

# Projecte de Programació

## Curs 2024/25 – Q2

### **Introducción**

- Asignatura de proyecto
- Introducció pràctica a la OOP
- Fomenta el treball en equip
- Fomenta la iniciativa pròpia

# Competencias transversales:

- Trabajo en equipo

Se evalúa usando los criterios que encontraréis en la Rúbrica

- Emprendimiento e innovación

Se evalúa en función de las funcionalidades opcionales propuestas vs las funcionalidades opcionales implementadas

Se evalúa en función de la iniciativa del grupo al escoger algoritmos y EDs



# Rúbrica de evaluación Competencia Trabajo en Equipo

	A	B	C	D
Asistencia	Asistió al 100% de las clases de laboratorio y demás reuniones del equipo.	Asistió a entre un 75% y un 99% de las clases de laboratorio y demás reuniones del equipo.	Asistió a entre un 50% y un 74% de las clases de laboratorio y demás reuniones del equipo.	Asistió al 50% o menos de las clases de laboratorio y demás reuniones del equipo.
Trabajo asignado	Siempre entregó el trabajo asignado dentro del plazo, y sin necesidad de seguimiento para ello.	Entregó el trabajo, aunque algunos tarde, y en ocasiones requirió seguimiento.	Dejó de entregar algún trabajo y requirió seguimiento.	Entregó sólo algunos trabajos y requirió mucho seguimiento.
Calidad del trabajo	En particular en el caso del código entregado y sus juegos de prueba asociados, la calidad es alta.	En particular en el caso del código entregado y sus juegos de prueba asociados, la calidad es media.	En particular en el caso del código entregado y sus juegos de prueba asociados, la calidad es justa.	En particular en el caso del código entregado y sus juegos de prueba asociados, la calidad es baja.
Contribución	Siempre aportó al logro de los objetivos. Buscó y sugirió soluciones a los problemas.	Casi siempre aportó al logro de los objetivos. Casi siempre buscó y sugirió soluciones a los problemas.	Pocas veces aportó al logro de los objetivos. Pocas veces buscó y sugirió soluciones a los problemas.	No aportó al logro de los objetivos. No buscó y sugirió soluciones a los problemas.
Integración al grupo	Siempre cumplió las normas y se adaptó a los cambios del equipo. Trató con respeto a sus compañeros y trabajó motivado, promoviendo la participación y la cooperación entre los miembros del equipo.	Casi siempre cumplió las normas y se adaptó a los cambios del equipo. Casi siempre trató con respeto a sus compañeros y trabajó motivado, promoviendo la participación y la cooperación entre los miembros del equipo.	Pocas veces cumplió las normas y se adaptó a los cambios del equipo. Pocas veces trató con respeto a sus compañeros y trabajó motivado, promoviendo la participación y la cooperación entre los miembros del equipo.	Nunca cumplió las normas y se adaptó a los cambios del equipo. No trató con respeto a sus compañeros ni trabajó motivado, promoviendo la participación y la cooperación entre los miembros del equipo.
Actitud ante la crítica	Siempre estuvo receptivo a aceptar críticas y sugerencias del profesor y de los demás compañeros del grupo.	Casi siempre estuvo receptivo a aceptar críticas y sugerencias del profesor y de los demás compañeros del grupo.	Pocas veces estuvo receptivo a aceptar críticas y sugerencias del profesor y de los demás compañeros del grupo.	Muy pocas veces o nunca estuvo receptivo a aceptar críticas y sugerencias del profesor y de los demás compañeros del grupo.
Actitud al comunicar	Siempre estuvo atento a las opiniones del profesor y de sus compañeros. Habló y escuchó equitativamente.	En la mayoría de las ocasiones escuchó y en pocas ocasiones habló.	En la mayoría de las ocasiones habló y en pocas ocasiones escuchó.	Siempre habló y muy pocas veces o nunca escuchó al profesor o a otros miembros del equipo.

- Clases de teoría:

Explican lo necesario para desarrollar el Proyecto:

Conceptos OO, Java, Arquitectura 3 capas, Testing, etc.

- Utilizaremos parte de lo que ya sabéis de IES pero nos interesará sólo aquello necesario para el proyecto

- Clases de laboratorio:

Esencialmente las usaréis para desarrollar el proyecto y resolver dudas con vuestro tutor. También el tutor las puede usar para aclaraciones relacionadas con el proyecto



- Decisiones predeterminadas en PROP:

Enfoque del proyecto: Orientación a Objetos

Notación diseño: UML

Metodología de diseño: Arquitectura en 3 capas

Lenguaje de programación: Java

NO se permite usar BDs  
(potenciar uso estructuras de datos)

## Decisiones predeterminadas en PROP:

- Lenguaje de programación: Java -> JDK22.\*


A menos que se explicita lo contrario, se puede usar por defecto **cualquier** librería incluida dentro del entorno JDK de Oracle

Se ha de pedir permiso expreso al tutor para usar **cualquier** librería no incluida en el entorno anterior

- Librería para testing: JUnit (versión = 4.\*)

- Nota Final PROP:

# 100% Nota proyecto

IMPORTANT: el Proyecto se hace en equipo, pero la nota del proyecto es individual  codificación individual de las clases

Se asume que la documentación que se pide en cada entrega se elabora de forma conjunta, y su elaboración no exime de programar



## - Proyecto: TRES entregas

VÍA RACÓ

- Entrega 1 (**22/4 23:59h**):

Casos de uso, diseño e implementación del modelo de datos y las funcionalidades principales. Algunos tutores pueden pedir exposiciones presenciales.

- Entrega 2 (**26/5 23:59h**):

Diseño completo.

- Entrega 3 (**2/6 23:59h**):

Proyecto completo. Es obligatoria una demostración interactiva (normalmente posterior) delante del tutor.

## - Nota Proyecto:

Sea:  $N_k$  la nota de la entrega  $k$  ( $k=1,2,3$ ),  
FT el factor de trabajo individual (entre 0 y 1)

La nota individual del proyecto es:

$$FT \times (0.4 \times N_1 + 0.15 \times N_2 + 0.45 \times N_3)$$

La nota dependerá de:

- corrección del contenido
- corrección del formato: atención al documento de "**Normes dels Lliuraments**"

Cada grupo de laboratorio se denomina cluster


Cada *cluster* tiene un tutor y se compone de un cierto número de equipos de 4 personas (o puntualmente 3 si no cuadra la matriculación)

Hay un solo enunciado para todos los equipos y *clusters*. Sale publicado durante la 2ª semana del curso.



Está estrictamente prohibido formar equipos entre personas de grupos de laboratorio diferentes

Cambios de grupo:

1. Intentar pedir cambio oficial a la FIB dentro del plazo
2. Los cambios de grupo no oficiales sólo están permitidos si son intercambios entre grupos. Es decir, si alguien quiere cambiar del grupo A al B es necesario que encuentre a alguien del grupo B que quiera ir al grupo A  FORO Racó

## Herramientas

Excepto Git, no se prefija ninguna herramienta concreta (decisión de cada equipo)

- Opcional: IDE para programación en Java  
(se permite generar automáticamente las clases de interfície a partir de su editor visual)
- Obligatorio: herramienta de edición de diagramas UML  
(hay múltiples opciones en la web de la asignatura)
- Imprescindible: herramienta de control de versiones -> **Git**  
(se crearán cuentas GitLab para cada equipo, y es OBLIGATORIO usarlas como repositorio del código)
- Opcional: herramienta de documentación (Doxygen, Javadoc)  
(objetivo: reusabilidad)
- Opcional: herramienta de planificación (Trello, *issues* de Git)

## Programa clases de teoría:

- Introducción
- Repaso IES: Ciclo de vida del software
- Ejemplo de la primera entrega
- Introducción a Java
- Repaso conceptos OO e implementación en Java
- Arquitectura 3 capas e implementación en Java
- Prueba de programas: Drivers & Mocks/Stubs
- Prueba de programas: Testing con Junit
- Patrones de diseño con Java
- Diseño de interfícies con Java



## - Bibliografia:

An Introduction to Object Oriented Programming

Timothy Budd

Addison-Wesley 2002

The Object Oriented Thought Process (3<sup>rd</sup>. ed)

Matt Weisfeld

Pearson Ed. 2009

JUnit in Action (2nd. ed.)

Tahcheiv, Petar; Leme, Felipe; Massol, Vincent; Gregory, Gary

Manning , 2011

# Ciclo de Vida

El desarrollo fructífero de una aplicación informática requiere un esfuerzo en equipo:

- Clientes: tienen necesidades de software que ha de ser desarrollado -> comunican lo que necesitan, se aseguran de que obtienen lo que necesitan, pueden cambiar de opinión y aun así obtener lo que necesitan
- Programadores: definen la arquitectura, diseñan la aplicación y escriben los tests y el código que la soporta
- Jefes: controlan los recursos del proyecto -> miden el progreso del proyecto y su calidad. Pueden contestar a la pregunta: "cuándo estará hecho?" (usando el rendimiento real de los programadores para predecir la finalización)

# Ciclo de Vida

Fases por las que pasa una aplicación informática desde que se comienza a pensar hasta que deja de ser útil

Guía la metodología de desarrollo que se utiliza. Hay tantas metodologías como ciclos de vida:

*Cowboy coding*

*Waterfall* (cascada)

Espiral

Iterativo

Ágil

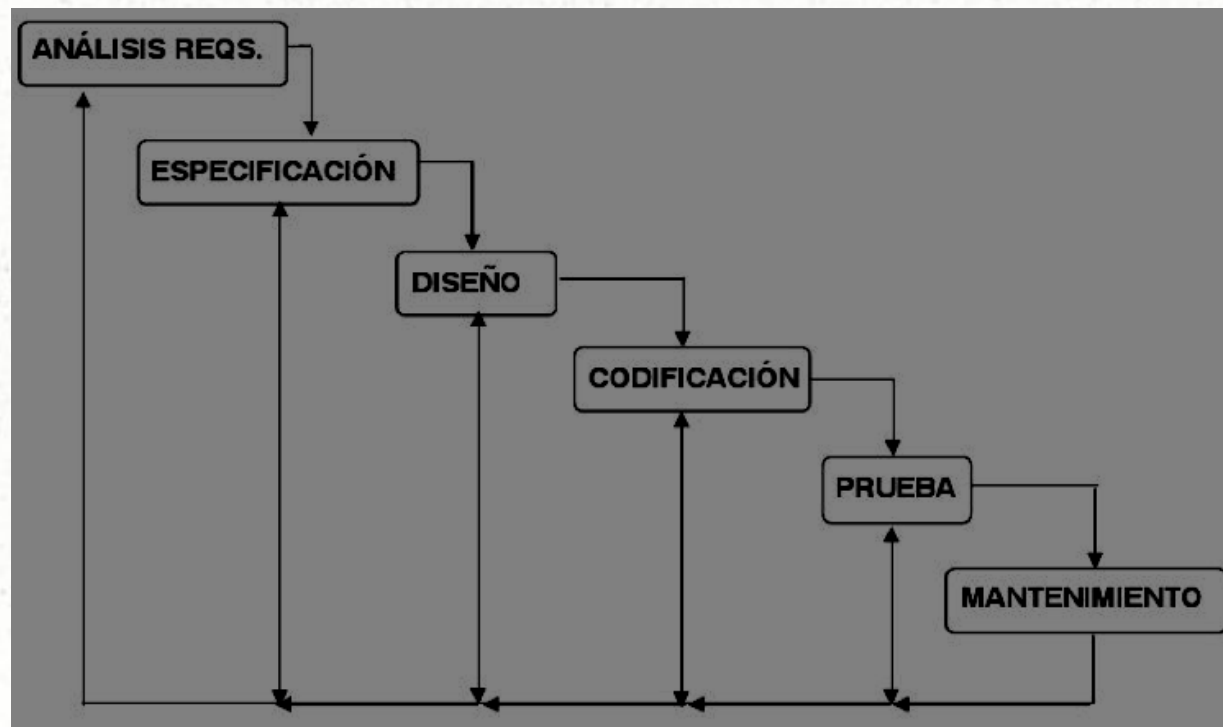
etc...

([http://en.wikipedia.org/wiki/Software\\_development\\_process](http://en.wikipedia.org/wiki/Software_development_process))



# Ciclo de Vida

En PROP usaremos el *modelo clásico aumentado*, basado en el modelo en cascada



En este modelo podemos replantearnos etapas previas del desarrollo

# Ciclo de Vida

## Descripción de las etapas: **Análisis Requerimientos**

En esta etapa hay que saber **QUÉ** quiere el usuario. El usuario es el “protagonista” de esta etapa. A partir de la información que nos proporciona hay que averiguar:

- Identificar el problema
- Qué sistema hay que construir
- Es posible construirlo?
- Lo que pide el usuario es lo que necesita?
- etc...

Esta fase es *independiente* de la tecnología.

# Ciclo de Vida

Descripción de las etapas: **Análisis Requerimientos**

Necesidad de unificar las diferentes visiones del problema:

- Lo que quiere el cliente
- Lo que en realidad necesita
- Lo que interpreta el desarrollador que hay que hacer
- Lo que realmente es el producto final



# Ciclo de Vida

Descripción de las etapas: **Análisis Requerimientos**

- Requerimientos del sistema: qué sistema hay que construir (objetivos y necesidades del usuario)
- Requerimientos del *software*: qué software hay que construir (subconjunto del sistema global)
  - Funcionales
  - No Funcionales
- Documento con la descripción de la aplicación
- Casos de uso: Identificación

# Ciclo de Vida

## Análisis Requerimientos en PROP

- Se parte del enunciado  hay que identificar funcionalidades:

Explícitas

Obligatorias

Implícitas

Opcionales

- Es necesario un estudio previo del problema, que puede incluir análisis de implementaciones ya existentes
- Importante la interacción con el tutor, que actúa a la vez como:  
    Cliente          Jefe

# Ciclo de Vida

## **Análisis Requerimientos en PROP**

Los requerimientos definidos son consensuados entre equipo y tutor

Los requerimientos definidos actúan como contrato que blindo la relación entre el equipo y el tutor



En la tercera entrega se tendrá que justificar funcionalidades propuestas y no implementadas

El tutor no podrá pedir nada que no se hubiera propuesto



# Ciclo de Vida

Descripción de las etapas: **Especificación**

En esta etapa hay que saber **QUÉ** ha de hacer el sistema y describirlo detalladamente:

- Especificación de los *datos*:  
Modelo conceptual de datos
- Especificación de los *procesos*:  
Modelo de comportamiento del sistema\*

Esta fase es *independiente* de la tecnología (aún no hay que decidir el *cómo*).

\*No hay que hacerlo en PROP

# Ciclo de Vida

## Primera entrega proyecto (I):

- **Análisis requerimientos:**
  - Casos de uso (identificación)
- **Especificación:**
  - Modelo conceptual datos (diagrama de clases del modelo, versión especificación)
  - Casos de uso (interacción con el usuario)

# Ciclo de Vida

## Descripción de las etapas: **Diseño**

En esta etapa hay que empezar a pensar **CÓMO** hay que hacer lo que se ha propuesto en las etapas anteriores

Esta fase es *dependiente* de la tecnología:

- Lenguaje de programación (familia y opciones, como por ejemplo si permite herencia múltiple)
- Requisitos no funcionales (responsables de la arquitectura de la aplicación)
- Sistema de Bases de Datos



# Ciclo de Vida

## Descripción de las etapas: **Diseño**

En esta etapa hay que empezar a pensar **CÓMO** hay que hacer lo que se ha propuesto en las etapas anteriores:

- Arquitectura de la aplicación (en PROP no habrá elección: *arquitectura en tres capas*)
- Modelo conceptual de datos (versión diseño)
- Diagramas de secuencia de las operaciones de las clases\*

\*No hay que hacerlo en PROP

# Ciclo de Vida

## Descripción de las etapas: **Diseño**

En esta etapa hay que empezar a pensar **CÓMO** hay que hacer lo que se ha propuesto en las etapas anteriores:

- Contratos de las operaciones de las clases\*
- Lenguaje de programación  
(en PROP será Java)
- Estructuras de datos y algoritmos

\*No hay que hacerlo en PROP



# Ciclo de Vida

Descripción de las etapas: **Codificación y Tests**

En esta etapa hay que implementar todo lo que se ha decidido hasta la fase de diseño



Hay que hacer *tests* sobre TODO lo implementado



# Ciclo de Vida

Descripción de las etapas: **Mantenimiento**

- Corrección de errores
- Ampliaciones o funcionalidades más potentes
- Modificaciones preventivas (efecto 2000) o adaptativas (la ley cambia)

Suele ser la etapa más larga y cara (80% del coste), y no se considera parte del proyecto, sino de explotación del programa

# Ciclo de Vida

## Primera entrega proyecto (II):

### - **Diseño:**

- Arquitectura de la aplicación (3 capas)
- Diagrama de clases del dominio
- Estructuras de datos y Algoritmos

### - **Codificación y Tests:**

- Dominio (modelo datos) completamente implementado y probado
- Código de las funcionalidades principales (se concretarán cada cuatrimestre en el enunciado) implementado y probado
- Drivers, stubs/mocks*, tests (JUnit) + Juegos de pruebas

# Ciclo de Vida

## Segunda entrega proyecto:

### **- Diseño completo**

- Arquitectura en 3 capas completa (añadimos interfície, persistencia, controladores, ...)
- Documentación definitiva de Estructuras de datos y Algoritmos



# Ciclo de Vida

## Tercera entrega proyecto:

### **- Proyecto completo**

- Completamente implementado, documentado y probado (añadimos interfície, persistencia, controladores, ...)
- Juegos de pruebas  
(diferentes estados del programa)  
(análisis de comportamiento de los algoritmos)
- Manual de usuario

# Rúbrica Evaluación: 1ª entrega

Precondición para aplicar estos criterios de evaluación:  
punto 2 de las Normes dels lliuraments de PROP

"El lliurament que sigui incomplet, que es faci després de la data prevista o en format incorrecte és equivalent al no lliurament"

1er Lliurament								
Casos d'Us		Model		Doc	Relació Classes	Codi		Testing
Diagrama	Descripció	Diagrama	Descripció	ED&ALG. Funcionalitat Ppal.	per Membre Grup	Model	Funcionalitat Ppal.	Jocs Prova +Junit
	20%		20%	20%	Sí/No		20%	20%
	50%		60%		40%			
					Informatiu, cal entregar o el lliurament no es considera complet	60%	40%	

# Rúbrica Evaluación: 2ª entrega

Precondición para aplicar estos criterios de evaluación:  
punto 2 de las Normes dels Lliuraments de PROP

"El lliurament que sigui incomplet, que es faci després de la data prevista o en format incorrecte és equivalent al no lliurament"

2on Lliurament			
Disseny			Doc
Capa Domini	Capa Presentació	Capa Persistència	ED&ALG
25%	25%	10%	40%
Es presenta el diagrama i es compleixen els criteris de arq 3 capes. Correcció de la compleció del disseny de la capa de domini respecte lliurament 1	Es presenta el diagrama i es compleixen els criteris de arq 3 capes. Correcció del disseny de l'interfície + ergonomia, usabilitat i estètica d'aquesta (desde el punt de vista del disseny)	Es presenta el diagrama i es compleixen els criteris de arq 3 capes. Correcció del disseny de la capa de persistència	Millores en estructures de dades i algoritmes respecte lliurament 1, info adicional



# Rúbrica Evaluación: 3ª entrega

Precondición para aplicar estos criterios de evaluación:  
punto 2 de las Normes dels Lliuraments de PROP

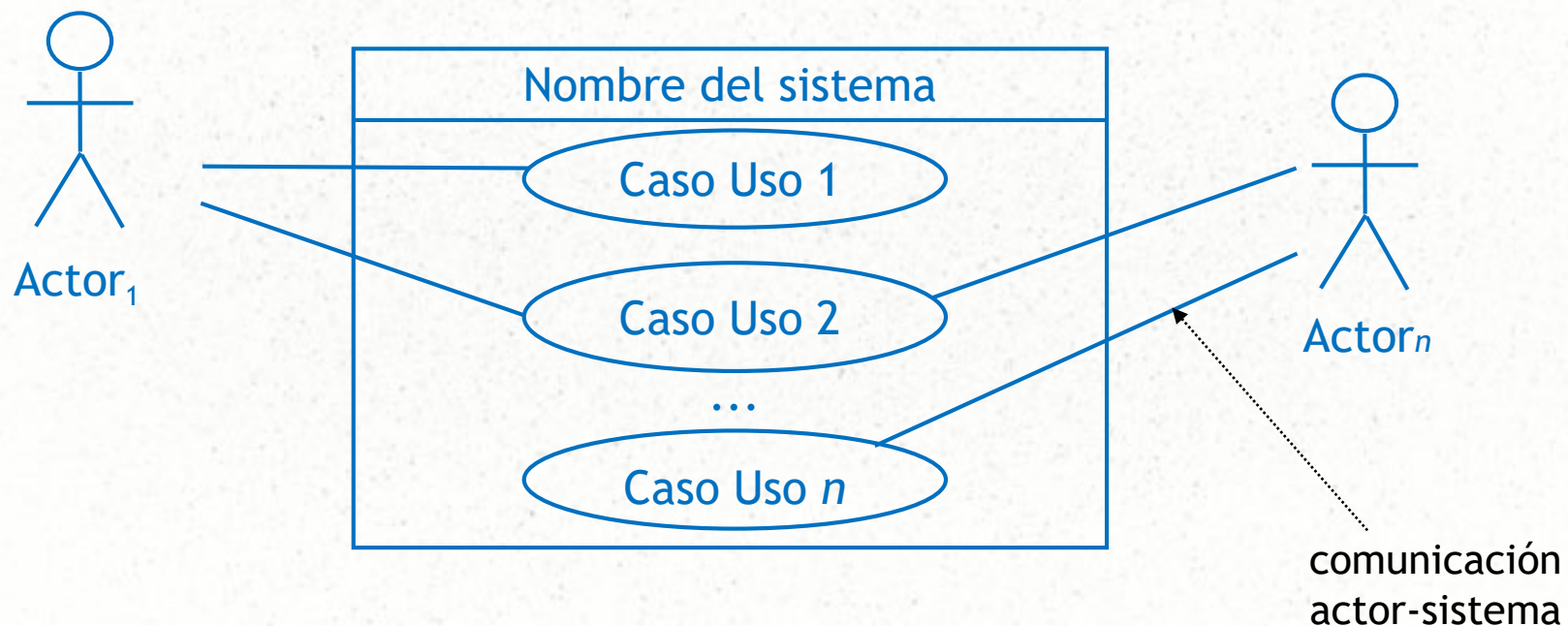
"El lliurament que sigui incomplet, que es faci després de la data prevista o en format incorrecte és equivalent al no lliurament"

3er Lliurament					
Relació Classes per Membre Grup Sí/No	Doc		Codi	Testing	
	Manual Usuari	Documentació Codi	Codi Projecte	Jocs Prova	Execució
	10%	10%	35%	20%	25%
Informatiu, cal entregar o el lliurament no es considera complet	Qualitat del manual	Qualitat de la documentació (independentment de si es fan servir o no eines de documentació automàtiques	Percentatge de les funcionalitats dels casos d'us implementades. Qualitat del codi	Documentació, qualitat i justificació dels jocs de prova presentats	Experiència d'usuari al executar el projecte (usabilitat, eficiència, ...)

# Diagrama de Casos de Uso

# Diagrama de Casos de Uso

Recoge SOLO la funcionalidad del sistema, los diferentes tipos de usuario (actores) y en qué funcionalidad participa cada tipo



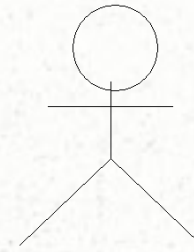


# Diagrama de Casos de Uso

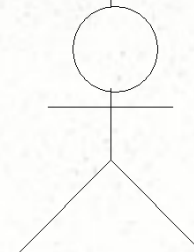
- Un **actor** es una entidad externa que participa en algún escenario de un caso de uso:
  - Persona/organización
  - Hardware
  - Otros sistemas software
- Un actor juega un cierto rol, y “dispara” los casos de uso. Tiene una responsabilidad hacia el sistema (entradas) y expectativas del sistema (salidas)
- Caso particular de actor: “Sistema/Reloj” (encargado de las funcionalidades que se ejecutarían de forma transparente a cualquier usuario externo) – Ej: copias de seguridad periódicas, carga de datos transparente al usuario

# Diagrama de Casos de Uso

- Los actores se pueden organizar en jerarquías: la descripción de un actor abstracto puede ser heredada y aumentada por descripciones de actores más específicas



Vendedor



Supervisor

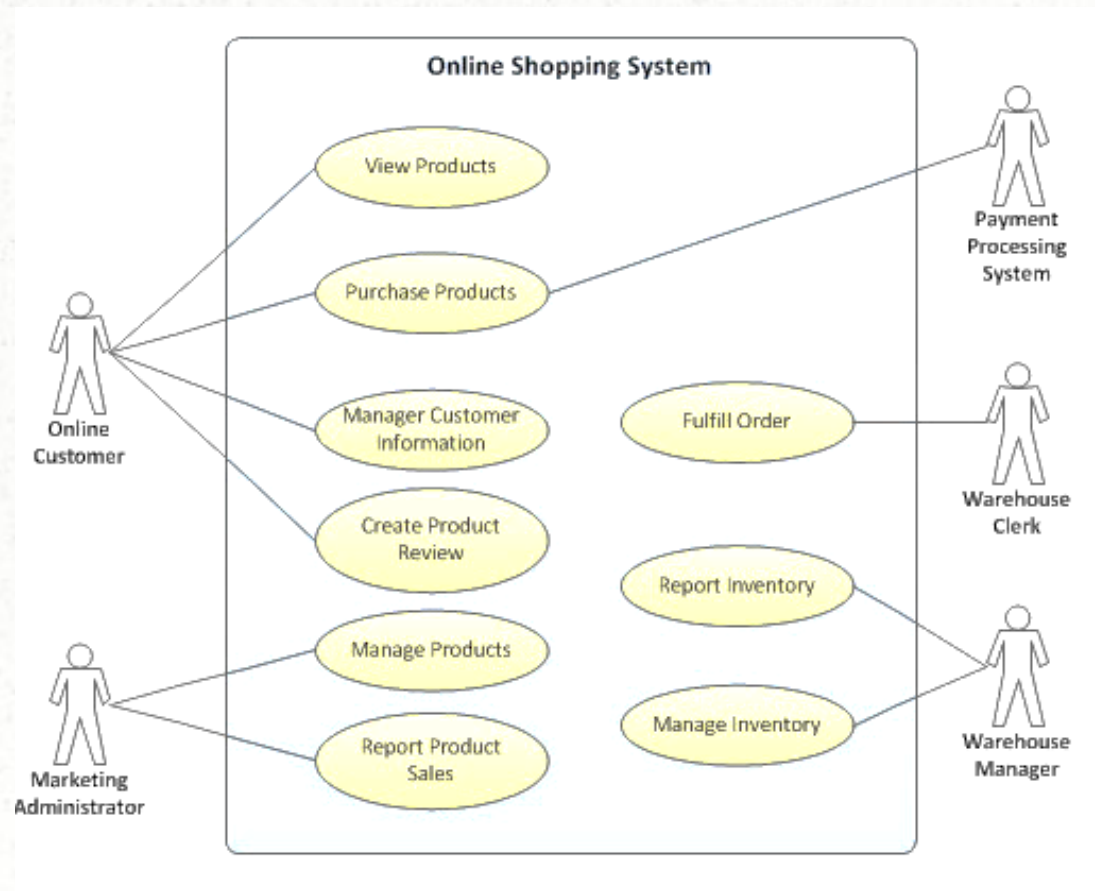
# Diagrama de Casos de Uso

- Un **caso de uso** es una funcionalidad del sistema accesible desde el exterior
- Para cada caso de uso, hay que describir el comportamiento observable del sistema cuando se ejecute (sin revelar la estructura interna del sistema)
- El comportamiento incluye:
  - Escenario principal (comportamiento normal)
  - Escenarios alternativos (comportamientos anómalos/erróneos)



# Diagrama de Casos de Uso

## Ejemplo: Sistema de Ventas (I)

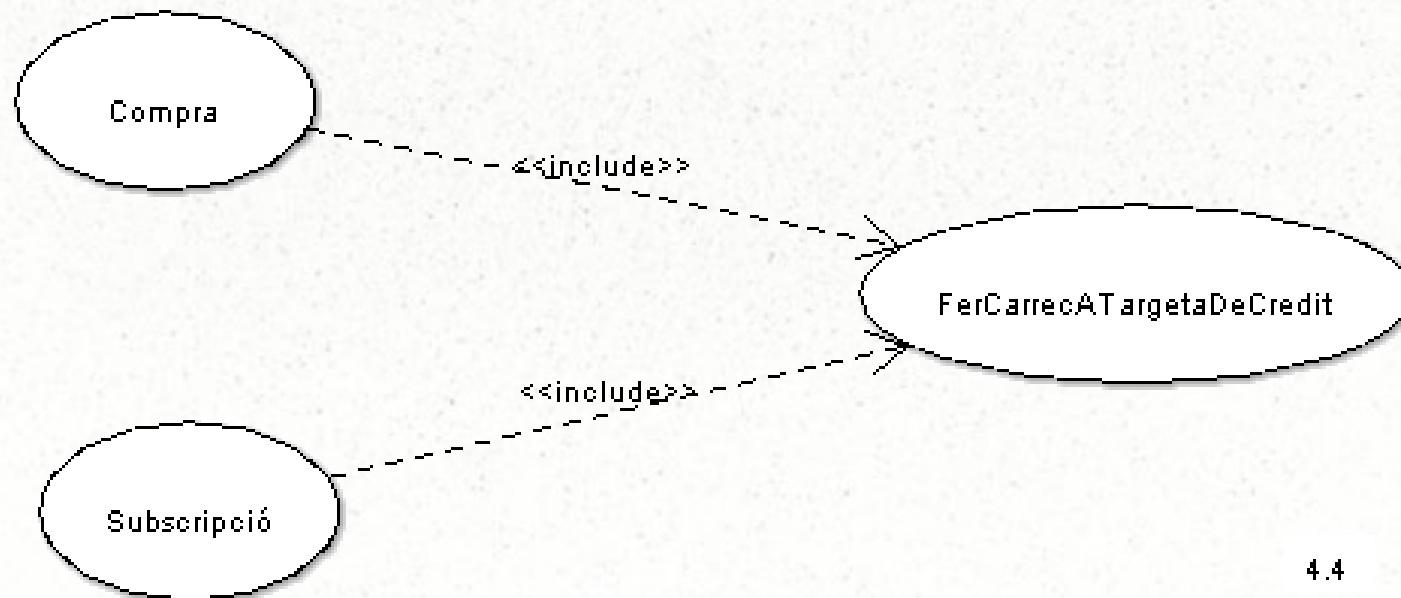


# Diagrama de Casos de Uso

- El diagrama de casos de uso describe las funcionalidades observables de forma estática, NO la relación dinámica entre ellas (que se describiría con otros artefactos UML)
- Aunque cada caso de uso es independiente, la descripción de un caso de uso puede factorizarse (de forma exhaustiva o no) en otros casos de uso más simples: relación "*includes*"
- Existen otras posibles relaciones entre casos de uso, pero no se contemplarán en PROP

# Diagrama de Casos de Uso

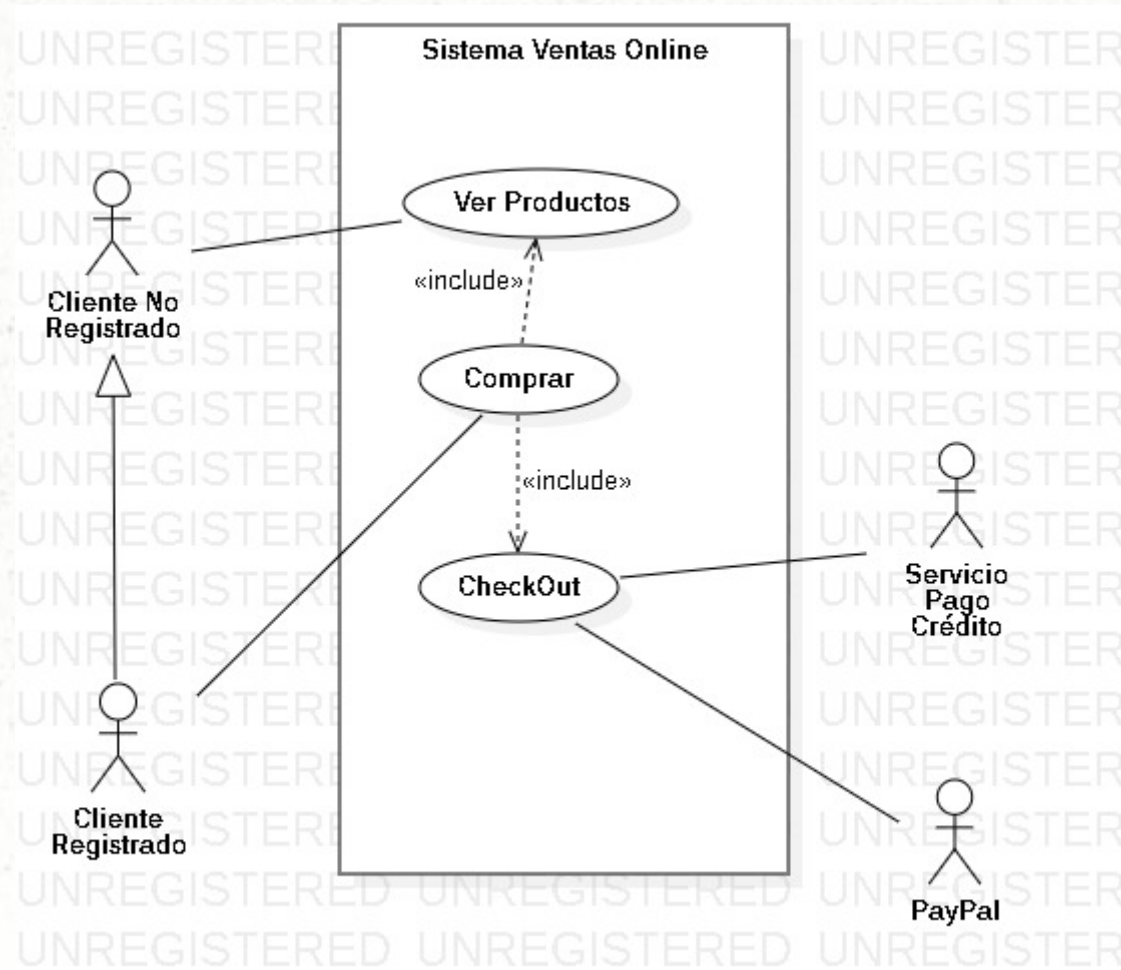
- Un caso de uso puede ser "included" por varios casos de uso más complejos, que usan la funcionalidad del caso de uso incluido -> incluyen el comportamiento especificado en el caso de uso incluido (se reusa el comportamiento común).





# Diagrama de Casos de Uso

## Ejemplo: Sistema de Ventas (II)



# Diagrama de Casos de Uso

Descripción de cada caso de uso en PROP, según la complejidad de cada caso concreto:

- Nivel breve (*brief*)

Resumen de un párrafo, normalmente del escenario principal

- Nivel informal (*casual*)

Diversos párrafos que cubren diferentes escenarios

# Diagrama de Casos de Uso

Descripción de cada caso de uso en PROP

Basta con que se cubra la siguiente información:

- Breve descripción del comportamiento observable del sistema cuando se ejecuta, incluyendo:
  - Qué datos de entrada se han de proporcionar
  - Qué datos de salida se obtienen
- Errores posibles y cursos alternativos (situaciones poco habituales) del caso de uso



# Diagrama de Casos de Uso

Descripción de cada caso de uso en PROP

Se puede opcionalmente seguir esta plantilla:

Actor principal: Entidad externa que interactúa con el Sistema

[Precondición]: Condición previa necesaria para poder ejecutar el caso de uso

[Detonante]: Condición que desencadena el caso de uso

Escenario principal: Secuencia de acciones entre el(los) actor(es) y el sistema

[Extensiones]: Alternativas y tratamiento de situaciones anómalas

# Diagrama de Casos de Uso

Ejemplo 1 de descripción de caso de uso:

## Caso uso #2 - *Log in*

Actor: Usuario registrado

Precondición: El usuario está registrado y no se ha *loggeado*

Detonante: El usuario quiere *loggearse*

Escenario Principal:

- 1) El usuario proporciona *username* y *password*
- 2) El sistema valida valores y proporciona acceso

Extensiones:

- 1a) El usuario no recuerda *password*: el sistema comienza el caso de uso #7 – cambio de *password*
- 2a) No existe un usuario con esas credenciales: el sistema avisa del error y vuelve al paso 1



# Diagrama de Casos de Uso

Ejemplo 2 de descripción de caso de uso:

## Alta de Usuario

Actor: Usuario no registrado

### Comportamiento:

El actor elige hacer un alta, ha de entrar el nombre y apellidos del usuario, el código de usuario (*username*), la contraseña (*password*) –dos veces– y el tipo de usuario –que se escoge de una lista. El sistema valida valores y coherencia de los datos, y los registra.

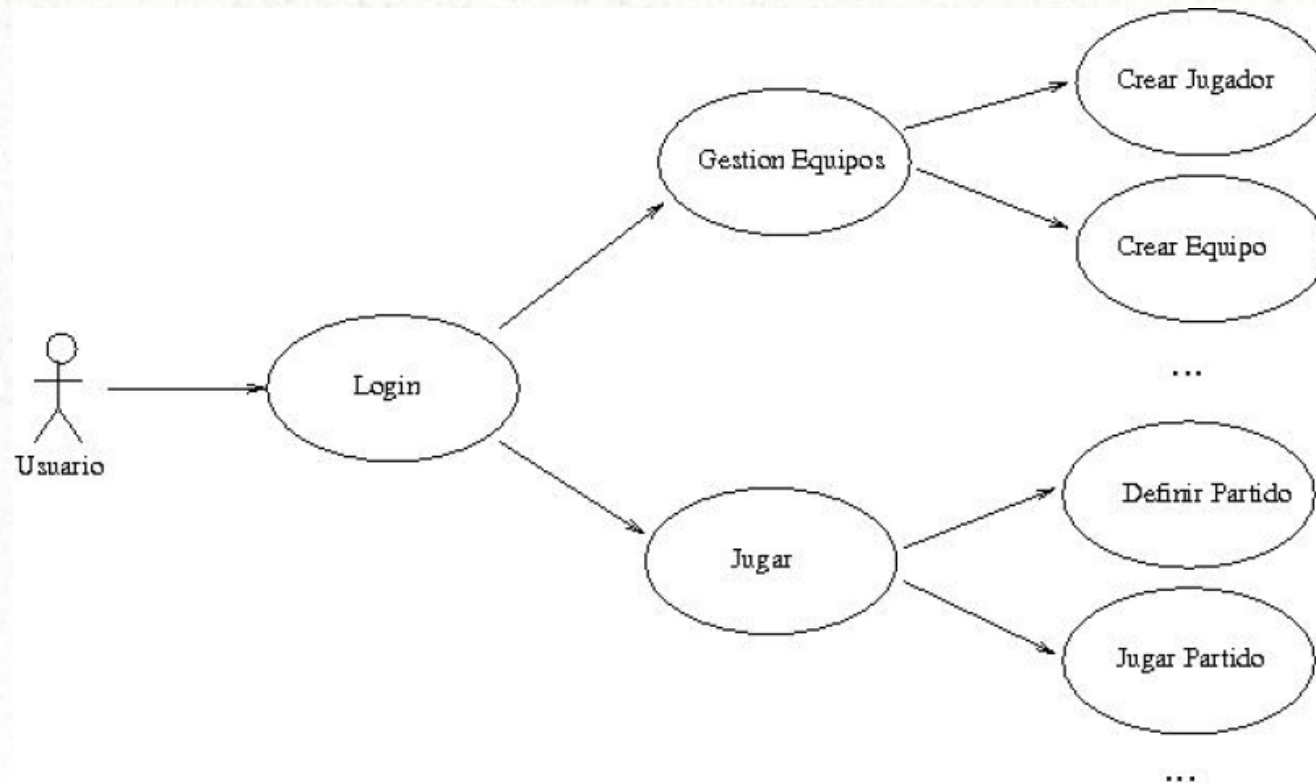
### Errores posibles y cursos alternativos:

Este código de usuario ya existe: cambiarlo o abandonar  
Las dos contraseñas no coinciden: volver a introducirlas



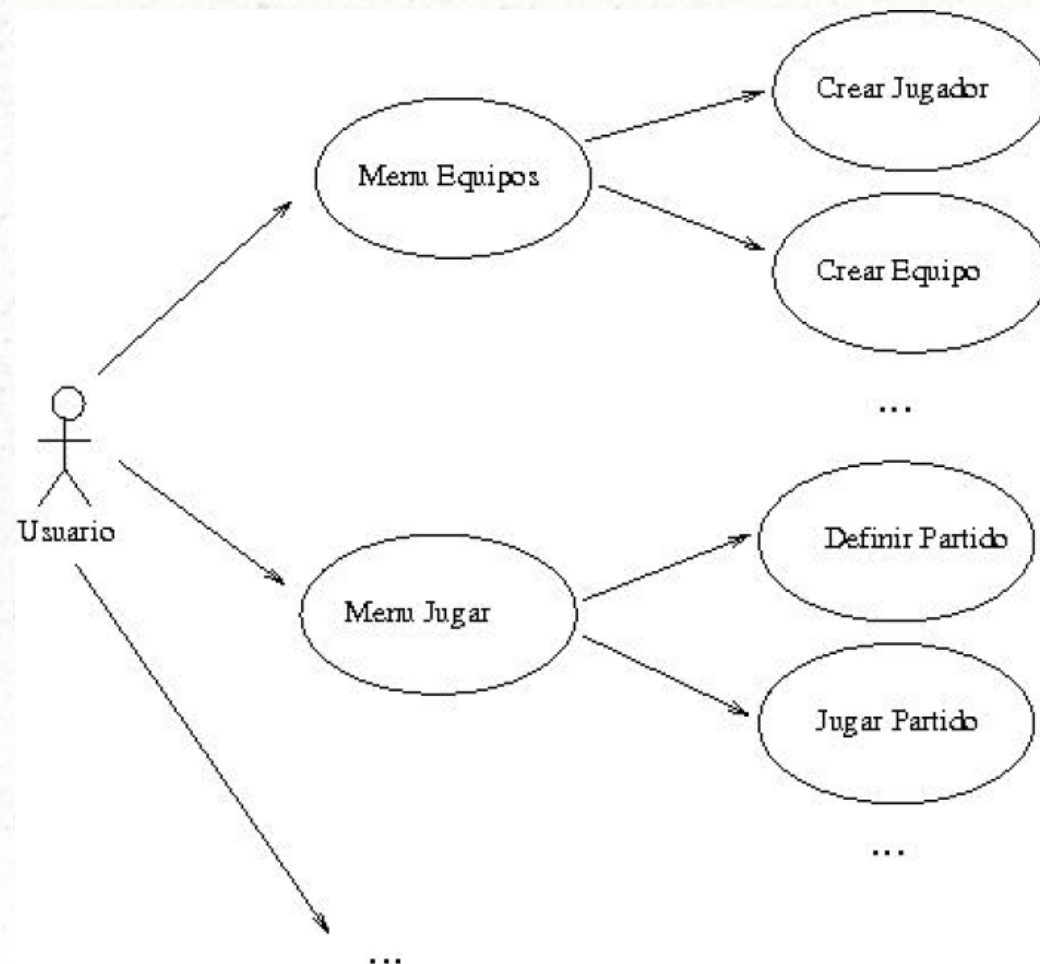
# Diagrama de Casos de Uso

**No** es un diagrama de casos de uso:  
Secuencias temporales



# Diagrama de Casos de Uso

**No** es un diagrama de casos de uso:  
Interfícies de  
usuario



# Diagrama de Casos de Uso

Por dónde empezar? A partir del enunciado (e info adicional recopilada):

Nombres  $\Rightarrow$  clases

Adjetivos  $\Rightarrow$  atributos

Verbos  $\Rightarrow$  relaciones y casos de uso

Objetivos:

- Identificar las funcionalidades de la aplicación
- Asociar las funcionalidades del software A CADA ACTOR



# Diagrama de Casos de Uso

## Cómo identificar actores?

- Quién usa el sistema?
- Quién instala el sistema?
- Quién lanza el sistema?
- Quién mantiene el sistema?
- Quién cierra el sistema?
- Qué otros sistemas usan el sistema?
- Quién obtiene información del sistema?
- Quién proporciona información al sistema?
- Hay algo que ocurra automáticamente en algún momento?

(Schneider and Winters – 1998)

# Diagrama de Casos de Uso

## Cómo identificar casos de uso?

Preguntándonos qué valores observables externamente desearía cada actor.

- Qué funciones querrá el actor del sistema?
- El sistema guarda información? Qué actores crearán, leerán, actualizarán o borrarán esa información?
- Necesita el sistema notificar a un actor acerca de cambios en el estado interno?
- Existen eventos externos que el sistema necesite conocer? Qué actor informa al sistema sobre esos eventos?

(Schneider and Winters – 1998)

# Diagrama de Clases



# Diagrama de Clases

Por dónde empezar? A partir del enunciado (e info adicional recopilada) + la descripción de los casos de uso:

Nombres  $\Rightarrow$  clases

Adjetivos  $\Rightarrow$  atributos

Verbos  $\Rightarrow$  relaciones y casos de uso

Posteriormente, refinar para obtener abstracciones con entidad propia

El diagrama de clases REPRESENTA LA REALIDAD y debe ser REUSABLE

# Diagrama de Clases

Todos los elementos que se entregan tienen que ser CONSISTENTES entre sí:

- Todas las clases del diagrama del modelo tienen que ser usadas en los casos de uso
- Todas las funcionalidades de los casos de uso deben estar relacionadas con clases

# Diagrama de Clases

El modelo conceptual de los datos es la representación de los conceptos (objetos) significativos en el dominio del problema.

Sugerencia: Empezar con esta primera versión "Especificación":

- Sólo ATRIBUTOS Y RELACIONES (métodos no)
- Sin normalizar (si se quiere sí, pero no es obligado)
- Tipos independientes de la tecnología ("conjunto de", ...)



# Diagrama de Clases

Es importante mantener un buen equilibrio entre las responsabilidades de las clases:

- Clases demasiado grandes nos darán modelos difíciles de modificar y poco reusables
- Clases demasiado pequeñas nos darán modelos que contienen más abstracciones de las que podemos manejar y entender razonablemente

Una vez tengamos clara la estructura y contenido del modelo, podemos empezar a traducir el diagrama a versión "diseño"

# Diagrama de Clases

## Relaciones:

- Generalización (o herencia)
- Asociación (o instancia)
- Agregación
- Composición (en PROP no se pide diferenciarla de la agregación)
- Dependencia (o mensaje)

# Diagrama de Clases

Relación de dependencia (o mensaje):

- Semántica: "usa"
- Conexión puntual y variable en el tiempo
- Existe para que un objeto de una clase pueda ejecutar los servicios de otro o pueda usar objetos de ese tipo como variables locales/parámetros de sus operaciones
- Notación: Flecha con la línea punteada desde la clase que usa a la usada:

```
graph LR; Alumno -.-> CalculadorNotas
```

Alumno - - - - - > CalculadorNotas