Tercera Práctica (P3)
Implementación completa del modelo

Competencias específicas de la primera práctica

- Implementar los métodos con funcionalidad simple especificados mediante lenguaje natural.
- Implementar los métodos con funcionalidad más compleja siguiendo lo indicado en un diagrama de comunicación o en un diagrama de secuencia.

A) Programación y objetivos

Tiempo requerido: Tres sesiones, S1, S2 y S3 (6 horas).

Comienzo: **semana del 23 de octubre excepto para A1, C2 y D1 que empezarán el 2 de noviembre.**

Planificación y objetivos:

Sesión	**Semana**	**Objetivos**
S1	23-27 octubre o 2 noviembre	• Implementar métodos sencillos a partir de su descripción en lenguaje natural.
S2	6-10 noviembre	• Interpretar diagramas de interacción de objetos.
S3	13-17 noviembre	• Implementar una prueba del código realizado y depurarlo.
La práctica se desarrollará tanto en Java como en Ruby en equipos de 2 componentes. El examen es individual.		

Evaluación:

El examen de la práctica 3 será en la semana del 20 al 24 de noviembre (cada grupo en su sesión) para todos los grupos excepto para los grupos A1, C2 y D1 que lo tendrán el 2 de noviembre.

El examen lo realizará **cada grupo en su hora** y aula de prácticas y se resolverá sobre el código que se haya desarrollado para las prácticas 1, 2 y 3 **en el ordenador del aula**. Para ello, cada estudiante deberá acudir a clase con su proyectos Java y Ruby en un pen drive o haberlos dejado en su cuenta de usuario de los ordenadores del aula.

No habrá entrega del código previa al examen, sino que se pedirá que cada estudiante suba a una tarea creada en PRADO a tal efecto los proyectos Java y Ruby con los cambios solicitados durante el examen. .

SESIONES 1 y 2

B) Implementación en Java y Ruby de métodos sencillos

A partir de la especificación proporcionada en lenguaje natural, implementa los siguientes métodos del modelo, tanto en Java como en Ruby. Siempre que puedas debes usar métodos ya implementados en el lenguaje en lugar de implementar los tuyos propios (sobretodo cuando trabajes con colecciones).

1) **En la clase *Tablero***:

- ***esCasillaCarcel (numeroCasilla: int): boolean***. Devuelve true si *numeroCasilla* se corresponde con el número de casilla de la cárcel y false en caso contrario.

- ***obtenerCasillaNumero(numeroCasilla: int): Casilla***. Devuelve la casilla que está en la posición dada en el argumento. Es precondición que *numeroCasilla* sea mayor que 0 y menor que el número máximo de casillas, por lo que no es necesario comprobarlo dentro del método.

- ***obtenerNuevaCasilla(casilla: Casilla, desplazamiento: int): Casilla***. Devuelve la casilla que está *desplazamiento* posiciones después de la posición de la *casilla* dada en el argumento. Ten en cuenta que el tablero es circular, de manera que si se sobrepasa la última casilla, se debe continuar por la primera.

2) En la clase *Qytetet*:

- ***siguienteJugador(): void***. Asigna a *jugadorActual* al siguiente en la lista de jugadores. Si el turno lo tenía el último jugador de la lista, se pasará el turno al primero de la lista.
- ***salidaJugadores(): void*** Posiciona a todos los jugadores en la casilla de salida, con un saldo de 7500 € y asigna de forma aleatoria el jugador actual.
- **propiedadesHipotecadasJugador(hipotecadas : boolean) : Casilla [0..*]** Devuelve las casillas propiedad del *jugadorActual* que estén hipotecadas (cuando el parámetro *hipotecadas* sea true) o que no estén hipotecadas (cuando el parámetro *hipotecadas* sea false).
- **getJugadores(): Jugadores[0..MAX_JUGADORES]** Añade este nuevo método con visibilidad pública para consultar la lista de jugadores.

3) En la clase Dado:

- ***tirar(): int***. Devuelve un número aleatorio entre 1 y 6.

4) En la clase *Casilla*:

- ***soyEdificable(): boolean***. Devuelve cierto sólo si es una casilla de tipo CALLE.

- ***estaHipotecada(): boolean***. Devuelve verdadero o falso según si el título de propiedad indica que está hipotecada o no.

5) **En la clase *Jugador*:**

- ***cuantasCasasHotelesTengo(): int.*** Devuelve el total de casas y hoteles que tiene ese jugador en todas sus propiedades.

- ***devolverCartaLibertad(): Sorpresa***. Devuelve la carta Sorpresa *cartaLibertad*, pues el jugador ya ha hecho uso de ella. Esto implica que el jugador se queda sin esa carta. Presta atención a las referencias nulas, tendrás que utilizar una variable intermedia.

- ***obtenerCapital(): int***. Devuelve el capital del que dispone el jugador, que es igual a su saldo más la suma de los valores de todas sus propiedades. El valor de una propiedad es la suma de su coste más el número de casas y hoteles que haya construidos por el precio de edificación. Si la propiedad estuviese hipotecada, se le restará el valor de la hipoteca base.

- ***tengoCartaLibertad():*** **boolean.** Cierto sólo si *cartaLibertad* no es nula.

- ***esDeMiPropiedad(casilla: Casilla): boolean***. Cierto si el jugador tiene entre sus propiedades el título de propiedad de esa casilla.

- ***puedoVenderPropiedad(casilla: Casilla): boolean***. Cierto sólo si la casilla es de la propiedad de ese jugador (usa para ello el método que acabas de implementar) y no la tiene hipotecada.

- ***eliminarDeMisPropiedades(casilla: Casilla): void***. Elimina el titulo de propiedad de esa casilla de su lista de propiedades.

- ***modificarSaldo(cantidad: int): void***. Añade al saldo la cantidad del argumento. Si el argumento es negativo, el saldo quedará reducido.

- **obtenerPropiedadesHipotecadas(hipotecada: boolean): TituloPropiedad [0..*]** Devuelve los títulos de propiedad del *jugadorActual* que estén hipotecados (cuando el parámetro *hipotecada* sea true) o que no estén hipotecados (cuando el parámetro *hipotecada* sea false).

- **tengoPropiedades: boolean.** Devuelve verdadero cuando el jugador es propietario de algún título de propiedad y falso en caso contrario.

- **tengoSaldo(cantidad: int): boolean.** Devuelve verdadero si el saldo del jugador es superior o igual a *cantidad*.

6) En la clase **TituloPropiedad:**

- ***tengoPropietario(): boolean***. Devuelve true si *propietario* no es nulo.

C) Implementación en Java y Ruby del resto de los métodos del modelo

Implementa en Java y Ruby las operaciones principales del sistema propuesto, partiendo de los diagramas UML de comunicación o secuencia que se encuentran en PRADO. Es importante tener en cuenta que la implementación que se haga de las mismas debe seguir escrupulosamente los diagramas.

Para algunos métodos se aportan dos diagramas para ilustrar la equivalencia entre los diagramas de comunicación y de secuencia, en esos casos se puede seguir cualquiera de los dos indistintamente para realizar la implementación.

Hay dos métodos para los que no se incluye diagrama y deberás plantearte cómo sería su diagrama de interacción e implementarlos. Son:

- **edificarHotel** de la clase Qytetet. Es muy similar a la operación edificarCasa. Se pide realizar un diagrama de comunicación del método e implementarlo.
- **cancelarHipoteca** de la clase Qytetet. Se pide realizar un diagrama de secuencia del método e implementarlo.