PROGRAMACIÓN

U5

Antonio

Eslava Hernández

1ºDAW

1. Contenido

1. Enunciado

Vamos a poner en práctica el uso de listas para implementar un juego de Blackjack, para ello vamos a implementar una serie de clases.

Clase Carta

Contiene los datos correspondientes a una Carta. La clase carta definirá un tipo enumerado para los palos de las cartas. Lo puedes definir dentro de la clase de la siguiente manera:

```
enum Palo {
TREBOL, DIAMANTES, CORAZONES, PICAS
};
```

Una carta tendrá sólo dos tipos de datos:

- Número de la carta. Es un numero del 1 al 12
- Palo. De tipo Palo (el tipo enumerado)

Define los siguientes métodos:

- Constructor parametrizado al que le pases el numero y el palo, y asigne los valores. El número deberá estar entre 1 y 13.
- Métodos getters de los atributos de las cartas. No crear lo setters
- Método getValor que devuelve el valor que tiene cada carta en el BlackJack. Los valores serán: 11 para el AS (1), 10 para las figuras (J,Q,K o 11, 12 y 13) y el valor correspondiente para el resto de las cartas (2, 3, 4, 5, 6, 7, 8, 9)
- Método mostrarNumero sin parámetros que devuelve un String con la representación de cada número. Para el 1 habrá que mostrar “AS”, para la 11 habrá que mostrar “J”, para la 12 habrá que mostrar “Q”, para la 13 “K” y para el resto su correspondiente valor.
- Sobreescribe el método toString para que muestre la carta de la siguiente manera [numero – Palo]. Por ejemplo, [3 – CORAZONES]

Clase Mazo

Define el mazo de Cartas que se va a utilizar en el juego.

- Cartas. Contendrá las 52 cartas de la baraja en una estructura de tipo ArrayList<Carta>

Deberás generar los siguientes métodos:

- Constructor sin parámetros. Genera un mazo de cartas con todas las cartas. Para ello, deberemos recorrer todos los Palos y todo los posibles números y generaremos todas las cartas y las añadiremos a la lista. Para recorrer los palos puedes utilizar:

```
for (Palo p: Palo.values())
```

- Método barajar() sin parámetros. Se encargará de barajar el mazo de cartas. Lo único que hace es utilizar el método shuffle de la clase Collecction:

```
Collections.shuffle(cartas);
```

- Sobreescribe el método toString para mostrar todo el mazo de cartas recorriendo la lista
- solicitarCarta. Método sin parámetros que devuelve la primera carta del mazo eliminándola de la lista.

Clase Mano

Representa la mano que tiene un usuario y por tanto también es una lista de cartas, por lo que será una clase que contenga los mismos datos que Mazo y por tanto la vamos a definir como una subclase de Mazo.

- La clase no define ningún atributo propio, sólo métodos nuevos, que son:
- Constructor de la clase. Define una lista de caratas vacía, no invoca al método de la superclase
- Método valorMano sin parámetros. Calcula cual es valor que tenemos en la mano de cartas. Para hacerlo recorreremos la lista de cartas sumando los puntos de cada una.
- Método finDeJuego que devuelve un boolean, indicando true si la mano tiene más de 21 puntos, significando que el jugador se ha pasado.
- Sobreescribe el método toString para que invoque al método de la superclase, pero antes muestre la puntuación de la mano
- Método pedirCarta(Mazo m) al que se le pasa un mazo de cartas, deberá solicitar una carta al mazo e incluirla en la mano.

Clase Juego

- Implementa el método main para crear el juego. Deberá crear un mazo de cartas y barajarlo.
- A continuación creará un mazo de carta para el jugador y le preguntará si quiere una carta. Si la quiere, se le solicitará un carta al mazo y acto seguido se mostrará la mano.
- Si ha llegado al fin del juego (se ha pasado de 21) se sale diciendo que ha perdido, si no le pedirá al usuario si quiere otra carta. Si la quiere se pedirá otra y se repetirá el proceso.
- Si no la quiere se saldrá del juego indicando la puntuación con la que se ha plantado.

Los resultados finales de cada jugador se registrarán en un xml.

2. Explicación.

i. Clase Carta.

Este código Java define una clase llamada Carta, que representa una carta de una baraja de cartas. Veamos los componentes principales del código:

- **Enumeración Palo**: En el código, se define una enumeración llamada Palo, que enumera los posibles palos de una carta de una baraja estándar. Esto incluye Trebol, Diamantes, Corazones y Picas. Usar una enumeración para los palos proporciona una forma conveniente y segura de representar estos valores, ya que evita errores de escritura y proporciona un conjunto limitado de opciones.
- **Constructor**: El constructor de la clase Carta se utiliza para crear instancias de cartas con un número y un palo dados. Verifica si el número de la carta está dentro del rango válido (entre 1 y 13) y muestra un mensaje de advertencia si no lo está. Esto asegura que las

cartas creadas estén dentro de los límites esperados y ayudan a prevenir errores en la aplicación.

- **Métodos de acceso:** La clase proporciona métodos para obtener el número de la carta y el palo de la carta. Estos métodos son útiles para acceder a la información de una carta desde otras partes de la aplicación de manera segura, ya que encapsulan el acceso a las variables de instancia de la clase.
- **Método getValor**: Este método devuelve el valor de la carta en un juego de Blackjack. Por ejemplo, en Blackjack, las cartas numéricas tienen el mismo valor que su número, mientras que las figuras (J, Q, K) tienen un valor de 10 y el As puede tener un valor de 11 o 1 según convenga al jugador.
- **Método mostrarNumero**: Este método devuelve una representación en forma de cadena del número de la carta. Convierte los valores numéricos especiales (1 para AS, 11 para J, 12 para Q y 13 para K) en sus correspondientes nombres de cartas. Esto es útil para mostrar la carta de una manera más comprensible para los usuarios, especialmente en interfaces de usuario de juegos de cartas.
- **Método toString**: Este método proporciona una representación de cadena de la carta, incluyendo el número y el palo de la carta. Sobrescribe el método toString heredado de la clase Object, lo que permite imprimir fácilmente una carta en forma legible para humanos, facilitando la depuración y el desarrollo de la aplicación.

ii. Clase Mazo.

- **Declaración de la clase Mazo**: La clase Mazo tiene una variable de instancia Cartas, que es un ArrayList que contiene objetos de tipo Carta.
- **Constructor Mazo():** El constructor inicializa el mazo de cartas. Utiliza dos bucles anidados para iterar sobre cada uno de los palos (definidos en la clase Carta) y cada número de carta del 1 al 13. Para cada combinación de palo y número de carta, crea un objeto Carta y lo agrega al mazo.
- **Método barajar():** Este método baraja las cartas en el mazo utilizando el método shuffle() de la clase Collections, que reorganiza aleatoriamente los elementos en la lista.
- Método toString(): Este método proporciona una representación de cadena del mazo. Itera sobre todas las cartas en el mazo y las concatena en una cadena. Además, formatea la salida para que cada par de cartas esté separado por un tabulador (\t) y cada carta esté en una nueva línea.
- **Método solicitarCarta():** Este método permite sacar una carta del mazo. Verifica si el mazo está vacío y, si no lo está, selecciona aleatoriamente una carta del mazo, la elimina y la devuelve. Si el mazo está vacío, imprime un mensaje de advertencia y devuelve null.

iii. Clase Mano.

La clase Mano extiende la clase Mazo, lo que significa que hereda todas las características y métodos de la clase Mazo. Además, tiene una anotación @XmlRootElement que especifica que esta clase puede ser mapeada a un elemento raíz en un documento XML.

- **Atributo carta:** Esta clase tiene un atributo carta que es un ArrayList de objetos de tipo Carta. Esta lista contiene las cartas que componen la mano del jugador.
- **Constructor Mano():** El constructor inicializa la lista de cartas de la mano como un nuevo ArrayList.

- **Método valorMano():** Este método calcula el valor total de la mano sumando los valores de todas las cartas en la mano utilizando el método getValor() de la clase Carta.
- **Método finDeJuego():** Este método verifica si la suma de los valores de las cartas en la mano supera 21, lo que significa que la mano está "quemada" en un juego de Blackjack.
- **Método toString():** Este método proporciona una representación de cadena de la mano, que incluye la puntuación total de la mano y una representación de cadena de las cartas en la mano, utilizando el método toString() de la clase Mazo.
- **Método pedirCarta():** Este método permite al jugador pedir una carta de un mazo dado. Agrega la carta solicitada a la mano del jugador.

iv. Clase Main.

El código es una implementación de un juego simple de Blackjack en Java, que permite a un jugador pedir cartas del mazo hasta que decida plantarse o sobrepasar un total de 21. A continuación, se detallan los principales componentes y funcionalidades del código:

- **Clase Main:** Esta es la clase principal del programa, que contiene el método main donde comienza la ejecución del juego.
- **Inicialización del mazo y barajado**: Se crea un objeto de la clase Mazo, se barajan las cartas y se prepara el mazo para el juego.
- **Bucle principal del juego:** Se utiliza un bucle while para permitir al jugador pedir cartas del mazo mientras no se pase de 21 puntos.
- **Interacción con el jugador**: Se muestra la mano actual del jugador y se le pregunta si desea pedir otra carta. La entrada del jugador se captura mediante un objeto Scanner.
- **Manejo de la lógica del juego:** Dependiendo de la decisión del jugador, se agregan cartas a la mano del jugador o se finaliza el juego.
- **Generación de un archivo XML**: Después de cada jugada, se utiliza JAXB (Java Architecture for XML Binding) para convertir la mano del jugador en un documento XML y se guarda en un archivo llamado salida.xml.
- **Finalización del juego**: El juego termina cuando el jugador decide plantarse o cuando su mano supera los 21 puntos. Se muestra un mensaje indicando si el jugador ganó o perdió.

3. Conclusion.

En conclusión, el código implementa un juego simple de Blackjack en Java, que permite a un jugador interactuar con un mazo de cartas y tomar decisiones estratégicas durante el juego. A través de la interacción con el jugador, el programa simula las acciones típicas de un juego de Blackjack, como pedir cartas adicionales o plantarse, mientras se asegura de que el jugador no supere los 21 puntos. Además, utiliza JAXB para convertir la información sobre la mano del jugador en un documento XML y guardarlo en un archivo.

Esta implementación usa varios conceptos y características importantes de Java, el manejo de excepciones, la entrada de usuario mediante Scanner, y el uso de anotaciones JAXB para mapear objetos Java a documentos XML. Además, hace uso de estructuras de datos como ArrayList para representar y manipular colecciones de objetos, en este caso, cartas de un mazo.