Tercera práctica

Implementación de la interacción de objetos

Competencias específicas de la tercera práctica

- Interpretar correctamente diagramas de interacción en UML.
- Traducir los diagramas de interacción de UML a Java y a Ruby.

Programación de la tercera práctica

Tiempo requerido: tres sesiones (seis horas).

Planificación y objetivos:

Sesión	Semana	Objetivos
Primera	Del 10 al 14 de noviembre	• Implementar en Java y Ruby operaciones simples del diagrama de clases visto en la práctica anterior.
Segunda	Del 17 al 21 de noviembre	• Saber interpretar un diagrama de interacción de UML, de secuencia o de comunicación. • Traducir diagramas de interacción a Java y Ruby.
Tercera	Del 24 al 28 de noviembre	• Probar todo el código desarrollado en Java y en Ruby.
• El trabajo se realizará tanto en Java como en Ruby. • La práctica se desarrollará en grupo.		

Entrega:

- Antes de que comience la sesión en la que se realiza el examen.

Examen de la segunda práctica en la semana del 1 al 5 de Diciembre.

- **Lugar:** Aula en la que se desarrolla la correspondiente sesión de prácticas.
- **Día:** El correspondiente a la sesión de cada uno de la semana indicada.
- **Duración**: 20 minutos al comienzo de la sesión.
- **Tipo examen:** Realizar pequeñas modificaciones sobre el propio código.

A) Primera sesión

Tareas para hacer:

Implementar los siguientes métodos en Java y en Ruby

En la clase Napakalaki:

- **initPlayers(names:String[]):void**

  Inicializa el array de jugadores que contiene `Napakalaki`, creando tantos jugadores como elementos haya en `names`, que es el array de `String` que contiene el nombre de los jugadores.

- **nextPlayer():Player**

  Decide qué jugador es el siguiente en jugar. Se pueden dar dos posibilidades para calcular el índice que ocupa dicho jugador en la lista de jugadores:

  - Que sea el primero en jugar, en este caso hay que generar un número aleatorio entre 0 y el numero de jugadores menos 1, este número indicará el índice que ocupa en la lista de jugadores el jugador que comenzará la partida.
  - Que no sea el primero en jugar, en este caso el índice es el del jugador que se encuentra en la siguiente posición al jugador actual. Hay que tener en cuenta que si el jugador actual está en la última posición, el siguiente será el que está en la primera posición.

    Una vez calculado el índice, devuelve el jugador que ocupa esa posición.

- **getCurrentPlayer():Player**

  Devuelve el jugador actual (`currentPlayer`)

- **getCurrentMonster():Monster**

  Devuelve el monstruo en juego (`currentMonster`)

- **nextTurnIsAllowed():boolean**

  Método que comprueba si el jugador activo (`currentPlayer`) cumple con las reglas del juego para poder terminar su turno. Devuelve false si el jugador activo no puede pasar de turno y true en caso contrario, para ello usa el método de `Player: validState()`

- **endOfGame(result:CombatResult):boolean**

  Devuelve `true` si result tiene el valor `WinAndWinGame` del enumerado `CombatResult`, en caso contrario devuelve `false`.

En la clase Player

- **canMakeTreasureVisible(t:Treasure):boolean**

  Comprueba si el tesoro (`t`) se puede pasar de oculto a visible, según las reglas del juego

- **discardNecklaceIfVisible():void**

  Si el jugador tiene equipado el tesoro tipo `NECKLACE,` se lo pasa al `CardDealer` y lo elimina de sus tesoros visibles.

- **computeGoldCoinsValue(t:Treasure[]):int**

  Calcula y devuelve los niveles que puede comprar el jugador con la lista `t` de tesoros.

En la clase BadConsequence

- **substractVisibleTreasure(t:Treasure)**

  Actualiza el mal rollo que tiene que cumplir el jugador, en función del tipo de tesoro de `t` y del tipo de mal rollo que tenga que cumplir el jugador.

- **substractHiddenTreasure(t:Treasure)**

  Igual que el anterior, pero para los ocultos.

En la clase CardDealer **(puede que algunos ya los tengas implementados):**

- **nextTreasure():Treasure**

  Devuelve el siguiente tesoro que hay en el mazo de tesoros (`unusedTreasures`) y lo elimina de él. Si el mazo está vacío, pasa el mazo de descartes (`usedTreasures`) al mazo de tesoros y lo baraja, dejando el mazo de descartes vacío.

- **nextMonster():Monster**

  Igual que la anterior pero con el mazo de monstruos.