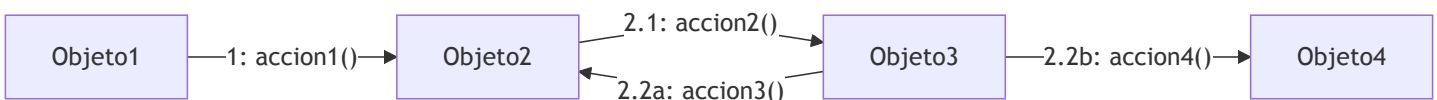


Los mensajes que se producen en la misma llamada tienen el mismo prefijo, pero usan diferentes sufijos según cuando se llaman. Aunque siguen manteniendo el orden, nos permite agrupar mensajes de tal forma que puedan tener más sentido.



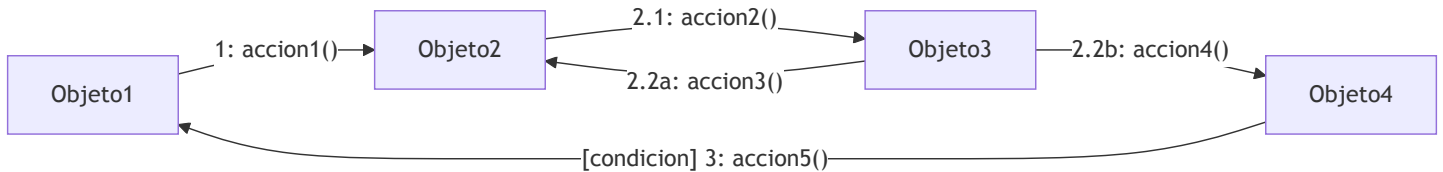
En este ejemplo, los mensajes suceden en orden, pero agrupamos los mensajes `accion3` y `accion4` dentro del mensaje 3. Podemos emplear tantos niveles como queramos para establecer una jerarquía con los mensajes.

Por otro lado, para expresar concurrencia, podemos emplear letras. Si dos mensajes comparten el mismo número, se ejecutarán de forma concurrente o simultánea. Para diferenciarlos, los etiquetaremos con a, b, c, etc.



En este ejemplo, hemos agrupado los mensajes `accion2` , `accion3` y `accion4` dentro del mensaje 2. En el mensaje 2, primero ha sucedido `accion2` y después, de forma simultánea, se han lanzado los mensajes `accion3` y `accion4` .

Finalmente, se pueden asociar mensajes a condiciones, expresadas entre corchetes, lo que nos permite representar estructuras de selección y de bucle.

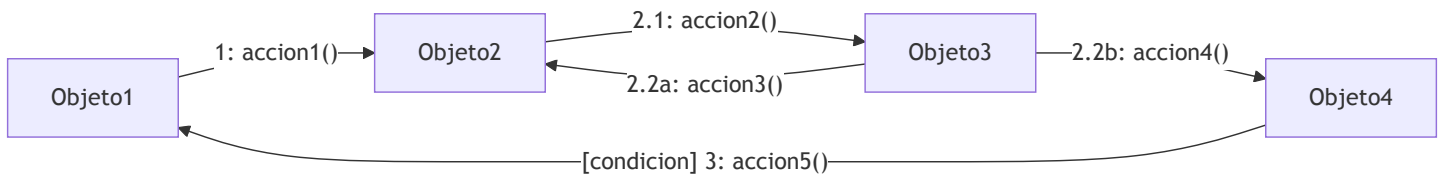


En este ejemplo, el mensaje `accion5` solo se lanzará si se cumple la condición `condicion` .

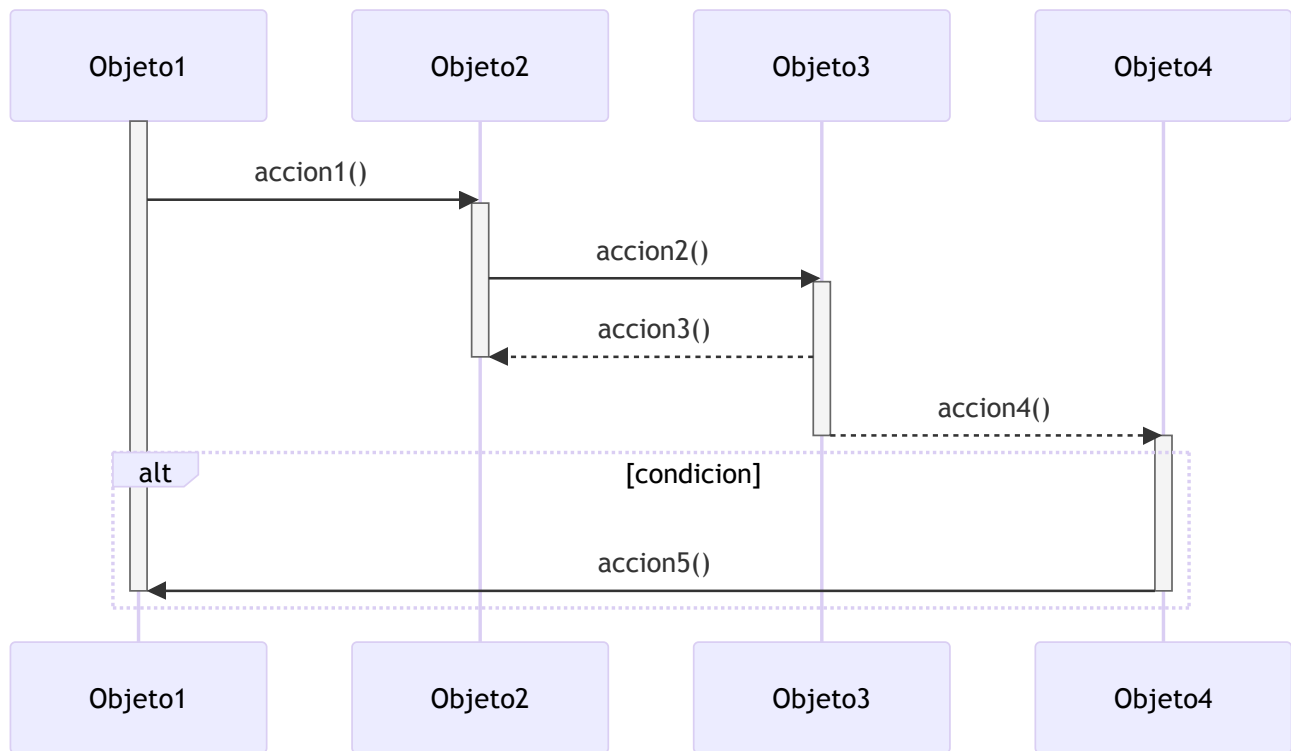
Los diagramas de comunicación muestran la misma información que los diagramas de secuencia, pero su presentación es totalmente diferente, lo que hace que algunos matices sean más evidentes en uno que en otro.

- En los diagramas de comunicación, se entiende fácilmente qué partes interactúan entre sí.
- En los diagramas de secuencia, se entiende fácilmente el orden de las interacciones.

Este diagrama de comunicación:



Es equivalente a este diagrama de interacción:



### Actividad

Crea un diagrama de secuencia y otro de comunicación a partir de la siguiente definición:

Una máquina de café automática gestiona la preparación y entrega de bebidas calientes. En su funcionamiento, la máquina puede encontrarse en varios estados. Al inicio, la máquina está a la espera de que un usuario realice una selección. Cuando el usuario elige una bebida (como café, té o chocolate caliente), la máquina pasa a un proceso de preparación específico según la bebida seleccionada. Si el proceso de preparación concluye sin problemas, la máquina entrega el producto al usuario y regresa al estado inicial para una nueva solicitud.

Sin embargo, si durante la preparación ocurre un error, como falta de ingredientes o una avería técnica, la máquina debe entrar en un estado de error que bloquea nuevas solicitudes hasta que un técnico de mantenimiento intervenga. El técnico podrá reiniciar el sistema y devolver la máquina a su estado inicial. Además, el proceso de **preparación** debe incluir detalles específicos dependiendo del tipo de bebida elegida: preparar café, preparar té o preparar chocolate caliente.



# Reto cooperativo de la Unidad de Programación 02

## Análisis de sistemas y diagramas de comportamiento

### Formación del equipo

Para esta unidad de programación, la formación de los equipos se hará a través del método *Dinámica de los colores*. Observa la tabla:

<b>Egun ona / Buen día</b> Zehatza Precisa Sistematikoa Sistemática Jakingura duena Curiosa, preguntona Analitikoa Analítica Zentzuduna Sensata Saiatua Perseverante Metodikoa Metódica Neurtua Controladora Disziplinatua Disciplinada Egonkorra Estable	<b>Egun txarra / Mal día</b> Ikuspegi itxikoa Estirada, cerrada Zekena Mezquina Erretxina Quisquillosa Aldaezina Inamovible Disoziatua Disociada Hotza Fria Fidekaitza Suspica Kritikoa Crítica Diplomazia gutxikoa Poco diplomática Hunkibera Susceptible	<b>Egun ona / Buen día</b> Ekintzailea Emprendedora Objektiboa Objetiva Tinkoa Decidida Zorrotza Exigente Saiatua Tenaz Helburuetara begira Orientada a objetivos Aktiboa, dinamikoa Enérgica Ordenatua Organizada Erabakitzailea Resolutiva Lehiakorra Competitiva	<b>Egun txarra / Mal día</b> Mendertzailea Dominante Oldarkorra Agresiva Intolerantea Intolerante Harroa Soberbia Pazientzia gutxikoa Impaciente Begirunerik gabekoa Desconsiderada Zakarra Grosera Eskrupulurik gabekoa Sin escrúpulos Sasijakintsua "Sabelotodo" Kontrolatzailea Controladora
<b>Egun ona / Buen día</b> Fidagarria Fiable Adeitsua Atenta Elkarbanatzen du Que comparte Leiala Leal Arretatsua Diligente Pazientziaduna Paciente Ulerkorra Comprensiva Lasaia Tranquila Pentsakorra Considerada Diskrettoa Discreta	<b>Egun txarra / Mal día</b> Burugogorra Obstinada Ernaia Cautelosa Zalantzia Dubitativa Aitzakiz bete Evasiva Adosteko gai ez dena Inconciliable Barnekoia Retraída Uzkurra Reacia Etsigarria Desesperada Sentibera Sensible Isila Reservada	<b>Egun ona / Buen día</b> Sinesgarria Convincente Hitz errazekoa Extrovertida Gogotsua Entusiasta Baikorra Optimista Lagunkoia Sociable Dinamikoa Dinámica Irekia Comunicativa Sortzailea Creativa Bat-batekoa Espontánea Burujahea Independiente	<b>Egun txarra / Mal día</b> Oldarkorra Impulsiva Oso urduria Sobexcitada Frenetikoa, gogorra Agitada Gehiegizkoa Exagerada Diskrezio gutxikoa Indiscreta Nabarmena Extravagante Zaratatsua, ozena Llamativa, ruidosa Azalekoa Superficial Axolagabea Descuidada Ordenik gabekoa Desorganizada

¿Con qué colores te identificas más? Escoge en la encuesta en FORMS tu primera y tu segunda opción. En base a ella, se crearán los equipos de la clase intentando tener representados todos los colores.

## Tarea

Lee el enunciado que tu equipo tiene asignado y, según los criterios del IEEE 830, establece los requisitos funcionales de cada aplicación. A continuación, basándote en dicha información, realiza los siguientes diagramas:

- Diagrama de casos de uso
- Diagrama/s de transición de estados
- Diagrama/s de actividad

- Diagrama/s de interacción

# Reto individual

## Crea los diagramas de comportamiento necesarios según la especificación

Lee el enunciado de esta problemática y establece, según los criterios del IEEE 830, los requisitos funcionales la aplicación que se te pide desarrollar. A continuación, basándote en dicha información, realiza los siguientes diagramas:

- Diagrama de casos de uso
- Diagrama/s de transición de estados
- Diagrama/s de actividad
- Diagrama/s de interacción

Se requiere diseñar un sistema para **gestionar el control de equipaje** en un aeropuerto. En el aeropuerto interactúan **tres tipos de usuarios: guardias, personal del aeropuerto y pasajeros**. Los datos de identificación de cada usuario son los siguientes:

- Los **guardias** y el **personal del aeropuerto** se identifican por su **nombre** y **DNI**.
- Los **pasajeros** se identifican por su **nombre**, **DNI** y el **número de vuelo** al que pertenecen.

El sistema debe controlar el flujo de equipaje bajo las siguientes **condiciones**:

1. Todo **equipaje** pasa por **áreas de control**. Estas áreas pueden estar en alguno de los siguientes estados: **operativas, bloqueadas o en mantenimiento**.
2. Los **guardias** pueden acceder a todas las áreas de control, independientemente de su estado.
3. El **personal del aeropuerto** puede **bloquear** áreas de control (por mantenimiento, seguridad, etc.), y mientras estén bloqueadas, no se puede procesar ningún equipaje.
4. Los **pasajeros** solo pueden acceder a áreas de control **operativas y asignadas a su vuelo**.
5. Un equipaje solo puede ser procesado si el área de control está **operativa y no bloqueada**.
6. Algunas áreas de control son **prioritarias** y permiten el procesamiento de equipaje **especial**.  
Dichas áreas deben ser autorizadas por un **guardia o personal del aeropuerto**.

Procesar un equipaje significa pasarlo por el control. En caso de que salte la alarma, el guardia debe revisar de forma manual el equipaje para detectar qué elemento hacía saltar el control y volver a poner el equipaje para pasarlo de nuevo, hasta que no salte la alarma o el pasajero desista. Por su

parte, mientras se está procesando el equipaje, el pasajero debe pasar por el control de metales. Si salta la alarma, debe quitarse la ropa que hacía saltar la alarma, ponerla en el control como si fuera equipaje y volver a comenzar el proceso hasta que todo sea correcto o desista.

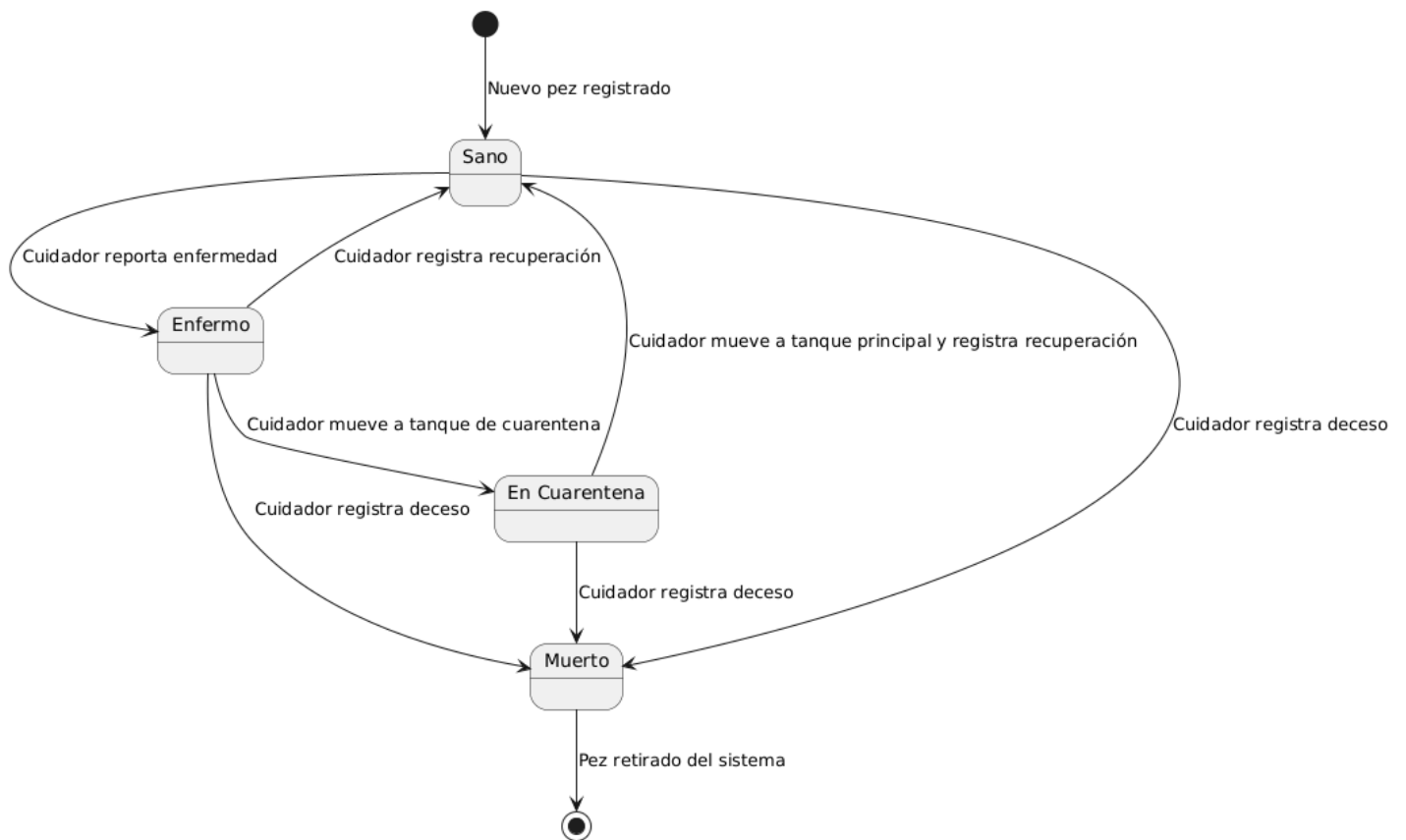
## Interpreta los siguientes diagramas y define el sistema representado por los mismos

### Diagrama de casos de uso



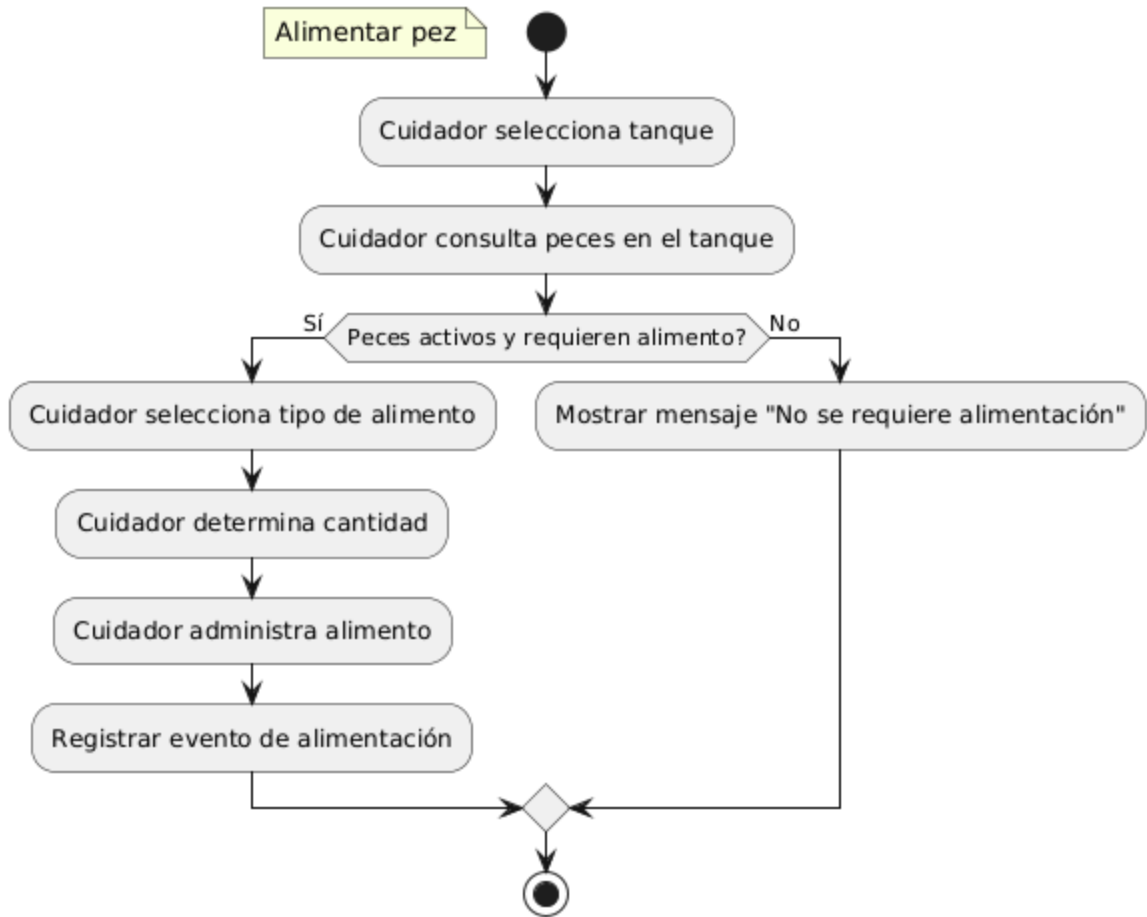
► Haz click aquí para ver el código

# Diagrama de transición de estados



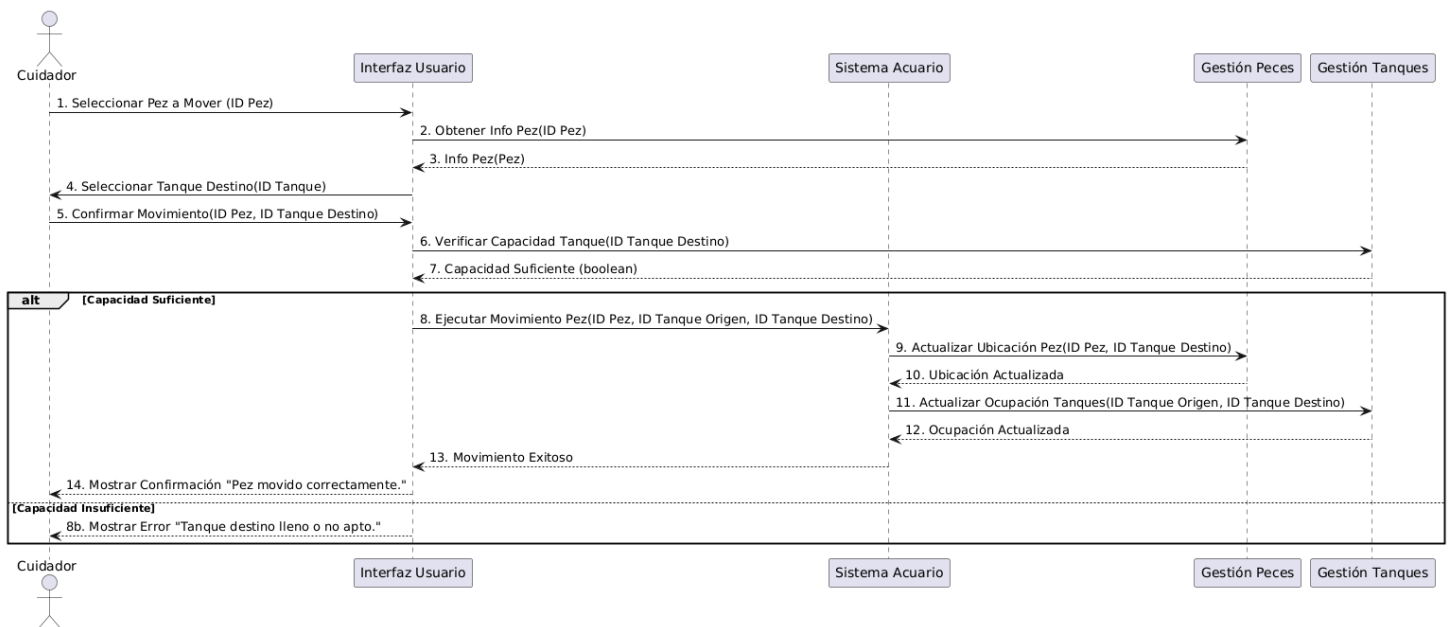
► [Haz click aquí para ver el código plantuml](#)

# Diagrama de actividad



► Haz click aquí para ver el código plantuml

# Diagrama de secuencia



► Haz click aquí para ver el código plantuml



## ANEXO VII: EJEMPLO DE PRUEBA OBJETIVA DE LA EVALUACIÓN DE LA UNIDAD DE PROGRAMACIÓN 2

### ANEXO VII-1: EJERCICIO 1: ESPECIFICACIÓN

En el ejercicio de especificación, se facilita al alumnado un enunciado como el que sigue, así como la rúbrica de evaluación, y se evalúa siguiendo los criterios de evaluación 6a, 6b, 6d, 6f y 6h. A continuación, el enunciado:

Se te ha encargado diseñar una aplicación para la **gestión de envíos de una empresa**. Cada envío se compone de diversos **paquetes**, unos datos de envío y unos datos de facturación, almacenados cada uno de ellos en un registro diferente (una sola variable de tipo “dirección”). Cada paquete se compone de una **caja** y diversos **productos** que van dentro. Las cajas tienen información acerca de su tamaño (en tres dimensiones) y si son aptas para envíos frágiles o no. Los productos son de muchos tipos, pero la información que nos interesa es su tamaño (en tres dimensiones), si son frágiles o no y su id de producto.

Los envíos se procesan en la aplicación y pasan por diferentes **estados**. Cuando un cliente hace un pedido y lo paga, el envío pasa a estar en proceso, que es cuando se consiguen los productos y se empaquetan. Una vez listo, pasa a estar listo para el envío, hasta que lo recoge el transportista. Hasta ese punto, el cliente puede cancelar el envío. Cuando el transportista lo recoge, el paquete pasa a estar en tránsito hasta que llega a su destino. Si va a un punto de recogida, se queda en espera de ser recogido. Si no, o cuando es recogido, pasa a estar recogido.

El cliente puede entonces gestionar una devolución en un plazo de quince días. Un paquete, cuando pasa a estar en proceso de devolución, es recogido por el transportista y enviado al almacén. Una vez en el almacén, tanto los pedidos cancelados como los pedidos devueltos son recolocados. Por otro lado, una vez pasados esos 15 días, algunos productos tienen una garantía de varios años (es decir, están en garantía) y, si surge algún problema, se les puede hacer un proceso equivalente al de la devolución. Es importante llevar el seguimiento del estado del paquete.

**La función del programa** es admitir los pedidos de los clientes, gestionar los procesos de los envíos, documentar las posibles incidencias e inventariar todos los envíos de la empresa, almacenándolos en una base de datos relacional que diseña otro equipo. El administrador de la aplicación puede, además, consultar la información de los envíos o cancelar algunos envíos, notificando al cliente, si hubiera algún problema.

## ANEXO VII -2: RÚBRICA DEL EJERCICIO 1

Ítem	Criterio de Evaluación	Objetivo	Valor	Resultado
Especificación de requisitos y Diagrama de casos de uso de la aplicación	6a, 6b	Ñ, S, T	20%	
Diagrama de estados de un envío	6a, 6h	Ñ, S, T	20%	
Diagrama de secuencia del proceso de cancelación por parte del administrador	6a, 6d	Ñ, S, T	10%	
Diagrama de comunicación del proceso de cancelación por parte del cliente	6a, 6d	Ñ, S, T	10%	
Diagrama de actividad del proceso de devolución	6a, 6f	Ñ, S, T	20%	

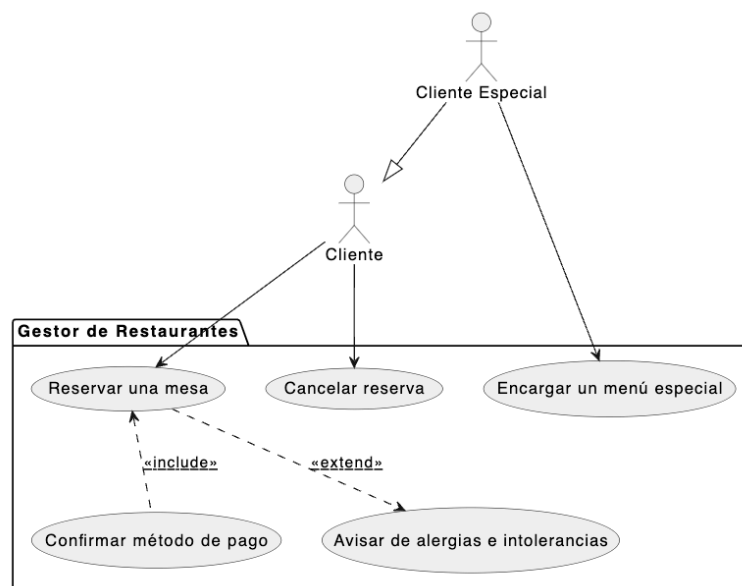
## ANEXO VII-3: EJERCICIO 2: TEST

En el ejercicio de tipo test se elaboran 20 preguntas cubriendo los Criterios de evaluación 6a, 6b, 6g, 6c y 6e. A continuación, se presenta una muestra con 5 preguntas, una por criterio de evaluación a tratar. Las respuestas correctas están en negrita.

(CE 6a) ¿Cuál de estos diagramas no es un diagrama UML de comportamiento?

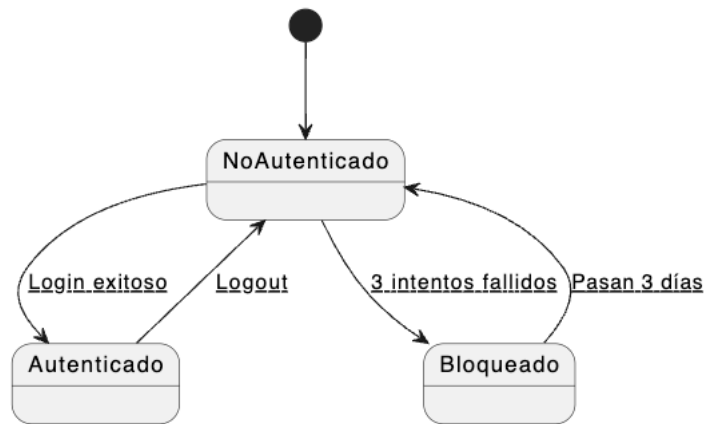
- a) El diagrama de comunicación global
- b) El diagrama de Tiempos
- c) El diagrama de Paquetes**
- d) El diagrama de Casos de Uso

(CE 6b) ¿Cuál de estas afirmaciones es falsa sobre el siguiente diagrama de casos de uso?



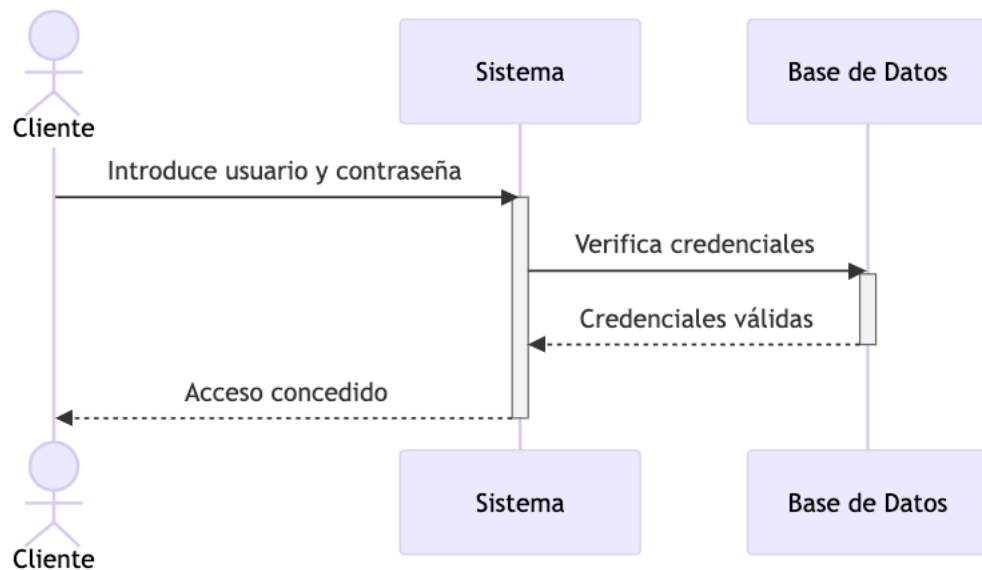
- a) El cliente especial puede reservar una mesa
- b) El cliente no puede encargar un menú especial
- c) Al reservar una mesa, es obligatorio avisar de alergias e intolerancias**
- d) Al reservar una mesa, es obligatorio confirmar un método de pago

(CE 6g) Dado el siguiente diagrama de estados, ¿cuál de las siguientes afirmaciones es cierta?



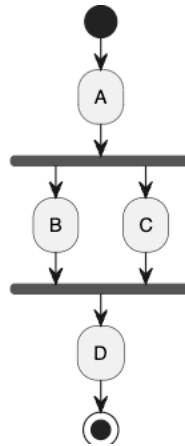
- a) Cualquier estado puede ser el inicial
- b) Autenticado es una transición
- c) Si un usuario intenta hacer login 2 veces sin éxito, es bloqueado
- d) **A los tres días, un usuario bloqueado se desbloquea**

**(CE 6c) Dado el siguiente diagrama de interacción, determina qué afirmación es falsa**



- a) La base de datos está activa todo el tiempo
- b) El cliente inicia el proceso
- c) Las respuestas acceso concedido y credenciales válidas son asíncronas
- d) El sistema despierta cuando el cliente introduce usuario y contraseña

(CE 6e) Dado el siguiente diagrama de actividad, ¿qué afirmación es cierta?



- a) La acción B y la acción C suceden a la vez
- b) Hasta que no hayan terminado las acciones A, B y C no puede empezar la D**
- c) La acción A y la acción D suceden en paralelo
- d) Ninguna de las anteriores

# Cuaderno de ejercicios 02

- Cuaderno de ejercicios 02
  - Diagramas de casos de uso
  - Diagramas de transición de estados
  - Diagramas de actividad
  - Diagramas de interacción
  - Prompts de IA para practicar
    - Prompts de la IA para aprender un lenguaje de programación o de modelado como Plantuml
    - Prompts de la IA para practicar preguntas de tipo test
    - Prompts para ejercicios

## Diagramas de casos de uso

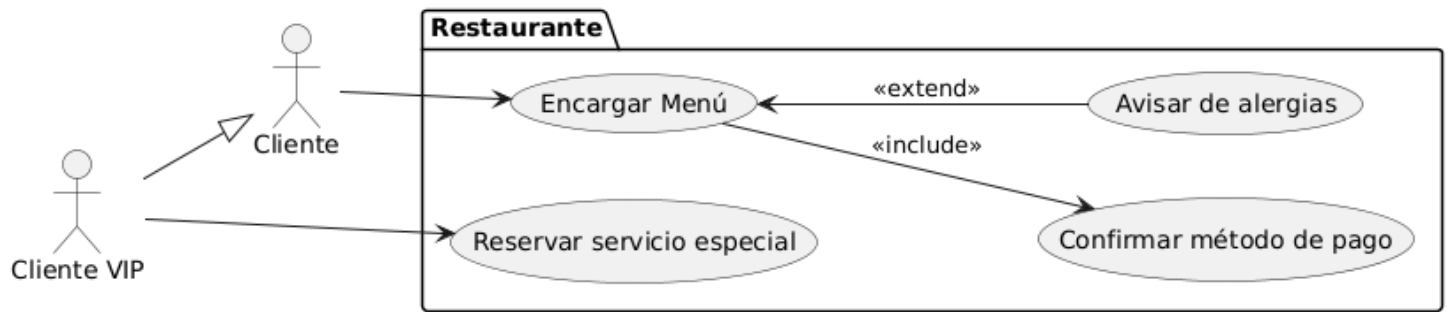
**Actividad:** Describe el funcionamiento de los siguientes sistemas según su diagrama de casos de uso.

**Diagrama 1:**



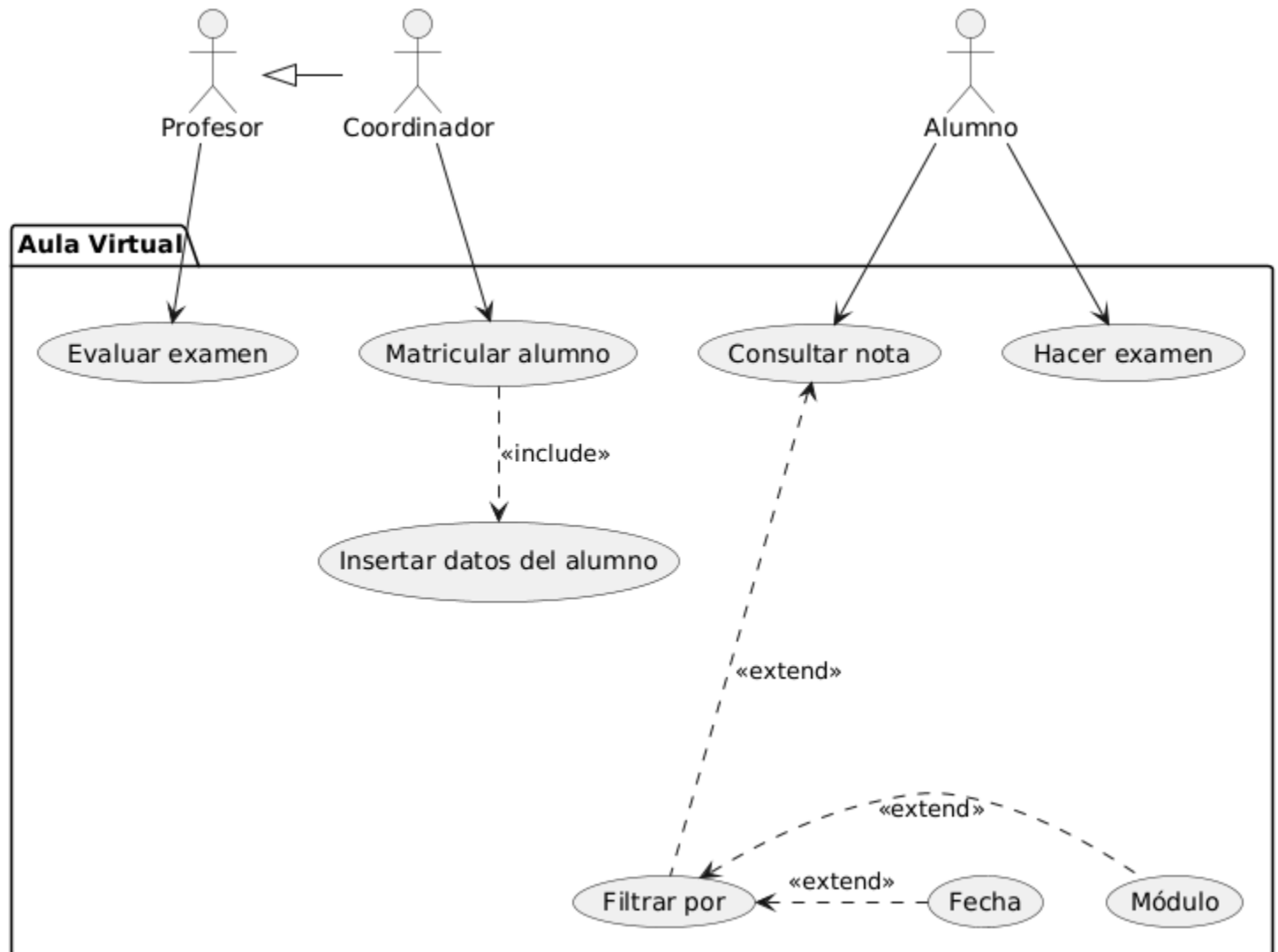
► Ver el código en plantuml

**Diagrama 2:**



► [Ver el código en plantuml](#)

Diagrama 3:



► [Ver el código en plantuml](#)

## Diagramas de transición de estados

**Actividad:** Identifica el significado de los siguientes diagramas de estados. Para ello, sigue los siguientes pasos:



- ¿Qué representa cada estado?
- ¿Qué significado tienen las transiciones?
- ¿Qué proceso o sistema modela el diagrama?

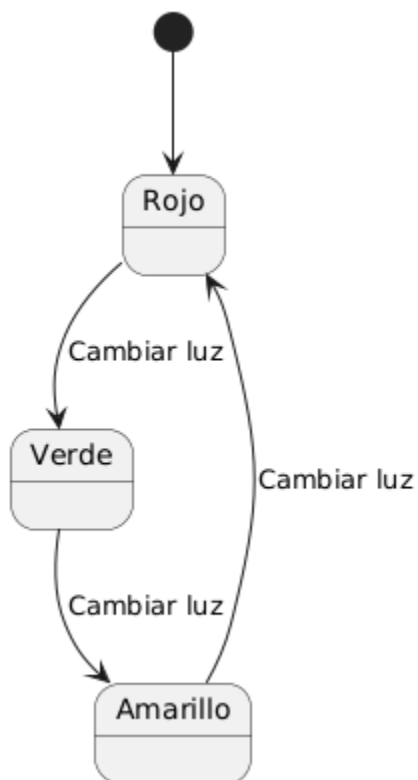
Finalmente, describe brevemente cada diagrama, explicando su **función y aplicación en el mundo real**.

**Diagrama 1: Ciclo de vida de una tarea**



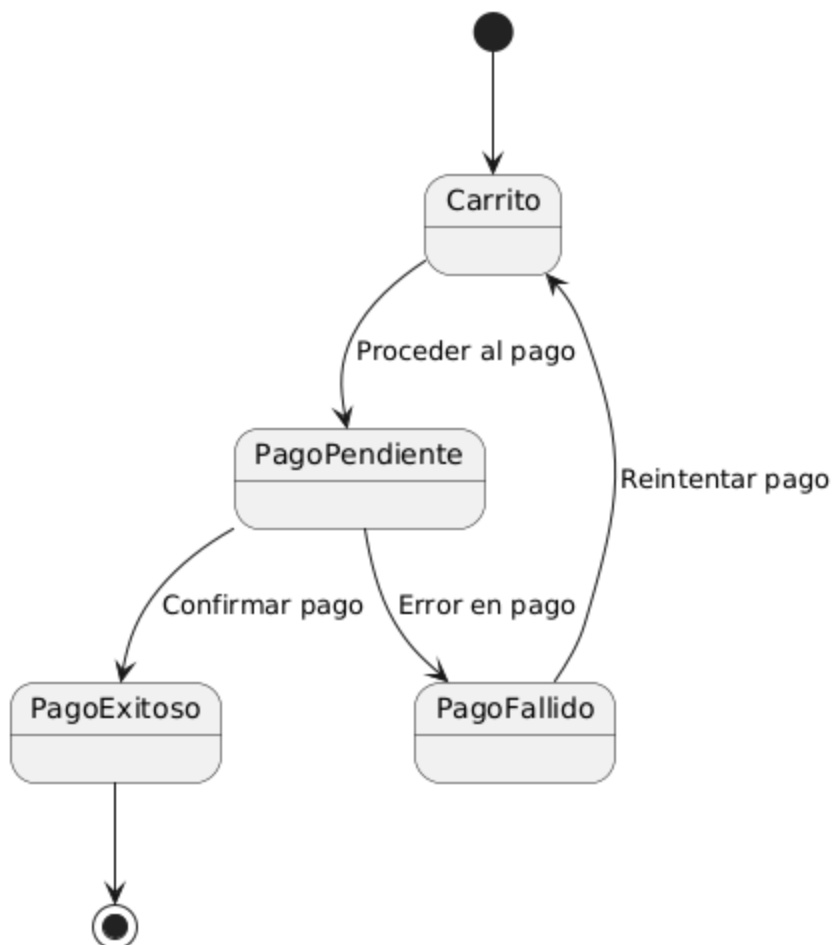
► [Ver el código en plantuml](#)

**Diagrama 2: Sistema de un semáforo**



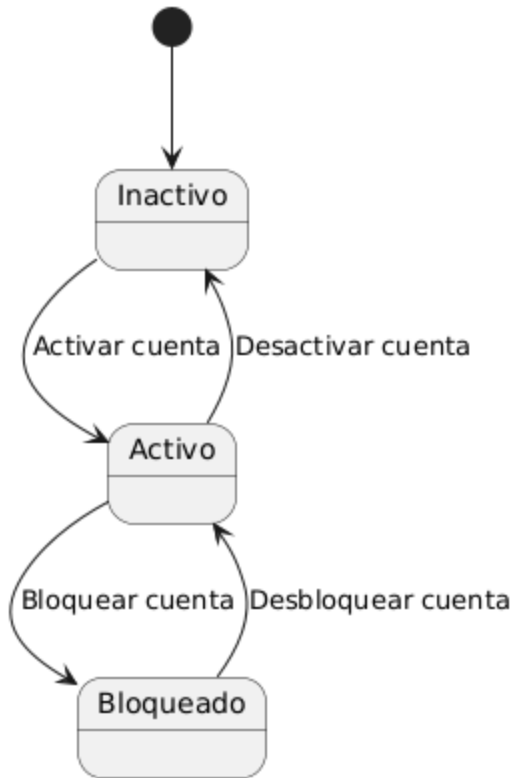
► [Ver el código en plantuml](#)

**Diagrama 3: Proceso de pago en línea**



► [Ver el código en plantuml](#)

Diagrama 4: Gestión de una cuenta de usuario



► [Ver el código en plantuml](#)

**Actividad:** Describe los siguientes diagramas de estados siguiendo estas pautas:

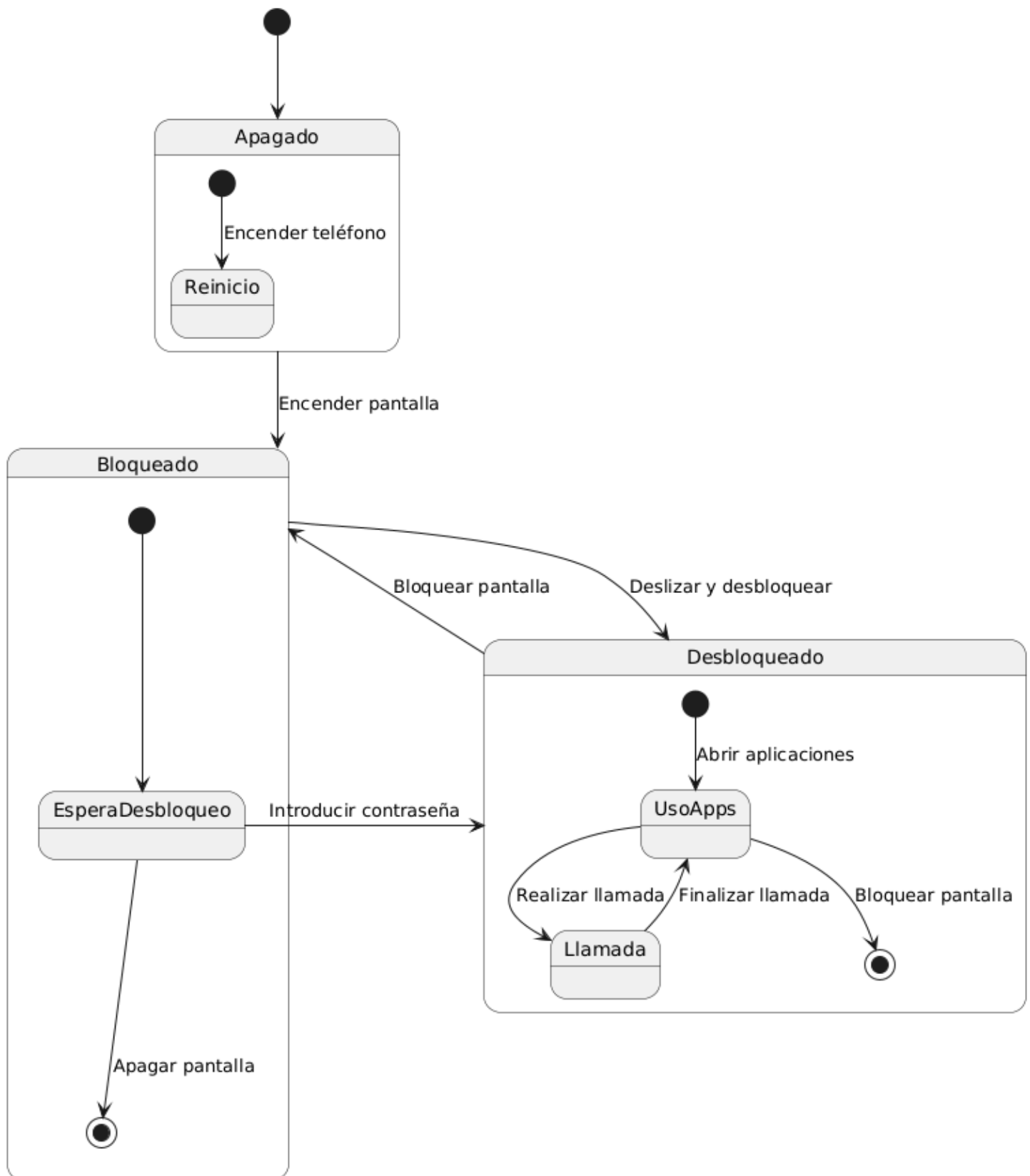
1. **Analiza cada diagrama compuesto:**

- Identifica los **estados principales** y los **subestados**.
- Describe el flujo general del sistema y qué proceso representa.

2. **Escribe una breve explicación** para cada diagrama, enfocándote en:

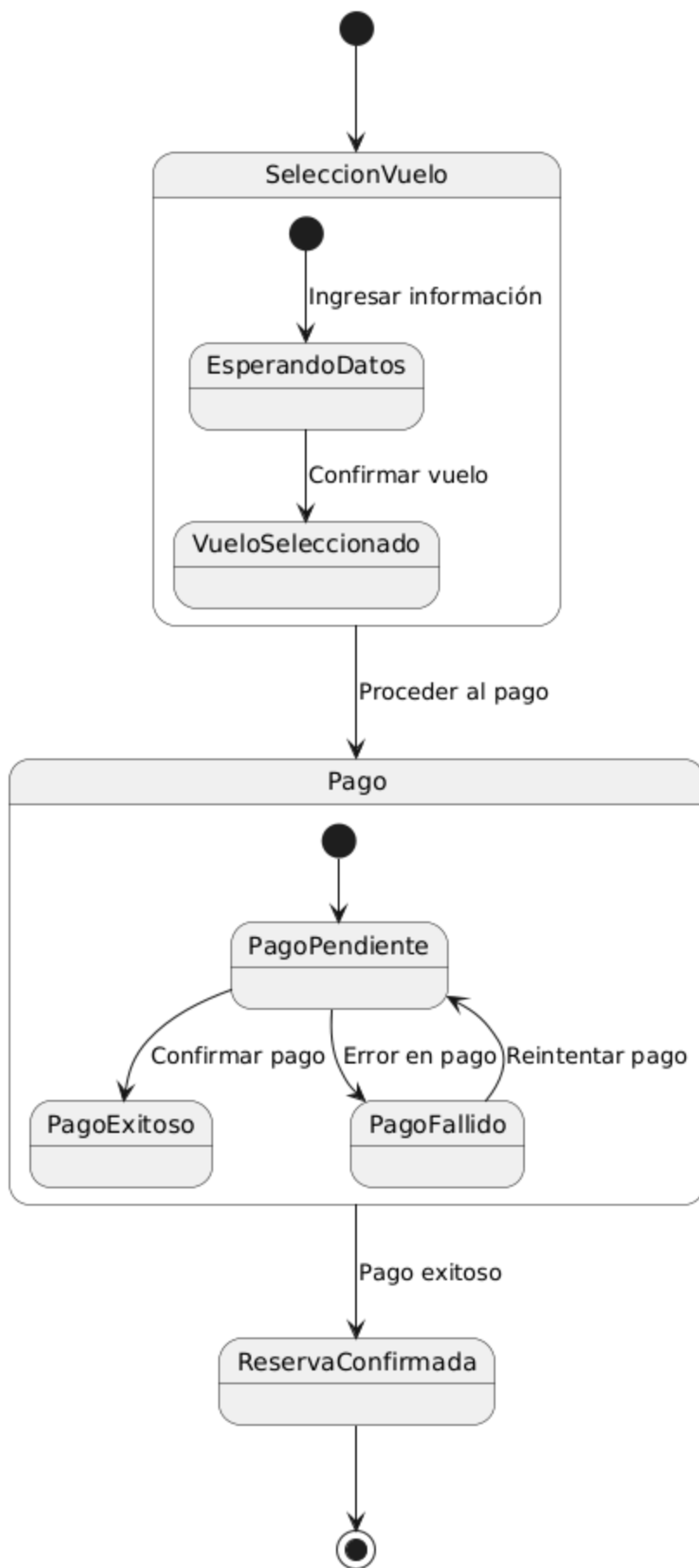
- **Qué sistema modela** (por ejemplo: máquina expendedora, proceso de reserva de vuelo).
- **Cómo los subestados mejoran la comprensión del sistema.**

Ciclo de vida de un teléfono móvil

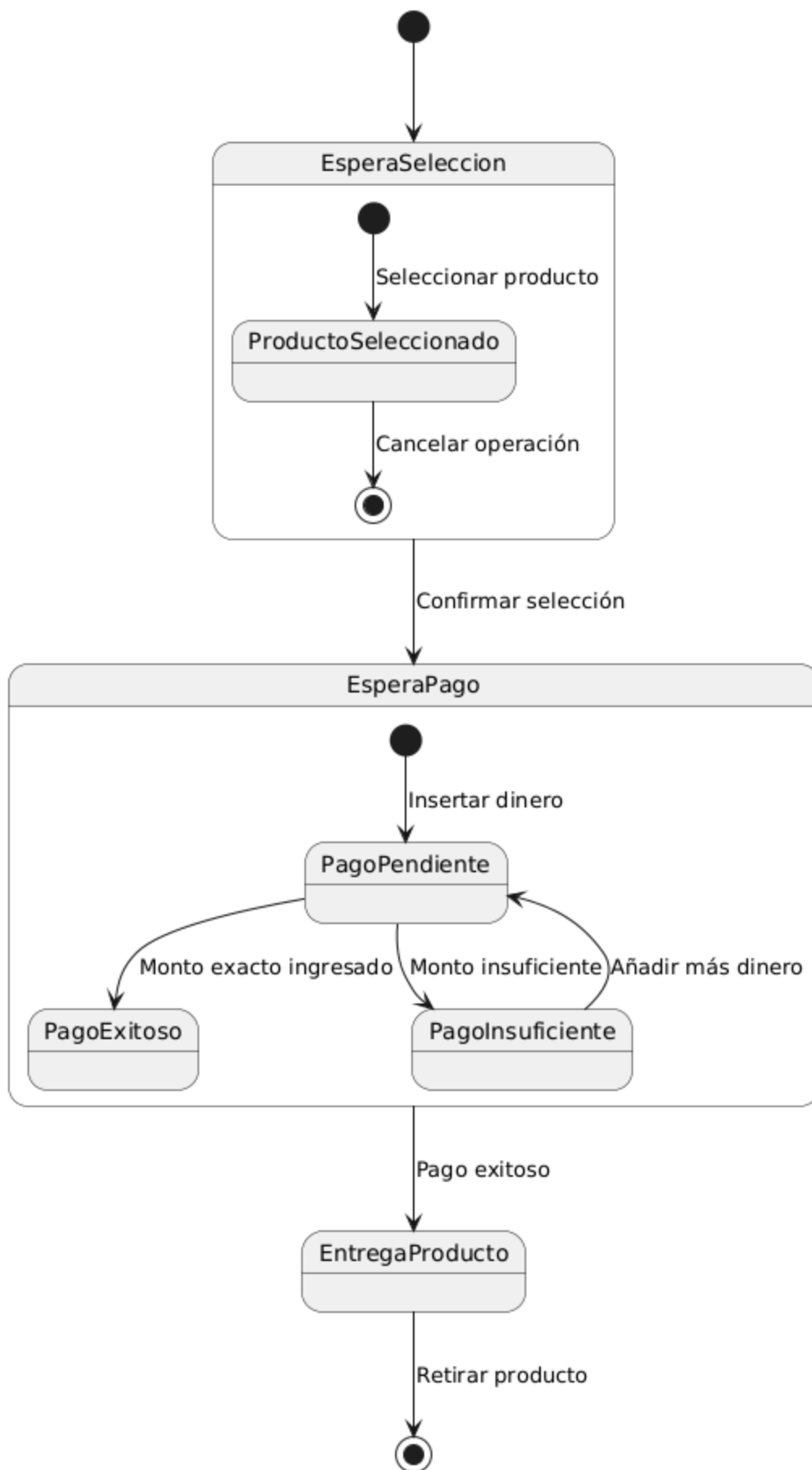


► Ver el código en plantuml

Proceso de reserva de un vuelo



► [Ver el código en plantuml](#)



# Diagramas de actividad

**Actividad:** Dibuja un diagrama de flujo que determine si un número es positivo, negativo o cero.

**Actividad:** Crea un diagrama de flujo para sumar todos los números entre dos números entrados por el usuario en un bucle. Comprueba cuál de los dos números es mayor para establecer el orden de forma correcta. Si el resultado tiene una cifra, escribe "una cifra", si tiene dos escribe "dos cifras" y si tiene más escribe "Tres o más cifras".

**Actividad:** Realiza un diagrama de actividad de cada una de las siguientes actividades:

- Determinar la cantidad de días que tiene un mes en un año, ambos datos introducidos por el usuario. Los datos se imprimen por pantalla. El programa termina si el usuario introduce un número de mes incorrecto (menor que 1 o mayor que 12).
- Un programa que calcule el factorial de un número entero y mayor o igual a 1 introducido por el usuario. La fórmula del factorial es  $\text{Factorial}(n) = n * \text{Factorial}(n-1)$  y  $\text{Factorial}(1) = 1$ . Hazlo de forma iterativa (sin recursividad).
- Un programa que recorra las estanterías de una biblioteca y diga la siguiente información:
  - Qué estantería tiene más libros (están numeradas del 1 al 10)
  - Cuál es el título del libro que más páginas tiene y en qué estantería está
  - Cuántas páginas hay en cada estantería

**Actividad:** Dibuja un diagrama de flujo para calcular el área de varios triángulos introducidos por el usuario. Define un subproceso para calcular el área de un triángulo e invócalo en el momento adecuado.

**Actividad:** Coordinación de Tareas en una Cocina

En una cocina, varios trabajadores deben colaborar para preparar y entregar hamburguesas según los pedidos de los clientes. Cada trabajador tiene una tarea específica, y el pedido solo puede ser entregado cuando todos hayan terminado su parte. El flujo de trabajo se divide en las siguientes actividades:

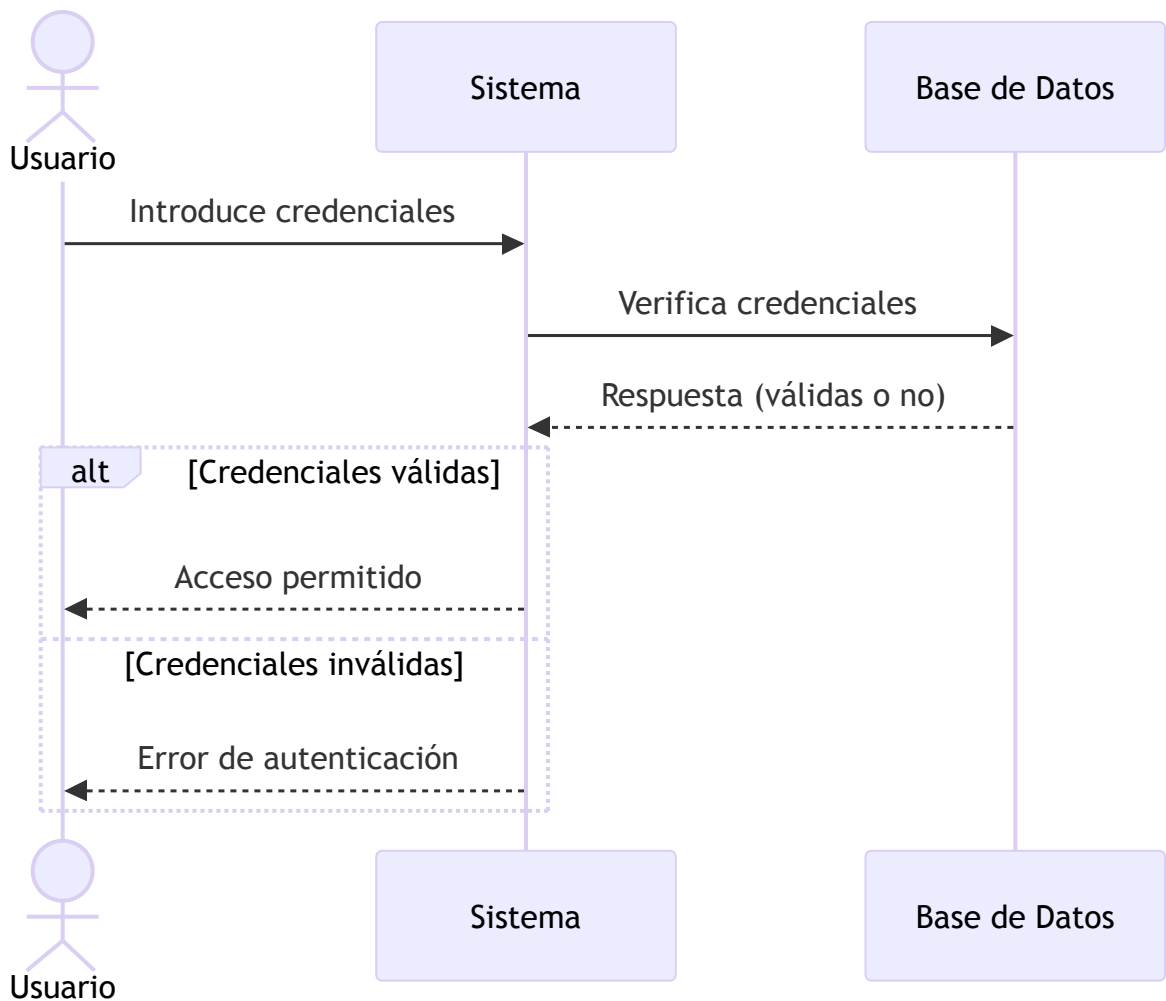
1. **Cocinero 1:** Cocina la carne de la hamburguesa.
2. **Cocinero 2:** Prepara el pan (lo corta y lo coloca).
3. **Cocinero 3:** Añade los condimentos (queso, salsas, etc.). Solo se puede hacer si el pan está preparado.
4. **Cocinero 4:** Prepara la guarnición (patatas fritas, ensaladas, etc.).
5. **Trabajador de entrega:** Ensambla el pedido y lo entrega al cliente, pero **solo puede hacerlo cuando todos los cocineros hayan terminado sus tareas.**

Modela el sistema de preparación de pedidos de hamburguesas mediante un **diagrama de actividad**, representando los flujos paralelos de trabajo.

## Diagramas de interacción

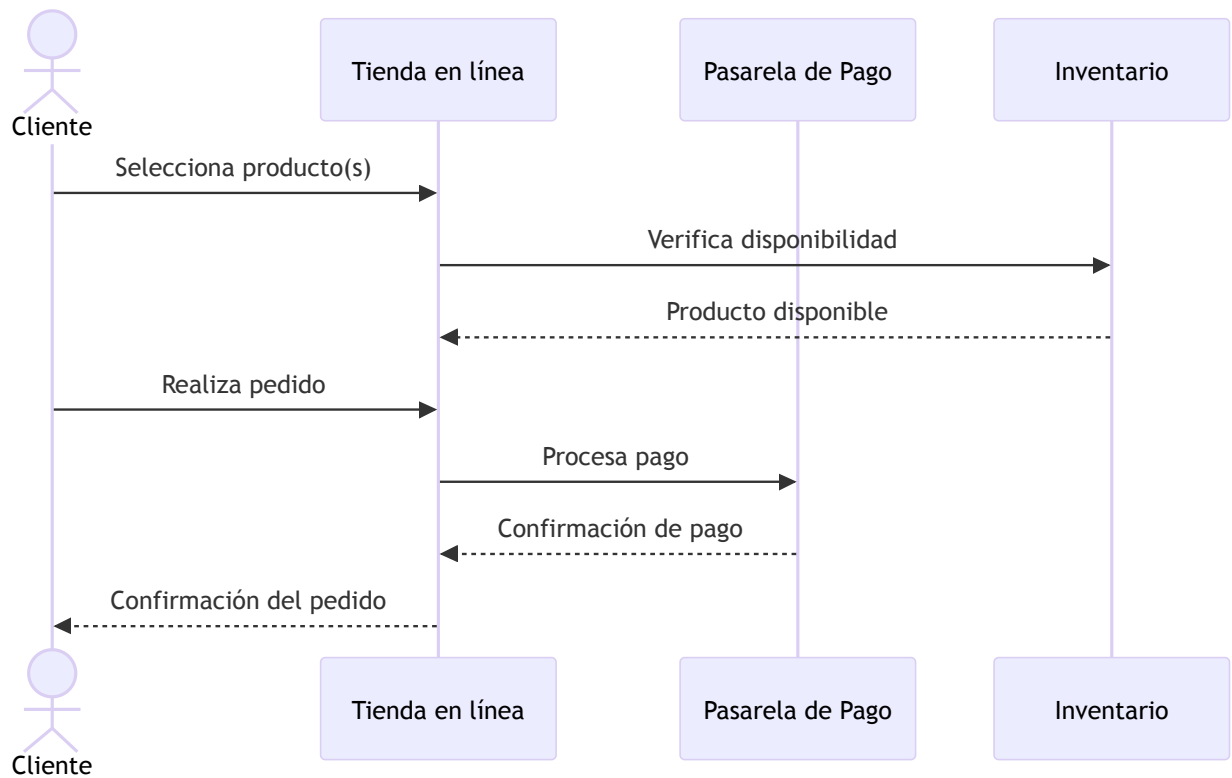
**Actividad:** Explica lo que ocurre en los siguientes diagramas de secuencia:

### Autenticación de un usuario

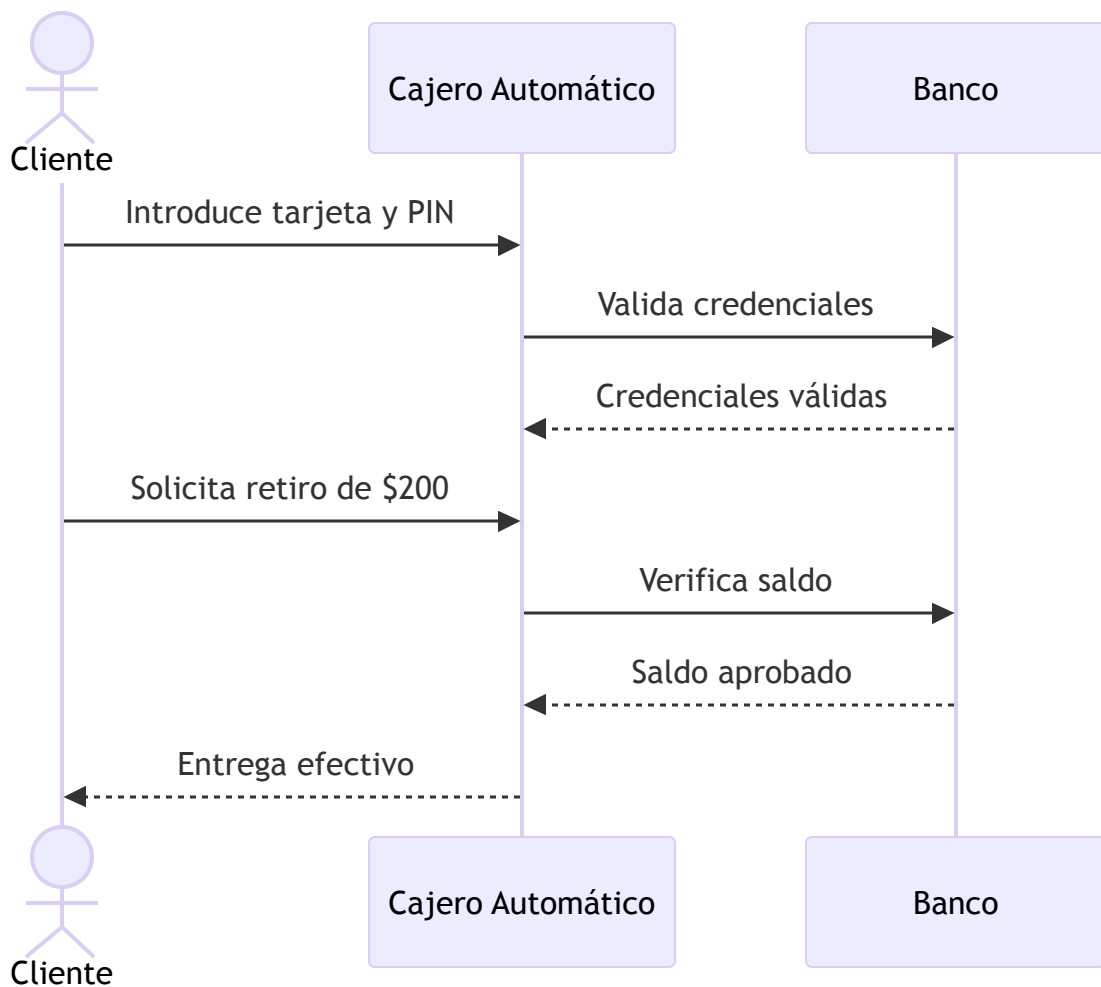


### Compra en línea

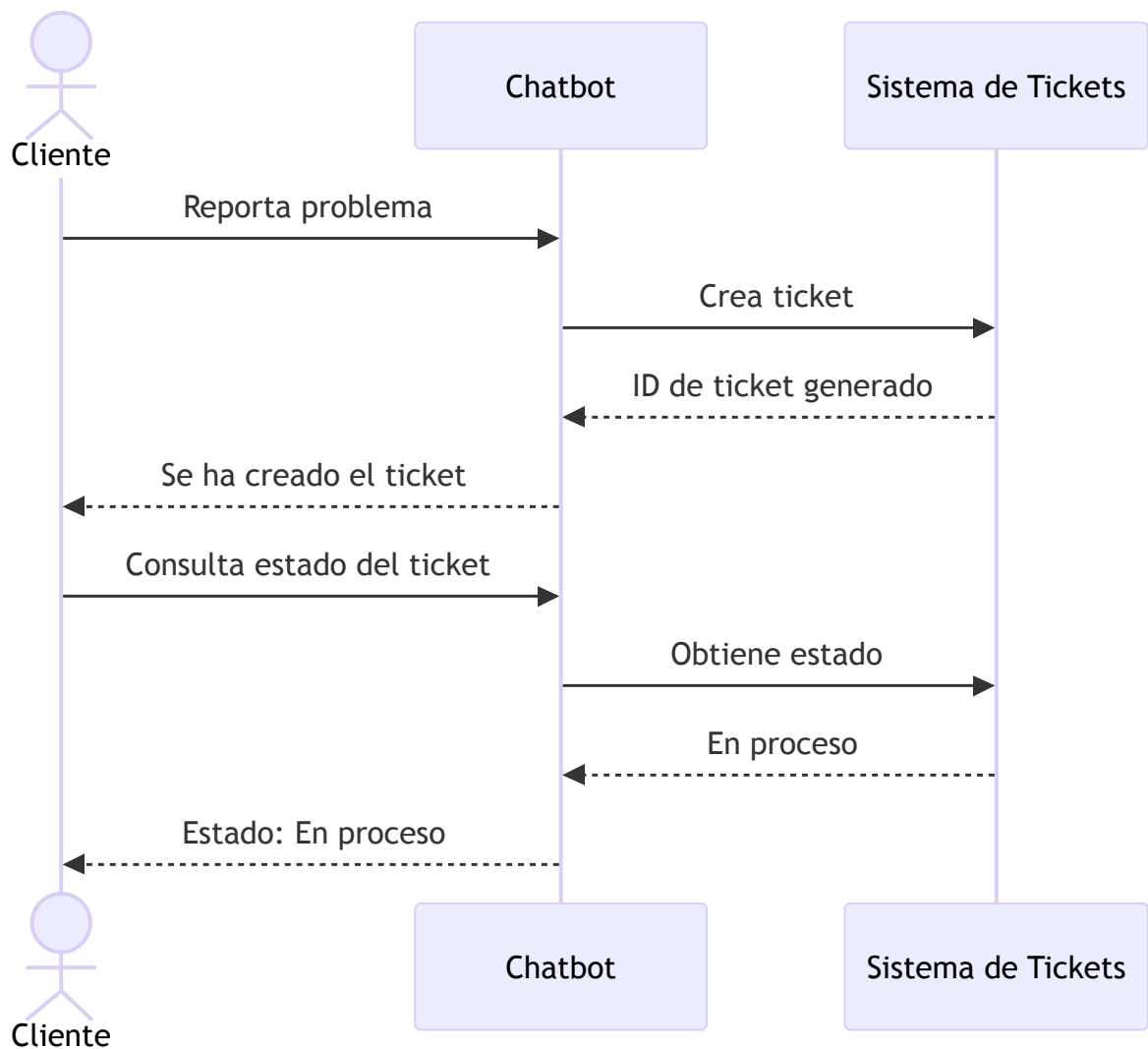




### Retiro de efectivo en un cajero automático



### Soporte técnico automatizado



## Gestión de una máquina de café automática

Una máquina de café automática gestiona la preparación y entrega de bebidas calientes. En su funcionamiento, la máquina puede encontrarse en varios estados. Al inicio, la máquina está a la espera de que un usuario realice una selección. Cuando el usuario elige una bebida (como café, té o chocolate caliente), la máquina pasa a un proceso de preparación específico según la bebida seleccionada. Si el proceso de preparación concluye sin problemas, la máquina entrega el producto al usuario y regresa al estado inicial para una nueva solicitud.

Sin embargo, si durante la preparación ocurre un error, como falta de ingredientes o una avería técnica, la máquina debe entrar en un estado de error que bloquea nuevas solicitudes hasta que un técnico de mantenimiento intervenga. El técnico podrá reiniciar el sistema y devolver la máquina a su estado inicial. Además, el proceso de **preparación** debe incluir detalles específicos dependiendo del tipo de bebida elegida: preparar café, preparar té o preparar chocolate caliente.

# Prompts de IA para practicar

## Prompts de la IA para aprender un lenguaje de programación o de modelado como Plantuml

La Inteligencia Artificial es una herramienta muy potente para aprender cualquier lenguaje de programación a nivel básico o básico-medio. Sin embargo, tiene una tendencia muy fuerte a darte soluciones sin permitirte discurrir. Para poder aprender bien un lenguaje de programación o marcado, como plantuml, debemos darle instrucciones precisas para que:

1. Vaya paso a paso
2. Ponga ejercicios después de cada paso
3. No te dé la solución de los ejercicios hasta que no le hayas presentado tú una propuesta.

Sabiendo esto, puedes convertir a la IA en una herramienta fundamental para aprender cualquier lenguaje, por ejemplo plantuml, para lo que podríamos escribir un prompt como el que sigue:

Necesito aprender a hacer diagramas en plantuml, hazme un tutorial para poder aprender a usarlos para hacer diagramas de actividad. Ponme ejercicios intercalados para que vaya practicando. Ve poco a poco. No me des la solución de los ejercicios hasta que no los haya presentado yo.

Entonces, la inteligencia artificial nos presentaría un tutorial de inicio de plantuml enfocado a resolver

## Prompts de la IA para practicar preguntas de tipo test

A la hora de practicar preguntas tipo test, es importante decirle a la inteligencia artificial cómo va a ser el test (número de respuestas posibles, cuántas pueden ser verdaderas) y darle un material o una temática para que elabore las preguntas. Le podemos adjuntar un archivo para que extraiga contenido de ahí, o hablarle de un tema del que queramos aprender.

También es importante darle instrucciones para que no nos diga la respuesta correcta hasta que no hayamos hecho nuestro intento. A la hora de comenzar a practicar, es recomendable hacer las preguntas de una en una, para profundizar en cada una, y poco a poco hacer más preguntas de golpe, pero siempre sin que nos diga la solución hasta que lo hayamos intentado.

Para hacer una prueba, primero añade el documento sobre el que quieras practicar y ponle el siguiente *prompt*.

Dado este documento, ¿me puedes crear preguntas tipo test para practicar? En estas preguntas, tiene que haber cuatro alternativas (a, b, c, d) de las cuales solo una puede ser correcta. Me haces primero la pregunta, yo te respondo con la letra que creo que es correcta y me respondes si he acertado o no.

Usando, por ejemplo, el archivo de transición de estados, podría contestar algo como esto:

Perfecto. Vamos a comenzar con una pregunta tipo test basada en el contenido del documento sobre **diagramas de transición de estados**.

### Pregunta 1:

¿Cuál de las siguientes afirmaciones describe **correctamente** un diagrama de estados?

- a) Representa únicamente los datos estáticos de un sistema orientado a objetos.
- b) Es una representación gráfica de los algoritmos empleados en la programación funcional.
- c) Representa visualmente los estados y transiciones que experimenta un objeto ante eventos.
- d) Se utiliza exclusivamente para modelar estructuras de base de datos relacionales.

¿Cuál crees que es la respuesta correcta? (a, b, c, d)

La pregunta correcta es C, a lo que contestaría:

 ¡Correcto!

La respuesta **c) Representa visualmente los estados y transiciones que experimenta un objeto ante eventos** es la acertada.

Los diagramas de transición de estados son útiles para modelar el comportamiento dinámico de un sistema, mostrando cómo un objeto cambia de estado en respuesta a distintos eventos.

¿Listo para la siguiente pregunta?

Le decimos que sí.

Genial, vamos con la siguiente:

## Pregunta 2:

En un diagrama de estados, ¿qué representa el **círculo negro relleno**?

- a) Un estado final
- b) Una transición obligatoria
- c) El estado inicial del sistema
- d) Una pausa en la ejecución del proceso

¿Cuál es tu respuesta? (a, b, c, d)

En este caso, fallamos y escribimos la letra b, para lo que nos responde:

**✗** Incorrecto.

La respuesta correcta es: **c) El estado inicial del sistema.**

En los diagramas de estados, el **círculo negro relleno** indica el **estado inicial**, es decir, el punto desde el cual comienza la ejecución del sistema o el ciclo de vida del objeto.

¿Quieres otra pregunta?

## Prompts para ejercicios

En este sentido, las inteligencias artificiales son menos eficientes a la hora de plantear ejercicios buenos, ya que tienen tendencia a decirte exactamente qué pasos tienes que hacer. Esto limita el aprendizaje y conviene afinar mucho para que no lo haga. Algo así:

Dado el documento adjunto, hazme un enunciado para un ejercicio de desarrollo en el que tenga que poner en práctica los conocimientos del documento. Puedes basarte en alguno de los ejercicios que hay desarrollados en él. No me des la solución, la quiero hacer yo y después pasártela.

Sobre el ejercicio, no me des lo que tengo que hacer paso a paso, sino que quiero extraer la información de un enunciado. Haz que sea desafiante y un poco abierto.

A lo cual, responderá:

Perfecto, aquí tienes un **enunciado de ejercicio de desarrollo** que pone a prueba tu comprensión de los diagramas de transición de estados. Es abierto, algo desafiante y te permite aplicar varios conceptos del documento:

### **Enunciado del ejercicio: Sistema de gestión de viajes compartidos**

Diseña un diagrama de transición de estados que represente el ciclo completo de un **viaje en una aplicación de transporte tipo Uber o Cabify**, desde el momento en que un usuario abre la app hasta que el viaje se completa o se cancela por algún motivo.

Tu diagrama debe incluir al menos:

- Estados relacionados con la solicitud del viaje, la asignación del conductor, la llegada del conductor, el desarrollo del viaje y su finalización.
- Posibles eventos como cancelaciones por parte del conductor o del usuario, fallos en la asignación, esperas prolongadas, etc.
- Considera la posibilidad de que el viaje no se complete y vuelva a un estado anterior, o que haya un estado de error.
- Puedes utilizar **subestados** si consideras que hay procesos internos más detallados (por ejemplo, dentro del estado "Viaje en curso").

**No se espera un diagrama perfecto, sino que elabores un modelo coherente que demuestre tu comprensión del concepto de estados, transiciones y eventos.**

Cuando tengas tu solución, compártela aquí (si usas PlantUML, perfecto). Estoy listo para revisarla y darte feedback.

# Ampliación: Diagramas Globales de Interacciones

- [Ampliación: Diagramas Globales de Interacciones](#)
  - [Diagramas Globales de Interacciones](#)
  - [Estructura de un Diagrama Global de Interacciones](#)
  - [Entrega de la actividad de ampliación](#)

## Diagramas Globales de Interacciones

Los diagramas globales de interacciones son una herramienta de modelado poderosa que te permite obtener una vista de alto nivel del flujo de control dentro de un sistema, mientras que, al mismo tiempo, te permite profundizar en los detalles de cómo los objetos interactúan en cada paso. Imagina un mapa de carreteras que, además de mostrarte las rutas principales, te permite hacer zoom en cada cruce para ver cómo se mueven los coches en detalle. Eso es un diagrama global de interacciones.

Estos diagramas **combinan la lógica de flujo de los diagramas de actividad con los detalles de interacción de los diagramas de secuencia o comunicación**. Esto los convierte en una opción ideal para describir escenarios complejos donde hay múltiples interacciones que ocurren en una secuencia lógica.

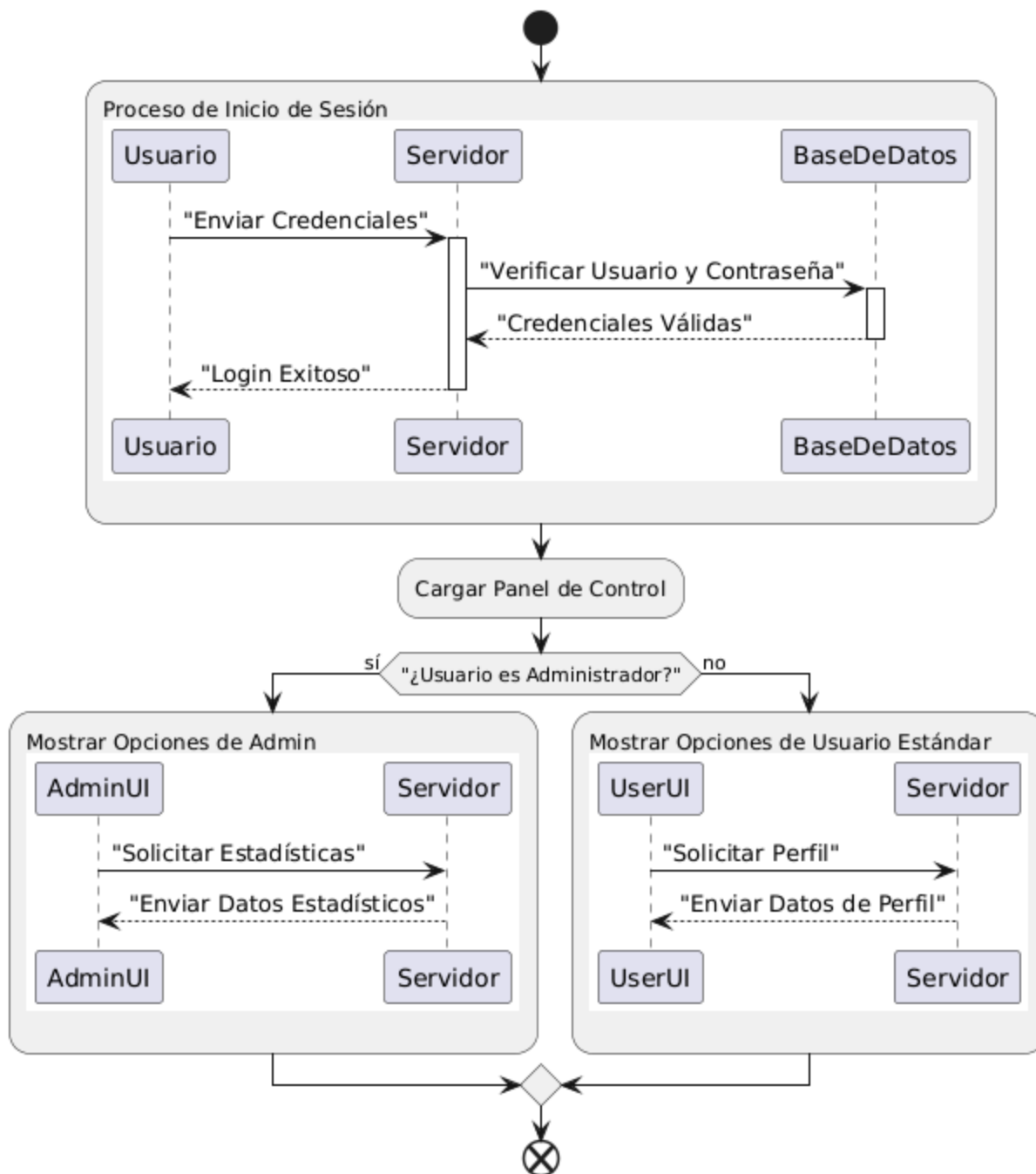
Los diagramas globales de interacción poseen las siguientes características:

- **Visión General y Detalle:** Permiten representar una secuencia de eventos o actividades de alto nivel, pero cada "actividad" puede ser, en realidad, una interacción compleja (como una secuencia de mensajes entre varios componentes).
- **Claridad en Sistemas Complejos:** Son excelentes para descomponer un proceso grande en partes manejables, donde cada parte es una interacción específica entre objetos.
- **Reuso:** Si tienes interacciones de secuencia o comunicación ya definidas, puedes reutilizarlas dentro de un flujo de actividades más amplio.
- **Flujo de Control:** Muestran decisiones, bifurcaciones y uniones de flujos, lo que es característico de los diagramas de actividad.

# Estructura de un Diagrama Global de Interacciones

Fundamentalmente, un diagrama global de interacciones es un **diagrama de actividad** que, en lugar de contener acciones o actividades simples, incluye **fragmentos de interacción**. Estos fragmentos pueden ser:

- **Referencias a Diagramas de Secuencia o Comunicación:** Un cuadro que representa una interacción completa definida en otro lugar.
- **Fragmentos de Interacción en Línea:** Directamente dentro del flujo de actividades, se incrusta un pequeño diagrama de secuencia o comunicación.





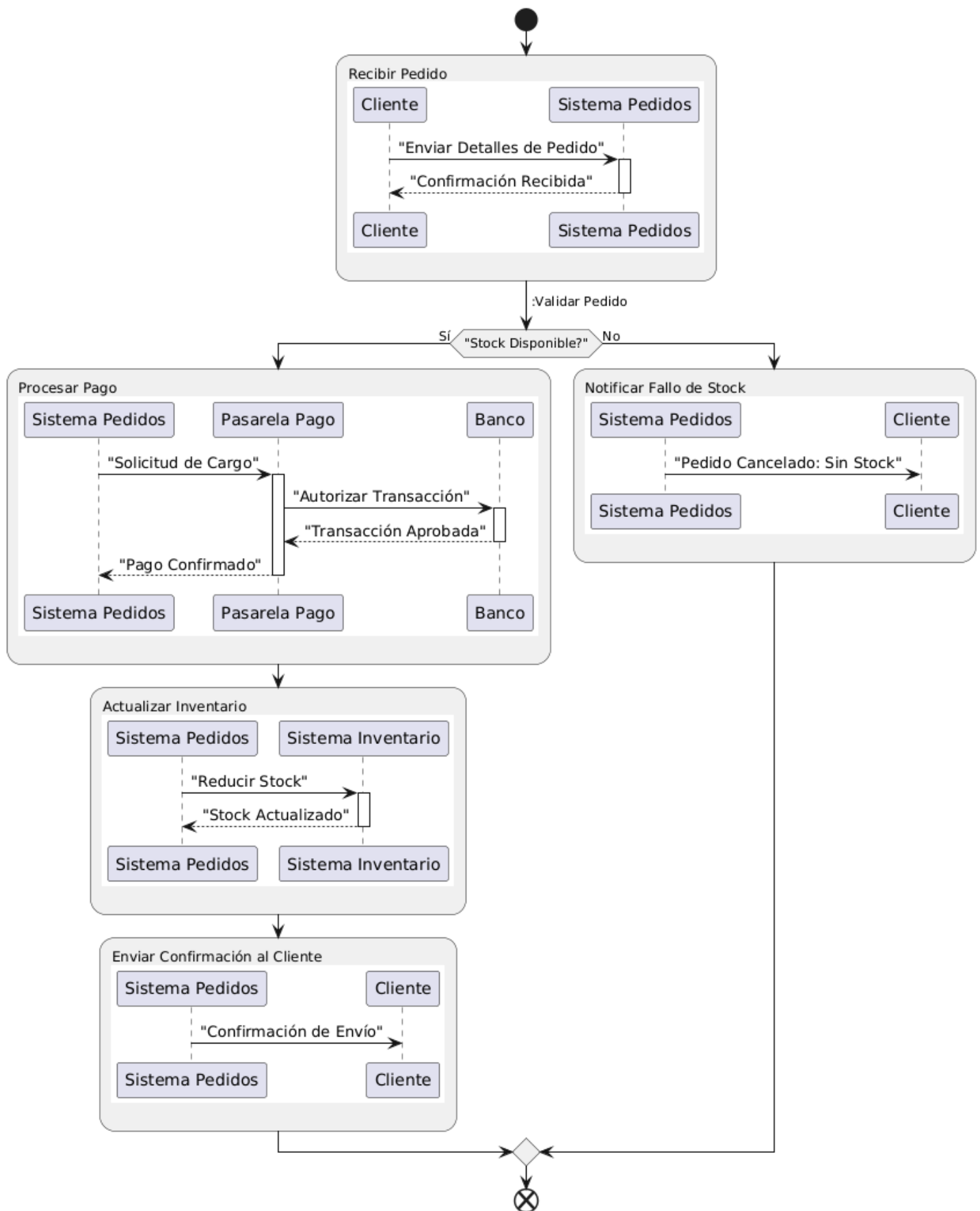
## ► Haz click aquí para ver el código plantuml

Observa el diagrama de arriba, que representa un flujo de inicio de sesión con diferentes acciones basadas en el rol del usuario.

1. **Inicio y Fin:** El diagrama comienza con un nodo de **inicio** ( Start ) y termina con un nodo de **fin** ( End ), igual que un diagrama de actividad.
2. **Actividades (acciones):** Las rectángulos con texto como :Proceso de Inicio de Sesión; o :Cargar Panel de Control; representan pasos o acciones generales en el flujo.
3. **Fragmentos de Interacción (los cuadros dentro de las actividades):**
  - Fíjate en el gran recuadro que aparece debajo de :Proceso de Inicio de Sesión; . Este es un **fragmento de interacción** incrustado.
  - Dentro de este fragmento, puedes ver **participantes** (Usuario, Servidor, BaseDeDatos) y **mensajes** que fluyen entre ellos (por ejemplo, "Usuario -> Servidor : "Enviar Credenciales"). Esto muestra en detalle cómo ocurre el "Proceso de Inicio de Sesión": el Usuario envía credenciales al Servidor, el Servidor verifica con la Base de Datos, y luego el Servidor responde al Usuario. Las barras de activación ( activate / deactivate ) indican cuándo un participante está activo en el proceso.
  - De manera similar, los fragmentos debajo de "Mostrar Opciones de Admin" y "Mostrar Opciones de Usuario Estándar" son otras interacciones que detallan la comunicación entre la interfaz de usuario y el servidor para cada tipo de rol.
4. **Flujo de Control (Flechas):** Las flechas conectan las actividades y los fragmentos de interacción, mostrando la secuencia en la que se ejecutan.
5. **Puntos de Decisión ( If . . . then . . . else . . . Endif ):** El rombo con la pregunta "Usuario es Administrador?" es un punto de decisión. El flujo se bifurca en dos caminos ( [Si] o [No] ) dependiendo de la condición. Cada camino lleva a una interacción diferente y, finalmente, ambos caminos convergen para terminar el flujo.

### Actividad

Interpreta el siguiente diagrama de interacción global:



► **Haz click aquí para ver el código plantuml**

Usa un diagrama global de interacciones cuando:

- Necesitas mostrar el **orden de ejecución** de varias interacciones.
- El sistema tiene **puntos de decisión** o **bucles** que afectan qué interacciones se llevan a cabo.
- Quieres proporcionar una **vista de alto nivel** de un proceso, pero con la capacidad de "hacer zoom" en cada paso para ver las interacciones detalladas entre objetos.
- Estás documentando un **caso de uso complejo** que involucra múltiples pasos y subsistemas.

Este tipo de diagrama es una herramienta de modelado muy versátil que cierra la brecha entre la descripción del flujo de alto nivel y el detalle de la interacción entre objetos. Te permite comunicar la arquitectura y el comportamiento de sistemas de manera mucho más completa.

### Actividad:

Crea un diagrama de tiempos a partir de la siguiente definición:

Una máquina de café automática gestiona la preparación y entrega de bebidas calientes. En su funcionamiento, la máquina puede encontrarse en varios estados. Al inicio, la máquina está a la espera de que un usuario realice una selección. Cuando el usuario elige una bebida (como café, té o chocolate caliente), la máquina pasa a un proceso de preparación específico según la bebida seleccionada. Si el proceso de preparación concluye sin problemas, la máquina entrega el producto al usuario y regresa al estado inicial para una nueva solicitud.

Sin embargo, si durante la preparación ocurre un error, como falta de ingredientes o una avería técnica, la máquina debe entrar en un estado de error que bloquea nuevas solicitudes hasta que un técnico de mantenimiento intervenga. El técnico podrá reiniciar el sistema y devolver la máquina a su estado inicial. Además, el proceso de **preparación** debe incluir detalles específicos dependiendo del tipo de bebida elegida: preparar café, preparar té o preparar chocolate caliente.

## Entrega de la actividad de ampliación

Entrega una memoria en la que se incluyan los ejercicios de este apartado y **un diagrama de tiempos correspondiente al reto individual**.

# Ampliación: Diagramas de tiempos

- [Ampliación: Diagramas de tiempos](#)
  - [Definición y uso de los Diagramas de Tiempos](#)
  - [Elementos Clave de un Diagrama de Tiempos](#)
  - [Entrega de la actividad de ampliación](#)

## Definición y uso de los Diagramas de Tiempos

Los diagramas de tiempos son una herramienta de modelado visual que te permite representar cómo el estado de uno o varios elementos (como componentes de software, hardware o procesos) cambia a lo largo del tiempo, y cómo interactúan entre sí en una secuencia temporal. Son excepcionalmente útiles para entender el comportamiento de sistemas en tiempo real, protocolos de comunicación o cualquier escenario donde la cronología precisa de los eventos es crucial.

Imagina que necesitas ver la secuencia de acciones y cambios de estado de un procesador y la memoria en un periodo de tiempo. Un diagrama de tiempos te ofrece una vista detallada de esto.

Usa un diagrama de tiempos cuando:

- Necesitas mostrar los **cambios de estado** de un elemento a medida que el tiempo avanza.
- Quieres visualizar **interacciones temporales** y la sincronización entre varios elementos.
- Estás diseñando sistemas donde la **secuencia precisa y la duración** de los eventos son fundamentales, como en sistemas embebidos, redes o hardware.
- Buscas identificar **cuellos de botella** o problemas de latencia en un proceso.

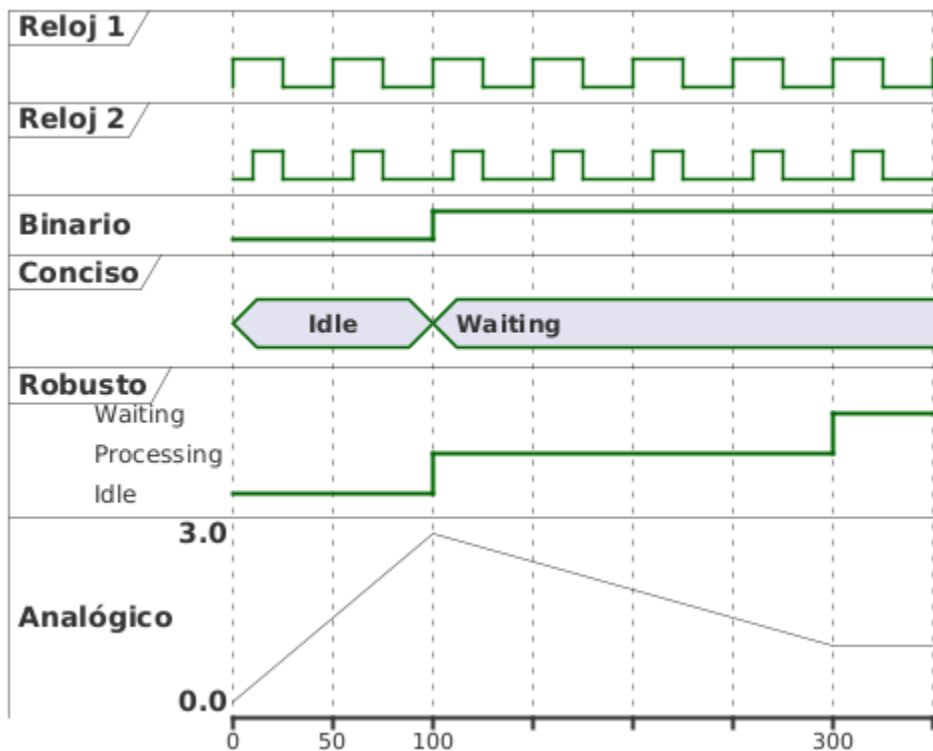
## Elementos Clave de un Diagrama de Tiempos

Un diagrama de tiempos se compone de los siguientes elementos principales:

1. **Líneas de Vida:** Representan los elementos o componentes cuyo comportamiento estamos modelando. Cada línea de vida tiene su propia línea horizontal donde se muestran sus estados a lo largo del tiempo. La línea del tiempo se puede representar de varias maneras: analógica, binaria, reloj, concisa o robusta.
  - a. La representación de reloj sirve para indicar pulsos.
  - b. La representación binaria sirve para representar objetos que cambian entre dos estados

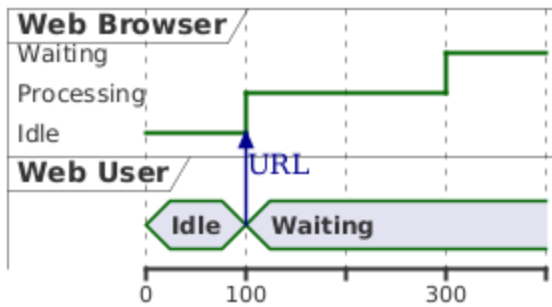
- c. Las representaciones concisa y robusta sirven para representar objetos que cambian de estado de forma discreta (tienen un número finito de estados).
  - d. La representación analógica sirve para representar valores infinitos entre dos puntos (estados).
2. **Eje del Tiempo:** Es el eje horizontal, que avanza de izquierda a derecha y representa progreso del tiempo. Puedes marcar puntos específicos en el tiempo o definir escalas.
  3. **Estados:** Indican el estado en el que se encuentra un elemento en un momento dado. Un elemento puede pasar de un estado a otro en distintos puntos del tiempo. En el modo robusto indicamos los estados en el eje vertical, mientras que en el modo conciso los representamos en un hexágono.

Observa el siguiente ejemplo:



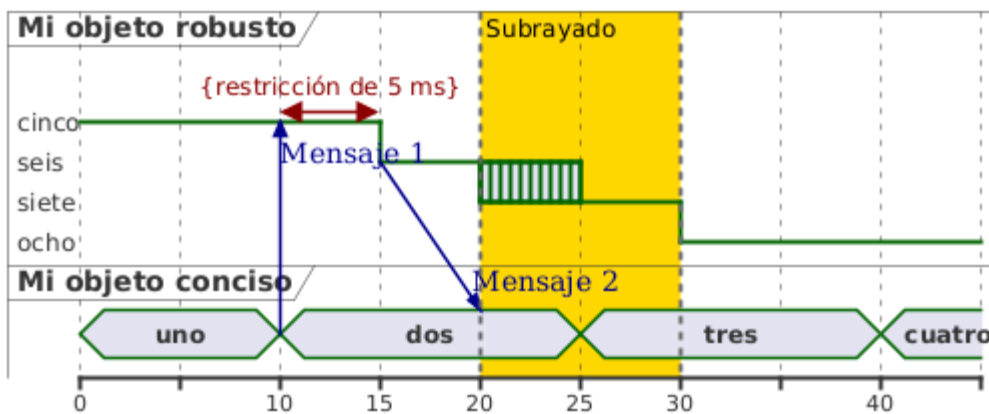
► **Haz click aquí para ver el código plantuml**

4. **Mensajes/Eventos:** Muestran interacciones o el envío de información entre elementos en un instante particular.



► [Haz click aquí para ver el código plantuml](#)

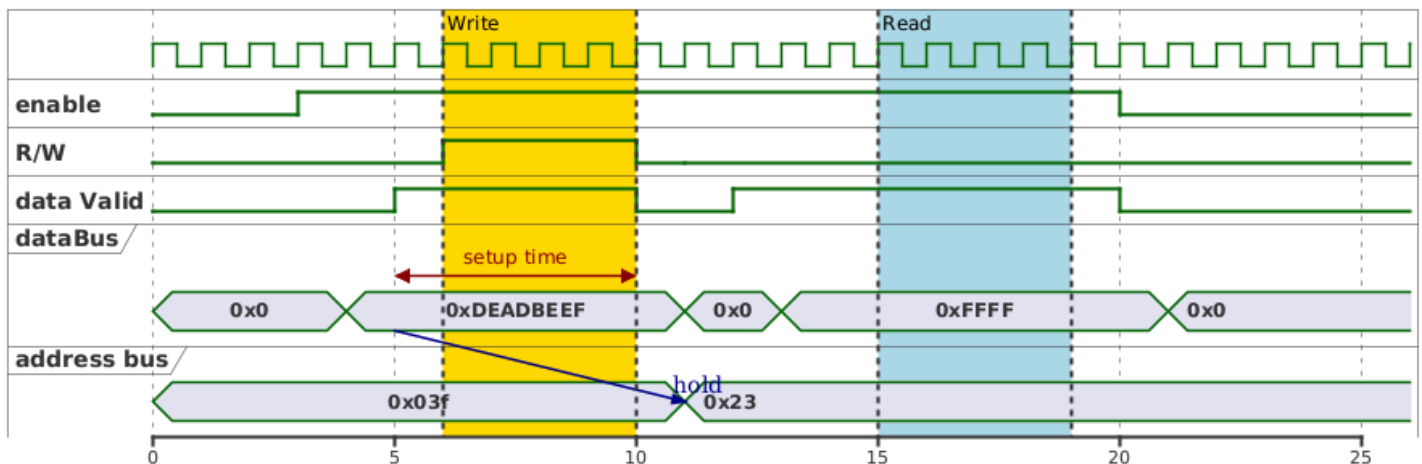
5. **Restricciones, indefiniciones, anotaciones y subrayados:** Además, se pueden añadir restricciones en forma de flechas horizontales, estados indefinidos (en los objetos robustos), anotaciones y subrayados a algunas secciones de tiempo, como en el siguiente ejemplo:



► [Haz click aquí para ver el código plantuml](#)

### Actividad

Interpreta el siguiente diagrama de tiempos.



► [Haz click aquí para ver el código plantuml](#)

A la hora de diseñar un Diagrama de Tiempos, debes tener en consideración sobre todo los siguientes puntos:

- **Claridad ante todo:** Usa nombres claros y concisos para tus elementos y sus estados. Si un estado tiene varias palabras, como "Processing Command", el diagrama lo mostrará correctamente.
- **Lee el Tiempo de Izquierda a Derecha:** Siempre sigue el diagrama de izquierda a derecha para entender la secuencia cronológica de eventos.
- **Enfócate en lo Importante:** No intentes modelar cada detalle minucioso. Resalta los cambios de estado y las interacciones clave que quieres comunicar para no sobrecargar el diagrama.
- **Etiquetas Descriptivas:** Las etiquetas de los mensajes (como "Sensor Ready") deben ser lo suficientemente descriptivas como para entender el propósito de la interacción.

Puedes profundizar más en la página de [Plantuml dedicada a los diagramas de tiempos](#).

### Actividad:

Crea un diagrama de tiempos a partir de la siguiente definición:

Una máquina de café automática gestiona la preparación y entrega de bebidas calientes. En su funcionamiento, la máquina puede encontrarse en varios estados. Al inicio, la máquina está a la espera de que un usuario realice una selección. Cuando el usuario elige una bebida (como café, té o chocolate caliente), la máquina pasa a un proceso de preparación específico según la bebida seleccionada. Si el proceso de preparación concluye sin problemas, la máquina entrega el producto al usuario y regresa al estado inicial para una nueva solicitud.

Sin embargo, si durante la preparación ocurre un error, como falta de ingredientes o una avería técnica, la máquina debe entrar en un estado de error que bloquea nuevas solicitudes hasta que un técnico de mantenimiento intervenga. El técnico podrá reiniciar el sistema y devolver la máquina a su estado inicial. Además, el proceso de **preparación** debe incluir detalles específicos dependiendo del tipo de bebida elegida: preparar café, preparar té o preparar chocolate caliente.

## Entrega de la actividad de ampliación

Entrega una memoria en la que se incluyan los ejercicios de este apartado y **un diagrama de tiempos correspondiente al reto individual**.

# Convertir VSCode en un editor de Markdown con el poder de las extensiones.

- [Convertir VSCode en un editor de Markdown con el poder de las extensiones.](#)
  - [1. ¿Qué es Markdown y cómo se utiliza?](#)
    - [Sintaxis básica de Markdown](#)
  - [2. Utilizar Markdown en Visual Studio Code](#)
    - [Instalación de Visual Studio Code](#)
      - [Extensiones importantes](#)
      - [Comandos útiles dentro de VSCode usando Markdown All In One](#)
  - [Memoria de entrega](#)

## 1. ¿Qué es Markdown y cómo se utiliza?

**Markdown** es un lenguaje de marcado ligero que permite formatear texto de manera sencilla e intuitiva. Es muy utilizado para escribir documentación, blogs o cualquier otro tipo de texto que luego se convierte a HTML, ya que facilita mucho su legibilidad. Un documento Markdown es esencialmente texto plano que se puede leer fácilmente y que luego se puede exportar a otros formatos como HTML o PDF.

Estos apuntes están escritos combinando Markdown con plantuml y mermaidjs. Además, Markdown es el lenguaje de marcado más empleado a la hora de elaborar **documentación técnica**, y también es el lenguaje de marcado que emplean los chatbots como ChatGPT o Gemini para imprimir sus resultados.

Además, la sencillez de Markdown se puede combinar con el poder de HTML5, permitiendo intercalar etiquetas HTML en documentos Markdown para así conseguir cosas complejas, como aplicar un estilo CSS o usar `<details>` para ocultar elementos (como se hace en estos apuntes con los códigos en plantuml).

## Sintaxis básica de Markdown

### 1. Títulos y subtítulos

Los títulos en Markdown se crean utilizando el símbolo de almohadilla `#`. Cuantas más



almohadillas se colocan, menor es el nivel del título:

- # Título 1
- ## Título 2
- ### Título 3

## 2. Texto en negrita

El texto se puede poner en negrita rodeándolo con dos asteriscos **\*\*** o dos guiones bajos **\_\_** :

- **\*\*Este texto está en negrita\*\***
- **\_\_Este texto también está en negrita\_\_**

## 3. Texto en cursiva

Para poner el texto en cursiva, se utiliza un solo asterisco *\** o un solo guion bajo *\_* :

- *\*Texto en cursiva\**
- *\_Texto en cursiva\_*

## 4. Listas

Para crear una **lista no ordenada** (puntos), se utiliza un asterisco *\** , un signo más *+* , o un guion *-* :

- *\* Elemento 1*
- *- Elemento 2*

Para **listas numeradas**:

- 1. Primer elemento
- 2. Segundo elemento

## 5. Enlaces

Los enlaces se crean utilizando la sintaxis `[texto](url)` :

- `[Google](https://www.google.com)`

## 6. Imágenes

Las imágenes se pueden insertar de manera similar a los enlaces, añadiendo un signo de exclamación al principio `![texto alternativo](url)` :

- `![Logo](https://url-de-la-imagen.com/logo.png)`

## 7. Código

Si quieres mostrar un fragmento de código dentro de una línea de texto, puedes utilizar las comillas invertidas ``` . Para **bloques de código**, se pueden usar tres comillas invertidas `````:

- inline code: `\ echo "Hola"`
- Bloques de código:
- Puedes especificar incluso el lenguaje de programación para que resalte la sintaxis si quieres (no funcionará en todos los entornos).

```
echo "Esto es un bloque de código"
```

## 8. Citas

Se pueden añadir citas utilizando `>` al principio de la línea:

- `>` Esto es una cita

# 2. Utilizar Markdown en Visual Studio Code

## Instalación de Visual Studio Code

Primero que todo y si no lo has hecho aún, es necesario instalar **VSCode**, que se puede descargar desde su [página oficial](#). Una vez instalado, ya estaremos listos para personalizarlo y utilizarlo para editar Markdown.

## Extensiones importantes

Para mejorar la experiencia trabajando con Markdown en VSCode, es necesario instalar algunas **extensiones muy útiles**:

### 1. Markdown All in One

Esta extensión proporciona herramientas útiles para editar Markdown, como vista previa, cierre automático de listas y otros atajos de teclado.

- Para instalarla, ve a la sección de extensiones dentro de VSCode y busca "Markdown All in One".

### 2. Markdown PDF

Esta extensión permite convertir ficheros `.md` a PDF, HTML u otros formatos.

- Busca "Markdown PDF" e instálala para poder exportar tus documentos.

### 3. Markdown Enhanced Preview (RECOMENDADA)

Esta extensión es ideal para trabajar con Markdown, Mermaid y Plantuml. Incorpora funciones de todo lo anterior, por lo que vas a poder exportar el resultado a pdf o incluso HTML. Puedes instalar solamente esta extensión y funcionará, es una especie de sustituta de las otras dos.

### 4. Markdownlint

Añade un **linter** que verifica si tu Markdown sigue buenas prácticas, ayudándote a corregir errores comunes y a seguir una sintaxis pura de Markdown (cosa que no siempre te será útil)

- Busca "Markdownlint" en la sección de extensiones.

## Comandos útiles dentro de VSCode usando Markdown All In One

### 1. Vista previa de Markdown

Para ver una vista previa del fichero Markdown que estás editando, puedes abrir el fichero `.md`

y hacer clic en `Ctrl+Shift+V` (Windows/Linux) o `Cmd+Shift+V` (Mac). También puedes hacer clic con el botón derecho sobre el fichero y seleccionar "Open Preview".

## 2. Exportar a HTML o PDF

Una vez tienes el documento editado, puedes exportarlo a HTML o PDF utilizando la extensión Markdown PDF:

- Para exportarlo, abre la paleta de comandos ( `Ctrl+Shift+P` o `Cmd+Shift+P` ) y escribe "Markdown PDF: Export". Selecciona el formato deseado.
- También puedes escribir `>` en la barra de búsqueda. Tiene el mismo efecto y puede ser más fácil de recordar.

Otra herramienta de diagramas que está ganando popularidad recientemente, en parte gracias a su compatibilidad con diferentes frameworks de aplicaciones web como `astro.build`, es `d2`. Puedes encontrar más información [aquí](#).

# Memoria de entrega

Para entregar la actividad, crea una memoria en la que repases paso a paso el proceso de instalación y preparación de las extensiones necesarias para poder editar `plantuml` y `mermaid.js` en VS Code. Crea un documento con capturas de pantalla de todo el proceso.