Asignatura	Datos del alumno	Fecha
Nombre de la asignatura ENTORNOS DE DESARROLLO	Apellidos:	
	Nombre:	

AD-3. Diagramas UML

Lee atentamente todo el documento antes de comenzar a trabajar.

Para esta actividad, solamente debes desarrollar la parte de UML (de los Pasos a seguir para realizar la actividad, solo del 1 al 4 y el 7). No hay que codificar nada, esa parte te la dejo a ti para que puedas practicar.

Objetivos

A través de esta actividad, vas a desarrollar un sistema de gestión de torneos de eSports, aplicando Programación Orientada a Objetos (POO) y utilizando diagramas UML para modelar su estructura y funcionalidad.

Con esta actividad, conseguirás:

1. Comprender y aplicar el modelado UML: Representar la estructura y funcionalidad del sistema mediante diagramas de clases y de casos de uso.
2. Analizar los requisitos del sistema: Identificar actores, casos de uso y relaciones en un sistema realista.
3. Definir las entidades y sus relaciones: Diseñar un diagrama de clases UML, estableciendo correctamente atributos, métodos y asociaciones.
4. Diferenciar los tipos de clases en un proyecto: Organizar las clases en Entidades, Control e Interfaz para una arquitectura modular.
5. Traducir el diseño UML a código Java: Implementar el sistema en Java, siguiendo buenas prácticas de encapsulamiento y modularidad.
6. Fomentar la capacidad de análisis y abstracción: Aprender a modelar sistemas software desde una perspectiva estructural y funcional.

© Universidad Internacional de La Rioja (UNIR)

Asignatura	Datos del alumno	Fecha
Nombre de la asignatura ENTORNOS DE DESARROLLO	Apellidos:	
	Nombre:	

Al finalizar, habrás adquirido una comprensión integral del proceso de diseño y desarrollo de software, aplicando técnicas fundamentales para el análisis y la implementación de sistemas en proyectos reales.

Pautas de elaboración

¿En qué consiste la actividad?

En esta actividad, diseñarás y desarrollarás un sistema de gestión de torneos de eSports, aplicando Programación Orientada a Objetos (POO) y utilizando diagramas UML para modelar su estructura y funcionalidad.

Para ello, trabajarás en dos aspectos clave del análisis y diseño de software:

1. **Modelado de funcionalidad con diagramas de casos de uso**
    - Representarás las principales interacciones entre los usuarios y el sistema.
    - Identificarás los actores (administradores, jugadores y sistema) y sus casos de uso.
    - Definirás relaciones como inclusión (<<include>>) y extensión (<<extend>>) en los casos de uso cuando sea necesario.
2. **Modelado estructural con diagramas de clases**
    - Diseñarás la estructura del sistema definiendo las clases principales y sus relaciones.
    - Diferenciarás los tipos de clases en Entidad, Control e Interfaz.
    - Establecerás atributos, métodos y asociaciones entre clases.

VOLUNTARIO: Tras la fase de diseño, implementarás el sistema en Java, organizando el código en paquetes y aplicando buenas prácticas de desarrollo. Finalmente,

© Universidad Internacional de La Rioja (UNIR)

documentarás el proyecto en el archivo README.md dentro de tu repositorio de GitHub, incluyendo los diagramas UML y una breve justificación del modelo.

Pasos a seguir para realizar la actividad

Para completar esta actividad de manera estructurada y eficiente, sigue estos pasos en orden secuencial.

1. Análisis del problema y requisitos del sistema

Antes de modelar, es fundamental comprender qué hace el sistema y qué se espera de él. Lee atentamente los requisitos y responde estas preguntas clave:

- ¿Quiénes son los actores que interactúan con el sistema?
- ¿Cuáles son las acciones que cada actor puede realizar?
- ¿Cómo se relacionan entre sí las entidades del sistema?

Consejo: Realiza una lluvia de ideas o un esquema rápido para visualizar las funcionalidades principales.

2. Identificación de los casos de uso y elaboración del diagrama

Ahora, debes definir qué interacciones existen entre los actores y el sistema.

A continuación, se presentan los casos de uso más relevantes del sistema. **Debes elaborar los correspondientes a la Gestión de equipos y jugadores,** el resto te los dejo por si quieres practicar:

1. **Gestión de equipos y jugadores**
    - Registrar equipo.
    - Añadir jugadores a un equipo.
    - Consultar lista de equipos y jugadores.
2. Gestión de torneos
    - Crear un torneo.

© Universidad Internacional de La Rioja (UNIR)

- Inscribir equipo en un torneo.
- Generar emparejamientos de partidas.

3. Gestión de partidas y resultados
   - Registrar resultado de una partida.
   - Actualizar clasificación del torneo.
4. Gestión de premios
   - Asignar premios a los ganadores.

Consejo: Piensa en qué casos de uso pueden estar relacionados mediante <<include>> o <<extend>>.

Dibuja el diagrama UML usando herramientas como draw.io, StarUML o PlantUML.

Te **recomiendo** que lo intentes primero a mano y cuando lo tengas claro, lo "transcribas" con la herramienta digital que tú prefieras.

3. Identificación de clases y relaciones

Una vez definidos los casos de uso, pasamos al modelo estructural del sistema.

1. Identifica las clases principales en función de los casos de uso seleccionados.
2. Distingue las clases de Entidad, Control e Interfaz para mantener una arquitectura modular.
3. Define atributos y métodos para cada clase.
4. Establece relaciones entre clases, asegurando la correcta representación de asociaciones, agregaciones y composiciones.

Consejo: Esboza primero un boceto antes de realizar el diagrama definitivo.

4. Creación del diagrama de clases UML

Ahora **debes representar gráficamente la estructura del sistema en UML**:

- Define las clases, atributos y métodos.

© Universidad Internacional de La Rioja (UNIR)

- Especifica las relaciones con cardinalidades y visibilidades.
- Usa asociaciones como agregación, composición y herencia cuando sea necesario.

5. Organización del código en paquetes y programación en Java

Una vez diseñado el sistema, implementa el código en Java, siguiendo una estructura organizada en paquetes:

```
src/
├── es/empresa/torneo/
│   ├── modelo/   # Clases de entidad
│   ├── control/  # Clases de control
│   ├── vista/    # Clases de interfaz de usuario (UI o consola)
│   ├── Main.java # Punto de entrada del programa
```

Consejo: Implementa primero las clases de Entidad, luego las clases de Control y por último la Interfaz.

6. Pruebas y depuración del sistema

Antes de entregar el trabajo, asegúrate de que el sistema funciona correctamente:

- Crea una clase Main para probar la funcionalidad.
- Ejecuta diferentes escenarios de prueba (crear torneo, registrar equipos, etc.).
- Usa el depurador para verificar que los datos se procesan correctamente.
- Opcionalmente, te recomiendo elaborar pruebas unitarias con JUnit para comprobar el correcto funcionamiento de tus clases y métodos. Si decides incluir pruebas unitarias, agrégalas en un paquete `test/` dentro de tu repositorio de GitHub.

7. Elaboración del documento de entrega

© Universidad Internacional de La Rioja (UNIR)

Asignatura	Datos del alumno	Fecha
Nombre de la asignatura ENTORNOS DE DESARROLLO	Apellidos:	
	Nombre:	

Toda la documentación del proyecto, incluyendo los diagramas UML y la justificación del diseño, debe incorporarse en el archivo `README.md` dentro del repositorio de GitHub.

La documentación debe incluir:

- Diagrama de casos de uso con una breve descripción de cada caso seleccionado.
- Diagrama de clases UML con explicación de la estructura y relaciones.
- Justificación del diseño: Por qué se eligió esa estructura y cómo se organizan las clases.
- Conclusiones sobre el aprendizaje obtenido.

Resumen de pasos a seguir

Recuerda: 1, 2, 3, 4 y 7

1. **Análisis** del problema y requisitos: Identifica los actores y funcionalidades principales del sistema.
2. Elaboración del **diagrama de casos de uso**: Dibuja el diagrama UML mostrando actores y relaciones (<<include>>, <<extend>>).
3. **Identificación de clases y relaciones**: Define las clases principales y su categoría (Entidad, Control, Interfaz). Establece atributos, métodos y asociaciones entre clases.
4. Creación del **diagrama de clases UML**: Representa la estructura del sistema con clases, atributos, métodos y relaciones.
5. **OPCIONAL:** Programación en Java: Organiza el código en paquetes: modelo, control, vista.
6. **OPCIONAL:** Implementa primero las clases de Entidad, luego las de Control y finalmente la Interfaz.
7. **OPCIONAL:** Pruebas y depuración: Crea una clase Main para comprobar el funcionamiento. Ejecuta diferentes escenarios de prueba y depura errores.

© Universidad Internacional de La Rioja (UNIR)

Asignatura	Datos del alumno	Fecha
Nombre de la asignatura ENTORNOS DE DESARROLLO	Apellidos:	
	Nombre:	

8. Entrega de la **documentación (README.md):** Incluye los diagramas UML con una breve explicación. Justifica el diseño elegido y describe las conclusiones del trabajo.

© Universidad Internacional de La Rioja (UNIR)

Asignatura	Datos del alumno	Fecha
Nombre de la asignatura ENTORNOS DE DESARROLLO	Apellidos:	
	Nombre:	

Extensión y formato

Debes entregar un repositorio en GitHub en el que publicarás tu proyecto. Para ello debes seguir las siguientes instrucciones:

Publicación del trabajo en GitHub

1. Crear un repositorio en GitHub con el nombre:

```
torneo-esports-uml
```

2. Estructura mínima del repositorio:

```
torneo-esports-uml/
├── src/                                    # Código fuente
│   ├── es/empresa/torneo/        # Paquete base del proyecto
│   │   ├── modelo/               # Clases de entidad
│   │   ├── control/              # Clases de control
│   │   ├── vista/                # Clases de interfaz (UI o consola)
│   │   ├── Main.java             # Punto de entrada del programa
├── diagrams/                     # Diagramas UML
│   ├── casos-uso.png             # Diagrama de casos de uso
│   ├── clases.png                # Diagrama de clases
├── README.md                           # Documentación del proyecto
├── .gitignore                                  #    Ignorar    archivos
innecesarios
├── LICENSE (opcional)                  # Licencia del proyecto
```

3. Elaborar el README.md como documentación del proyecto.

Contenido del README.md

© Universidad Internacional de La Rioja (UNIR)

Asignatura	Datos del alumno	Fecha
Nombre de la asignatura ENTORNOS DE DESARROLLO	Apellidos:	
	Nombre:	

El archivo README.md servirá como documentación principal del proyecto y debe incluir la siguiente estructura:

```
# Sistema de Gestión de Torneos de eSports

## Autor
Tu nombre completo
Tu perfil de GitHub

## Descripción del Proyecto

Link al repositorio del proyecto

Este proyecto implementa un sistema de gestión de torneos de eSports
utilizando UML para el modelado y Java para la implementación.

## Diagramas UML
### Diagrama de Casos de Uso
![Diagrama de casos de uso](diagrams/casos-uso.png)

### Diagrama de Clases
![Diagrama de clases](diagrams/clases.png)

## Estructura del Proyecto

torneo-esports-uml/ ├── src/
│ ├── es/empresa/torneo/
│ │ ├── modelo/
│ │ ├── control/
│ │ ├── vista/
│ │ ├── Main.java
├── diagrams/
│ ├── casos-uso.png
│ ├── clases.png
├── README.md
├── .gitignore
```

© Universidad Internacional de La Rioja (UNIR)

```
├── LICENSE (opcional)



## Instalación y Ejecución
1. Clonar el repositorio:
`git clone https://github.com/usuario/torneo-esports-uml.git`

2. Compilar y ejecutar el proyecto:
`cd src javac es/empresa/torneo/Main.java java es.empresa.torneo.Main`

## Justificación del diseño
Por qué se eligió esa estructura y cómo se organizan las clases.

## Conclusiones
Sobre el aprendizaje obtenido.
```

Nomenclatura estándar

Para mantener consistencia en todos los proyectos, se establecen las siguientes normas:

Nombres de paquetes

Todos los paquetes deben seguir la convención en minúsculas y estructurados de la siguiente manera:

```
es.empresa.torneo.modelo   # Clases de entidad (Torneo, Equipo, Jugador, etc.)
es.empresa.torneo.control  # Clases de control (GestorTorneos, GestorEquipos, etc.)
es.empresa.torneo.vista    # Clases de interfaz (UI o consola)
```

Nombres de archivos y clases

Cada archivo debe llevar el mismo nombre que la clase y la extensión .java.

```
Torneo.java
```

© Universidad Internacional de La Rioja (UNIR)

```
Equipo.java
Jugador.java
GestorTorneos.java
VistaPrincipal.java
Main.java
```

Nombres de clases y métodos

Las clases deben seguir PascalCase, los métodos camelCase y los atributos deben ser privados con getters/setters.

```
public class Torneo {
    private String nombre;
    private List<Equipo> equipos;

    public Torneo(String nombre) {
        this.nombre = nombre;
        this.equipos = new ArrayList<>();
    }

    public String getNombre() {
        return nombre;
    }

    public void agregarEquipo(Equipo equipo) {
        equipos.add(equipo);
    }
}
```

Nombres de diagramas UML

Los diagramas deben tener nombres descriptivos y estar en la carpeta /diagrams/.

```
casos-uso.png
clases.png
```

© Universidad Internacional de La Rioja (UNIR)

Asignatura	Datos del alumno	Fecha
Nombre de la asignatura ENTORNOS DE DESARROLLO	Apellidos:	
	Nombre:	

Entrega final

1. Subir el código fuente en GitHub respetando la estructura establecida.
2. Los diagramas UML deben guardarse en la carpeta /diagrams/ con los siguientes nombres estándar:
    1. Diagrams/
        1. teamRegister-useCase.png
        2. addPlayer2Team-useCase.png
        3. viewListTeamsAndPlayers-useCase.png
    2. diagrams/clases.png para el diagrama de clases.
3. Documentar el proyecto en README.md.
4. Compartir la URL del repositorio en el campus.

© Universidad Internacional de La Rioja (UNIR)

Asignatura	Datos del alumno	Fecha
Nombre de la asignatura ENTORNOS DE DESARROLLO	Apellidos:	
	Nombre:	

Rúbrica

Criterio de Evaluación	Descripción	Puntos	Peso %
1. Publicación en GitHub	El repositorio está correctamente creado y estructurado según las normas establecidas (carpetas, archivos, README.md, .gitignore).	1.0	10%
2. Documentación en README.md	Explica claramente el proyecto, incluye los diagramas UML y proporciona instrucciones para ejecutar el código.	1.0	10%
3. Diagrama de Casos de Uso	Representa correctamente los actores, casos de uso y sus relaciones (<<include>>, <<extend>> si aplica).	2.0	20%
4. Diagrama de Clases	Define las clases principales, sus relaciones, atributos y métodos. Aplica correctamente herencia, agregación o composición si es necesario.	2.0	20%
5. Estructura del Código	El código está organizado en paquetes (modelo, control, vista) y sigue la nomenclatura estándar de clases, métodos y atributos.	1.5	15%
6. Implementación en Java	El código es funcional, bien estructurado y sigue buenas prácticas de POO (encapsulación, modularidad, reutilización de código).	1.5	15%
7. Pruebas y Ejecución	Se proporciona una Main.java que demuestra el funcionamiento del sistema con pruebas de las funcionalidades implementadas.	1.0	10%
		10	100%

© Universidad Internacional de La Rioja (UNIR)