Programación Orientada a Objetos
Curso 2025-2026

Monopoly ETSE - Parte 3

En esta tercera entrega del proyecto se introducirán los conceptos de **herencia** y **polimorfismo**, para favorecer el mantenimiento y modificación del juego con el fin de que pueda ser evolucionado de manera más sencilla. Además, en esta entrega se completará la implementación del Monopoly ETSE con la **gestión de los tratos** entre los jugadores. Los requisitos que se evaluarán serán los siguientes:

Requisitos de diseño	
25	**Crear la clase Juego**. La clase *Juego* deberá de tener todas las instancias necesarias para el desarrollo del juego y que puede que se encuentren en la clase principal o en le clase menú, como, por ejemplo, la lista de jugadores o la instancia del propio tablero. Por tanto, en la clase principal **no** se deberán instanciar ninguno de los elementos del juego ni tampoco las operaciones relacionadas con la resolución de los comandos introducidos por el jugador. Si en las entregas anteriores del proyecto ya se había incluido una clase con estas características, se renombrará con el nombre *Juego* y se puntuará en esta entrega.
26	**Jerarquía de casillas**. Las casillas deberán estar jerarquizadas. Habrá una clase abstracta llamada *Casilla*, en la que se definan los métodos y atributos comunes a todas las casillas; un segundo nivel compuesto por las clases *Propiedad*, *Accion*, *Impuesto*, *Grupo* y *Especial*, donde la clase *Grupo* deberá tener una relación de composición con la clase *Propiedad*, es decir, un grupo tiene una lista de propiedades. Finalmente, existirá un tercer nivel de clases finales donde se representan los tres tipos de propiedades, *Solar*, *Servicio* y *Transporte*, así como los tipos de acciones, *Suerte, CajaComunidad y Parking*. La clase raíz *Casilla* deberá de tener **implementados**, por lo menos, los siguientes métodos: • *boolean estaAvatar(Avatar avatar)* • *int FrecuenciaVisita( )* • *String toString( )* Por otra parte, en la clase *Propiedad* se deberán de definir, al menos, los siguientes métodos: • *boolean perteneceAJugador(Jugador jugador)* • *abstract boolean alquiler( )* • *abstract float valor( )* • *void comprar(Jugador jugador)* Los métodos abstractos de la clase *Propiedad*, es decir, *alquiler* y *valor*, deberán ser implementados en las correspondientes subclases. Además, la clase *Solar* deberá implementar los siguientes métodos: • *void edificar(String tipoEdificio)* • *void hipotecar( )* • *boolean estaHipotecada( )* **Los métodos que sean iguales para los distintos tipos de subclases deben de estar implementados en la clase padre**.

27	**Jerarquía de edificios**. Los edificios deberán estar jerarquizados. Habrá una clase abstracta denominada *Edificio*, en la que se definirán los métodos y atributos comunes; y un segundo nivel de clases que estará compuesto por las clases *Casa*, *Hotel*, *Piscina* y *PistaDeporte*.
28	**Jerarquía de cartas**. Las cartas deberán de estar jerarquizadas. Habrá una clase raíz llamada *Carta* que defina los métodos y atributos comunes; y un segundo nivel de clases compuesto por las clases *Suerte* y *CajaComunidad*. La clase raíz *Carta* deberá de tener, por lo menos, el siguiente método: • *abstract void accion( )*
29	**Jerarquía de excepciones**. Los errores asociados a las acciones del usuario deberán de ser controlados con excepciones. Cuando se detecte una condición que impida realizar una acción (como comprar una propiedad que ya pertenece a otro jugador o hipotecar una propiedad que ya está hipotecada, etc.) se lanzará una excepción que **será capturada en el bucle principal** para poder indicarle el error al usuario. Se deberán de definir al menos cinco tipos propios de excepciones (a elección del grupo de prácticas) que se tratarán de manera diferente desde el bucle principal. Además, la jerarquía tendrá 3 niveles.
30	**Interface Comando**. Los comandos se representarán por una interface *Comando*. Esta interfaz tendrá un conjunto de métodos correspondientes a los comandos que pueden ser introducidos por el usuario (comprar, hipotecar, edificar, listar, etc.). La clase *Juego* deberá implementar esta interfaz, o lo que es lo mismo, implementará los métodos indicados en la interface *Comando*. Cuando el usuario introduzca un comando, únicamente se podrá invocar a los métodos correspondientes de la clase *Juego*.
31	**Interfaz Consola.** Esta interfaz debe declarar la funcionalidad de imprimir mensajes y de pedir datos. De esta forma, para mostrar mensajes al usuario **no** podrá haber llamadas al método *System.out.println* a lo largo del código del programa, salvo en el método de la clase que implementa el siguiente método: • *public void consola.imprimir(String mensaje)* donde *mensaje* es el mensaje que se desea mostrar al usuario. De la misma forma, cada vez que se tenga que preguntar al usuario por un dato, se utilizará el siguiente método: • *public String consola.leer(String descripcion)* donde *descripcion* es el mensaje que se le muestra al usuario antes de esperar a que introduzca los datos. Por ejemplo, *consola.leer("Introduce nombre: ")* debería de imprimir por pantalla *"Introduce nombre: "*, leer con *Scanner* lo que indique el usuario y devolver los datos introducidos como un *String*. Se proporcionará al menos una implementación de este interfaz, es decir, una clase que implemente los métodos *leer* e *imprimir* y que se llamará *ConsolaNormal*. En esta implementación se imprimirá usando *System.out* y se leerá utilizando, por ejemplo, la clase *Scanner* de Java (o la que se prefiera). Finalmente, una de los atributos de la clase *Juego* será un **atributo estático** del tipo *ConsolaNormal* para que los métodos de *imprimir* y *leer* se puedan invocar desde cualquier clase del programa sin tener que instanciar continuamente dicha clase.

Funcionalidades generales del juego	
32	**Proponer trato**. Un jugador podrá proponer un trato a otro jugador, aunque no se admitirán tratos en el que estén involucrados varios jugadores. Los tratos que se podrán proponer son los siguientes: • Cambiar *<propiedad_1>* por *<propiedad_2>* • Cambiar *<propiedad_1>* por *<cantidad_dinero>* • Cambiar *<cantidad_dinero>* por *<propiedad_1>* • Cambiar *<propiedad_1>* por *<propiedad_2>* y *<cantidad_dinero>* • Cambiar *<propiedad_1>* y *<cantidad_dinero> por <propiedad_2>* Si un trato no puede ser cerrado, porque la propiedad que se desea cambiar no pertenece al jugador al que se propone el trato o porque el jugador que propone propio el trato no tiene el dinero suficiente para comprar la propiedad, se lanzará una excepción indicando el problema y que el trato no puede llevarse a cabo. --- `$> trato Luis: cambiar (Solar14, Solar10)` `Luis, ¿te doy solar Solar14 y tú me das Solar10?` --- `$> trato Pedro: cambiar (Solar1, 2500000)` `Pedro, ¿te doy Solar1 y me das 2500000?` --- `$> trato Maria: cambiar (Solar1, Solar14 y 300000)` `María, ¿te doy Solar1 y tú me das Solar14 y 300000€?` --- `$> trato Maria: cambiar (Solar1, Solar14 y 300000)` `No se puede proponer el trato: Solar14 no pertenece a Maria.`
33	**Aceptar trato**. La aceptación de un trato no se lleva a cabo en el turno del jugador que propone dicho trato, sino en el turno del jugador al que se le ha propuesto. De este modo, las acciones indicadas en un trato no tendrán lugar durante el turno del jugador que lo ha propuesto. Cuando un jugador cambia de turno, se le deberán mostrar automáticamente los tratos que le han sido propuestos por otros jugadores. Así, durante su turno podrá decidir en cualquier momento si los acepta o no (por ejemplo, antes de lanzar los dados, después de realizar una compra, etc.). --- `$> aceptar trato20` `Se ha aceptado el siguiente trato con Luis: le doy Solar14 y Luis me da Solar10.` --- `$> aceptar trato14` `El trato no puede ser aceptado: Maria no dispone de 300000€.` --- `$> aceptar trato20` `El trato no puede ser aceptado: Solar10 no pertenece a Luis.`
34	**Listar tratos**. Listar los tratos que se le han propuesto a un jugador. Un jugador no podrá ver los tratos propuestos a otros jugadores.

	`` $> tratos { id: trato20 jugadorPropone: Luis, trato: cambiar (Solar14, Solar10) }, { id: trato14 jugadorPropone: Pedro, trato: cambiar (Solar1, 300000) } ```
35	**Eliminar trato**. Un jugador podrá eliminar un trato que ha propuesto y que no ha sido aceptado por otro jugador. Ahora bien, el jugador que no acepta el trato no puede eliminarlo y, además, el no aceptar un trato no supondrá su eliminación. ``` $> eliminar trato20 Se ha eliminado el trato20. ```