

Projecte de Programació

Curs 2018/19 – Q1

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

- Clases de teoría

Explican lo necesario para desarrollar el Proyecto:
conceptos OO, Java, Arquitectura 3 capas, etc.

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

- Clases de laboratorio

(*comienzan la semana del 17 de septiembre*):

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 \geq JDK8


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 $\geq 4.*$)

- Examen teoria: **21/11** 12:30h – 14:30h
- Nota Final PROP:

80% Nota proyecto
20% Nota examen

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

- Proyecto: DOS entregas.

Entrega 1 (**16/11 17h**):

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

Entrega 2 (**21/12 17h**):

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

- Nota Proyecto:

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

La nota individual del proyecto es:

$$FT \times (0.4 \times N_1 + 0.6 \times N_2)$$

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 grupos de 3 personas cada grupo (o puntualmente 4 si no cuadra la matriculación)

Hay un solo enunciado para todos los grupos y *clusters*. Sale publicado durante la 1ª semana de laboratorio del curso.

Está estrictamente prohibido formar grupos 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

No se prefija ninguna herramienta concreta (decisión de cada grupo)

- 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
- Imprescindible: herramienta de control de versiones
(se crearán cuentas GitLab para cada grupo, aunque se puede usar otra herramienta cualquiera)
- Opcional: herramienta de documentación (Doxygen, Javadoc)
(objetivo: reusabilidad)

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 & Stubs
- Prueba de programas: Testing con Junit
- Patrones con Java
- Diseño de Interfícies con Java

- Bibliografia:

An Introduction to Object Oriented Programming

Timothy Budd

Addison-Wesley 2002

The Object Oriented Thought Process (3rd. ed)

Matt Weisfeld

Pearson Ed. 2009

JUnit in Action (2nd. ed.)

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

Manning , 2011

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

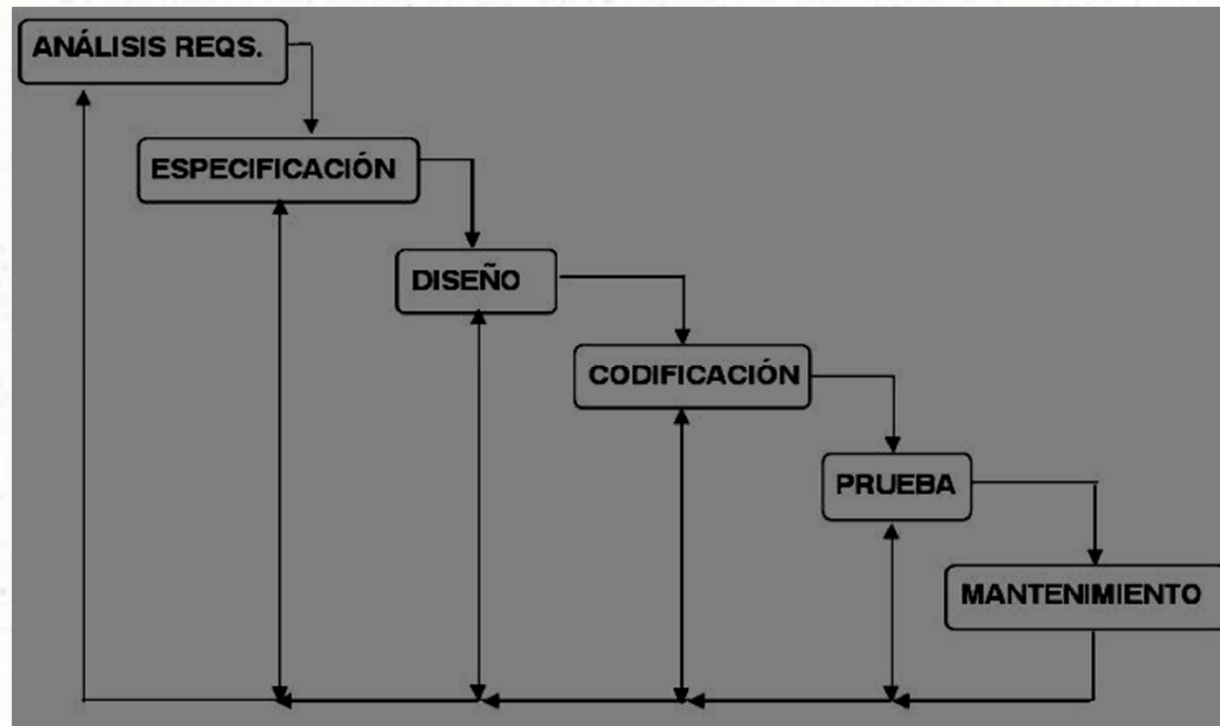
Ágil

etc...

(http://en.wikipedia.org/wiki/Software_development_process)

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 grupo y tutor

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



En la segunda 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*).

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) implementado y probado
- Drivers, stubs, tests (JUnit) + Juegos de pruebas

Ciclo de Vida

Segunda entrega proyecto:

- **Proyecto completo**

- Completamente implementado, documentado y probado (añadimos interfície, persistencia, controladores, ...)
- Juegos de pruebas (diferentes estados del programa)
- Manual de usuario
- Documentación definitiva de Estructuras de datos y Algoritmos (análisis de comportamiento de éstos)

Diagrama de Casos de Uso

Diagrama de Casos de Uso

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

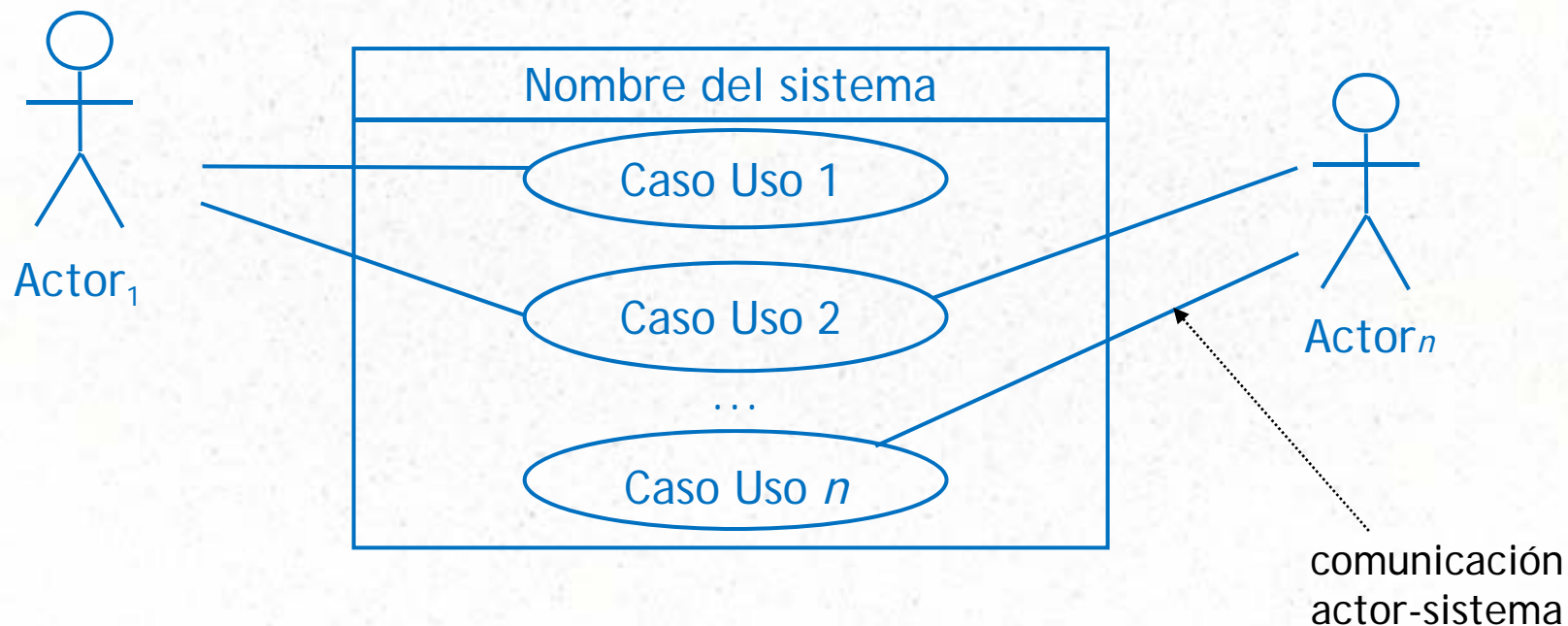


Diagrama de Casos de Uso

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

- Un actor es una entidad externa que participa en algún escenario de un caso de uso:

Persona/organización

Hardware

Otros sistemas software

- 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

Diagrama de Casos de Uso

Ejemplo: Sistema de Ventas

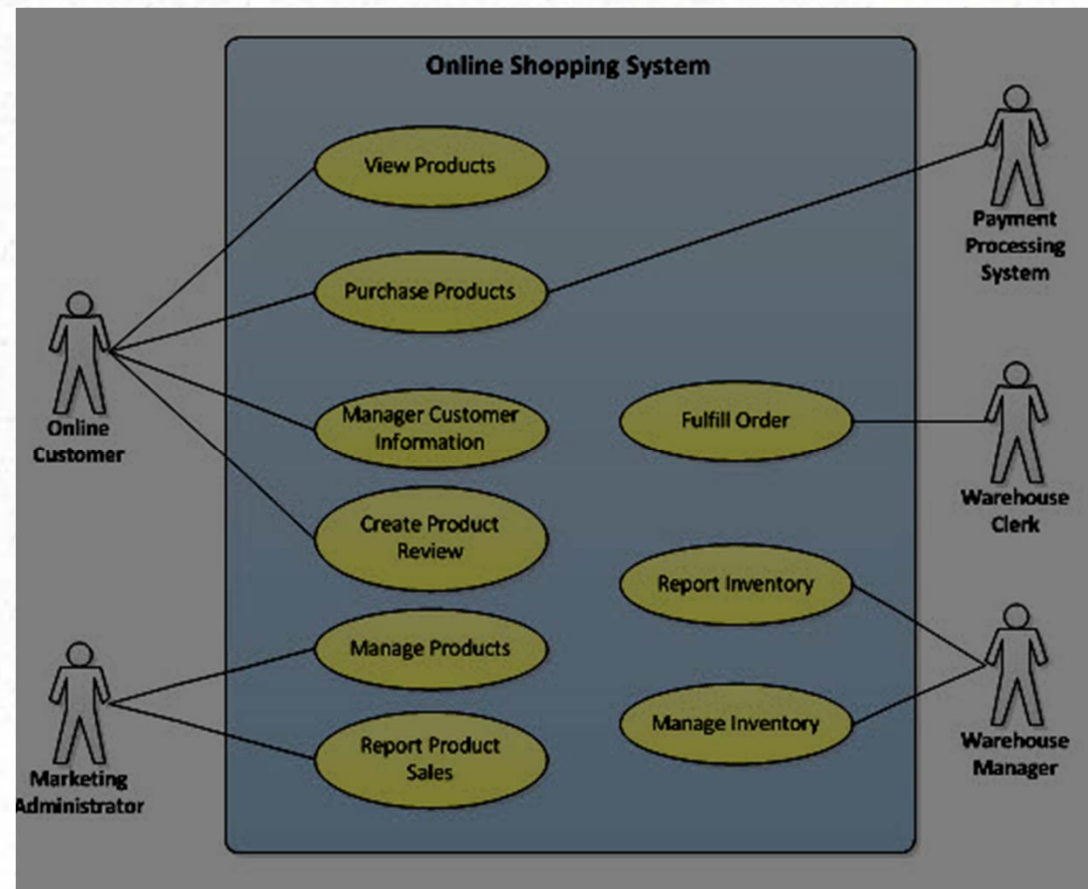
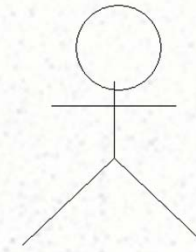
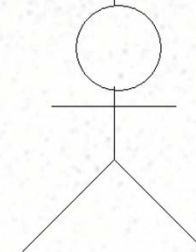


Diagrama de Casos de Uso

- 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

- 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 "incluido" por varios casos de uso más complejos

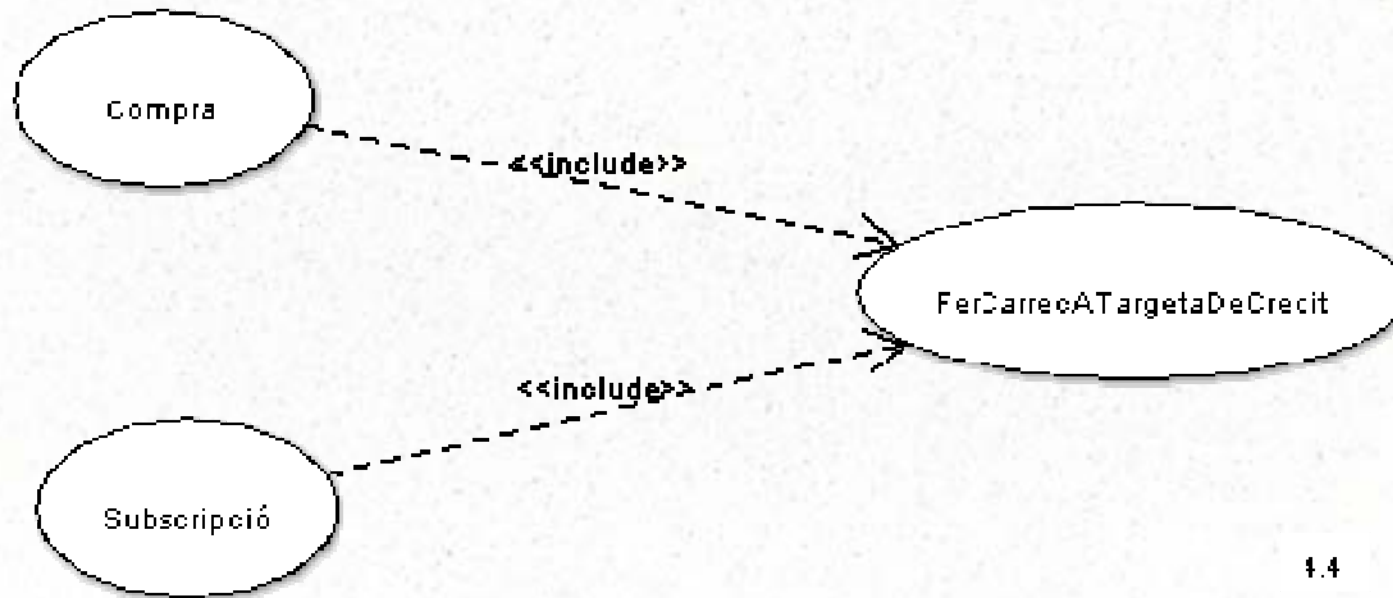
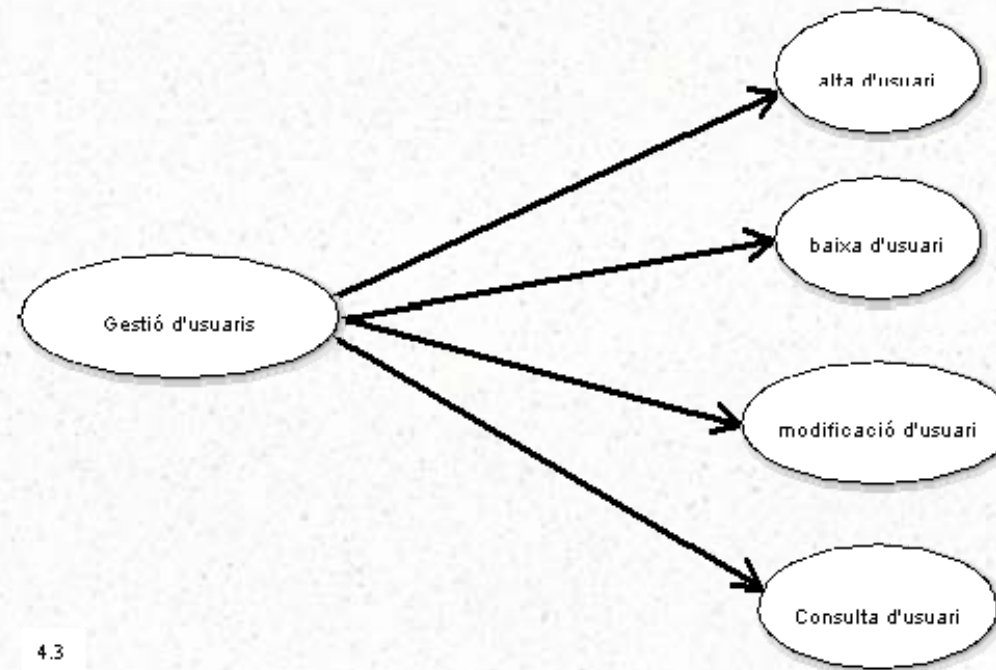


Diagrama de Casos de Uso

- Otro ejemplo de relación "*includes*": Gestión de usuarios



4.3

- La descripción de cada caso de uso también se factoriza

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

Ejemplo de descripción de caso de uso:

Alta de Usuario

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

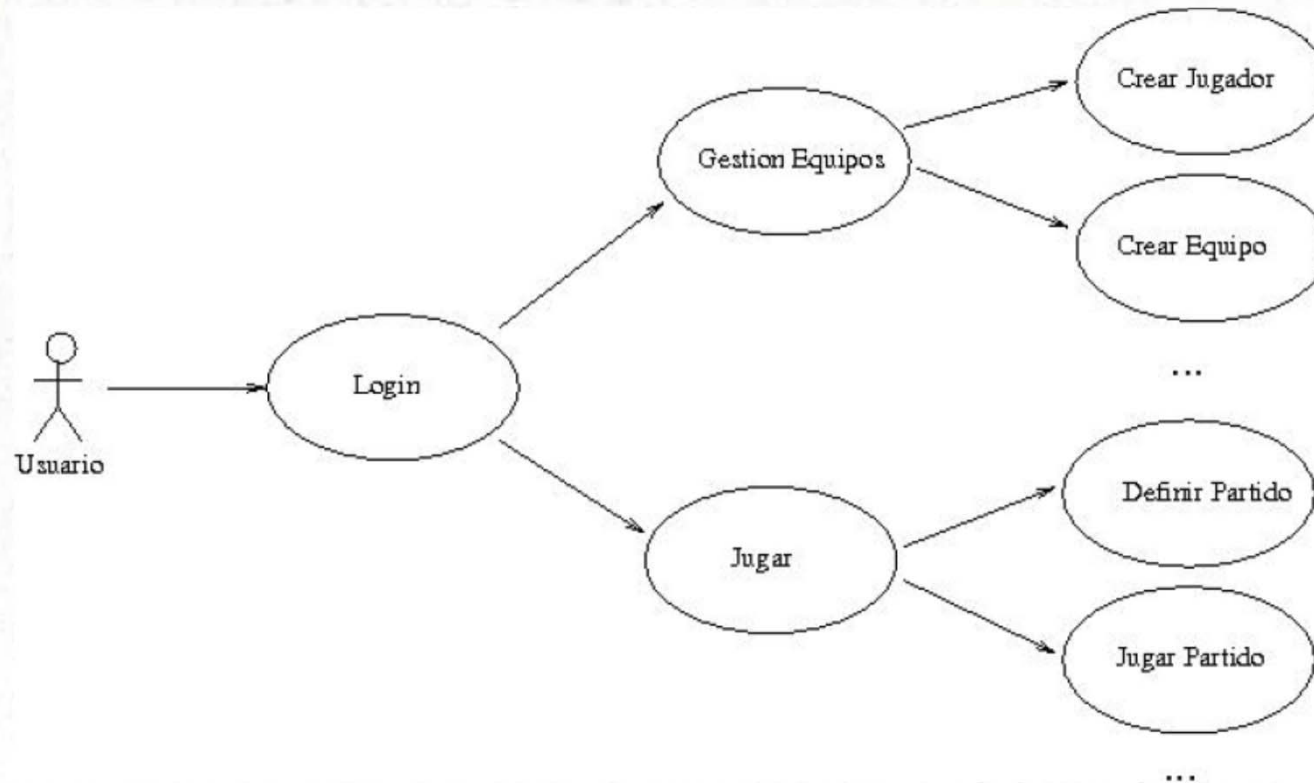


Diagrama de Casos de Uso

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

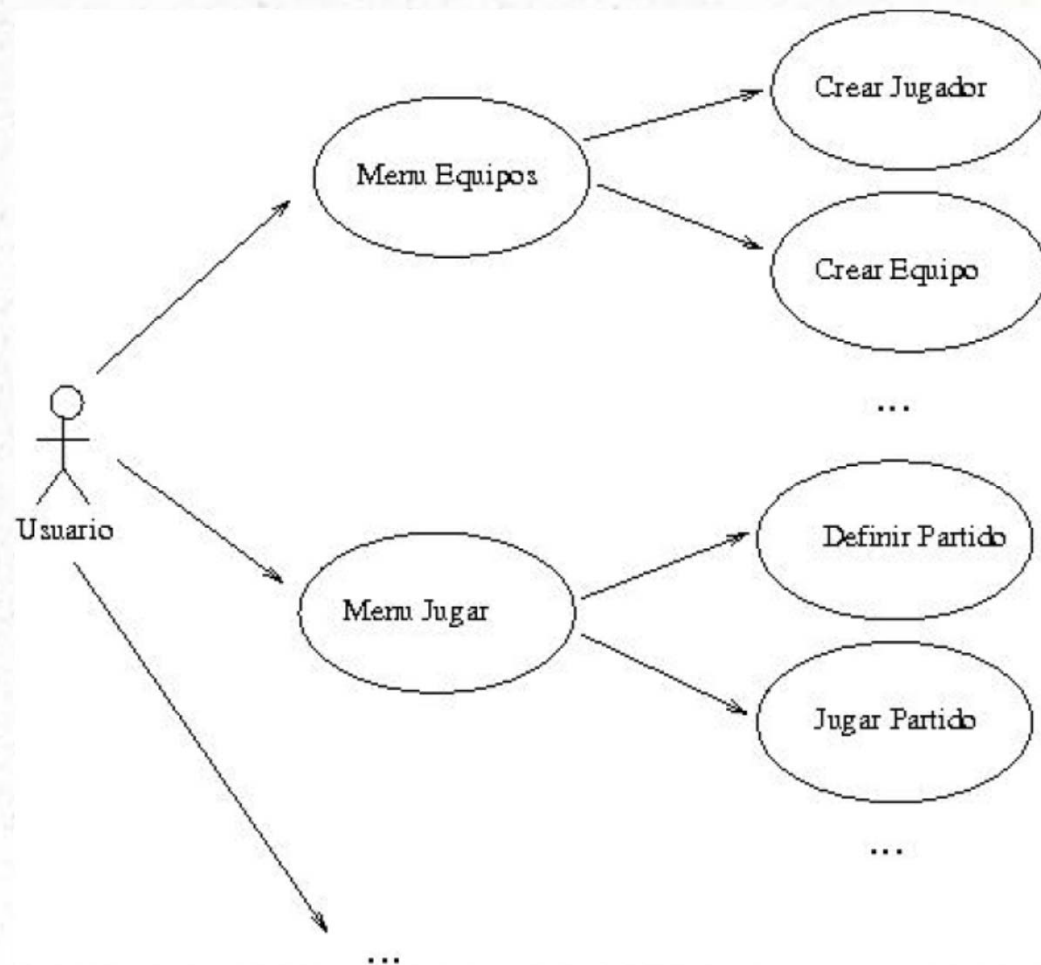


Diagrama de Clases

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:

Alumno - - - - - > CalculadorNotas