Introducción al Software - EXAMEN DE PRÁCTICAS SEPTIEMBRE 2017.

“Clash Royale” es un juego de cartas en tiempo real para móviles que seguramente muchos de vosotros conoceréis.

En el juego dos jugadores se enfrentan entre sí en una especie de batalla. Cada jugador dispone de un conjunto de cartas que representan criaturas y personajes, cada una con habilidades y estadísticas diferentes, que cada jugador podrá invocar en el campo de batalla a cambio de una cantidad de elixir. El objetivo no es otro que destruir las 3 torres de tu contrincante. Aquel que consiga destruir más torres al finalizar de los 3 minutos de partida, será el vencedor.

Pretendemos desarrollar nuestra propia versión del juego, mucho más sencilla y modesta, que llamaremos “Clash Unican” basándonos en algunas de las ideas de su hermano mayor.

Para empezar, nos centraremos en la parte central del juego: las cartas. La **clase Carta** contiene la información de cada una de las cartas del juego se ajusta al siguiente diagrama de clase:

Carta
- String nombre - int elixir - Tipo tipo - Calidad calidad - int arena
+ Carta(String nombre, int elixir, Tipo tipo, Calidad calidad, int arena) + String nombre() + int elixir() + Tipo tipo() + Calidad calidad() + int arena() + String datosCarta()

• 5 atributos: nombre de la carta, la cantidad de elixir que cuesta usarla, el tipo de carta que es (Tropa, Hechizo o Estructura), la calidad de la carta (Común, Especial, Épica o Legendaria) y la arena en la que está disponible.

• constructor que recibe como parámetros los datos de la carta y los guarda en los atributos correspondientes.

• un método observador para cada atributo

• un método datosCarta que devuelve una cadena de texto con todos los datos de la carta.

El tipo y calidad están definidos mediante dos tipos enumerados definidos dentro de la clase Carta de esta forma:

```
public enum Tipo {TROPA, HECHIZO, ESTRUCTURA}

public enum Calidad {COMUN, ESPECIAL, EPICA, LEGENDARIA}
```

Para almacenar el set completo de cartas del juego disponemos de la clase SetCartas, que almacena en su interior un único objeto con el que trabajan los métodos, que son estáticos. El constructor es privado, por lo que no pueden crearse otros objetos. Esta clase ofrece las siguientes operaciones:

SetCartas
…
+static Carta buscaPorNombre(String nombreCarta) +static Carta buscaPorIndice(int indiceCarta) +static ArrayList<Carta> buscaPorArena(int arenaCarta) +static int numCartas()

• buscaPorNombre (): método estático que nos devuelve una carta concreta, dado su nombre. En caso de que no exista la carta solicitada, devuelve null.

• buscaPorIndice (): método estático que nos devuelve una carta concreta, dada su posición en el set de cartas. En caso de que no exista la carta solicitada, devuelve null.

• buscaPorArena(): método estático que nos devuelve un ArrayList de cartas con todas aquellas cartas de una determinada arena, que se pasa como parámetro. En caso de que no existan cartas de esa arena, se devuelve un ArrayList vacío.

• numCartas(): método estático que nos indica el número total de cartas disponibles en el juego.

Nota: Estas dos clases se nos facilitan ya desarrolladas.

Se nos pide proseguir con el desarrollo de la clase Jugador, parcialmente realizada, que se encarga de gestionar la información relativa a cada jugador del juego. Su diagrama de clase es el siguiente:

Jugador
- String jugador - Carta[] mazo - ArrayList<Carta> cartasDisponibles - int trofeos - int maxTrofeos - int arena
+ Jugador(String jugador) + boolean anyadeCartaMazo(String carta) + int sustituyeCartaMazo(String cartaOut, String cartaIn) + double elixirMedioMazo() + void simuladorPartida(int resultado) + String fichaJugador()

La clase consta de 6 atributos:

- nombre de el jugador
- el mazo con el que juega el jugador, que es un array de 8 cartas

- el conjunto de cartas disponibles para ese jugador, que dependerán de los avances del jugador
- el número de trofeos que tiene el jugador actualmente
- el máximo número de trofeos conseguidos por el jugador, es decir, su record personal.
- la arena más alta a la que ha llegado el jugador. Es la que define el conjunto de cartas disponibles, que serán todas aquellas de arena igual o inferior a la arena más alta conseguida por el jugador.

Asimismo, tendrá los siguientes métodos:

Constructor: Crea un nuevo jugador con el nombre que se pasa como parámetro. Se genera un mazo de 8 cartas inicialmente vacío (es decir, con todos sus elementos a valor null), se inicializan el número de trofeos y el número máximo de trofeos a 0, se define la arena actual como 0 y las cartas disponibles para el jugador, que son las de la arena 0 en el set de cartas.

reseteaMazo(): Este método carga en el mazo del jugador las 8 primeras cartas del set de cartas disponibles para el jugador. Si por algún motivo en el set de cartas disponibles no hubiera cartas suficientes, el mazo se deja tal y cómo estaba y se devuelve falso. Si la carga se realiza correctamente, devuelve verdadero.

sustituyeCartaMazo(): método que sustituye una carta del mazo, identificada por su nombre que se recibe en el parámetro cartaOut, por otra, cuyo nombre se recibe en el parámetro cartaIn, siempre que sea posible. Los casos en los que la sustitución no es posible son:

  - Que la carta que se quiere quitar no esté previamente en el mazo, en cuyo caso se retorna el código de error -1.
  - Que la carta que se quiere incluir se encuentre ya en el mazo, en cuyo caso se retorna el código de error -2.
  - Que la carta que se quiere añadir no esté disponible en el conjunto de cartas disponibles para el jugador, en cuyo caso se retorna el código de error -3.

Los casos de error se deben comprobar en el orden especificado. Si no se produce error, se sustituye en el mazo una carta por otra y se retorna el valor 0.

elixirMedioMazo(): método que retorna el elixir que cuesta lanzar cada carta que compone el mazo y devuelve la media de todos los costes. En caso de que el mazo esté incompleto, retorna -1.

simuladorPartida(): Este método simulará una partida y permitirá al jugador ir ganando trofeos. Este simulador recibe como parámetro un valor entero que identifica el resultado de la partida, actualizando el valor de trofeos según la tabla siguiente:

Valor resultado	Significado	Trofeos
1	VICTORIA	+30
0	EMPATE	0
-1	DERROTA	-26

En caso de que el valor recibido no sea ninguno de los de la tabla (-1,0 o 1), entonces el método no hace nada.

Asimismo, deberá actualizar el número máximo de trofeos si es necesario (siempre que el número de trofeos actual suponga un nuevo record del jugador).

Por último, debe actualizar la lista de cartas disponibles del jugador siempre que este suba a una arena superior. Para subir de arena, el jugador debe conseguir cierto número de trofeos, tal y como se muestra en la tabla:

Arena	Trofeos necesarios
1	50
2	400
3	800
4	1100

En el momento en que se consiguen los trofeos necesarios para pasar a una nueva arena, el conjunto de cartas disponibles aumenta, añadiendo a las que ya había las cartas de la colección completa de la arena a la que subimos. Además se anota en arena la nueva arena alcanzada.

fichaJugador(): Este método retorna un String con toda la información del jugador: su nombre, trofeos, record de trofeos y arena máxima alcanzada. Tambien devuelve la información de las cartas que componen el mazo del jugador: su nombre, elixir, arena, tipo y calidad. Este método se da ya resuelto.

Valoración

Para la valoración, al tratarse de un examen de prácticas, es **imprescindible que el código entregado compile correctamente y que los métodos realicen la funcionalidad requerida.**

Para poder probar la funcionalidad se facilita **una clase con un método principal**, capaz de probar los métodos.

- Método Constructor: 1 punto
- Método reseteaMazo: 1,5 puntos
- Método sustituyeCartaMazo: 3,5 puntos
- Método elixirMedioMazo: 1,5 puntos
- Método simuladorPartida: 2,5 puntos