P4

PRÁCTICA EVALUABLE

Desarrollando juegos con una baraja de cartas

OBJETIVOS DIDÁCTICOS

- Escribir un videojuego simulando juegos de cartas.
- Consolidar los conceptos estudiados en las unidades 6, 7 y 8.
- Escribir programas utilizando estructuras dinámicas y conceptos avanzados de programación orientada a objetos (POO).

1. CONTENIDO

A continuación, un listado de los puntos principales de este documento que deberéis leer con detalle para desarrollar con éxito esta práctica:

- Enunciado de la práctica
- Estructura del proyecto
- Implementación
- Criterios de calificación de la práctica.

2. ENUNCIADO DE LA PRÁCTICA

Vamos a realizar un proyecto donde se pueda jugar a diferentes juegos de cartas con la baraja española. Para simplificar la práctica implementaremos solamente el juego del 7 y medio, aunque preparemos las clases de Tute y Brisca para futuras ampliaciones.

Para conocer estos juegos os dejo los siguientes enlaces:

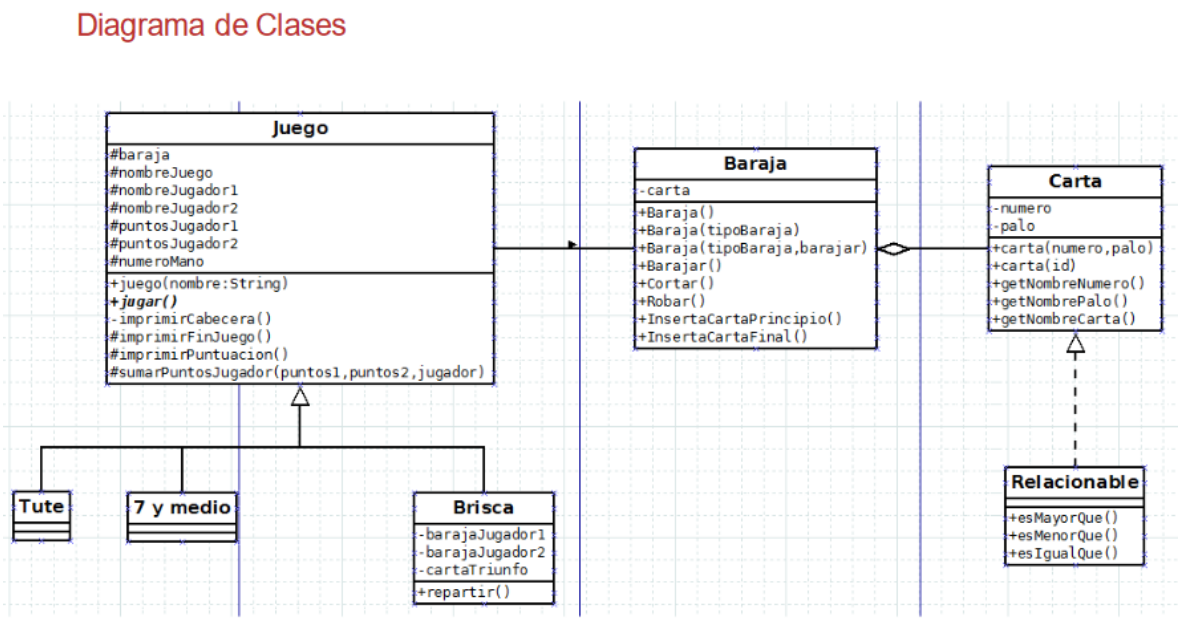
- Siete y Medio: ver descripción y reglas en Wikipedia
- Tute: ver descripción y reglas en Wikipedia
- Brisca: ver descripción y reglas en Wikipedia

3. ESTRUCTURA DEL PROYECTO

3.1. DIAGRAMA DE CLASES DEL PROYECTO

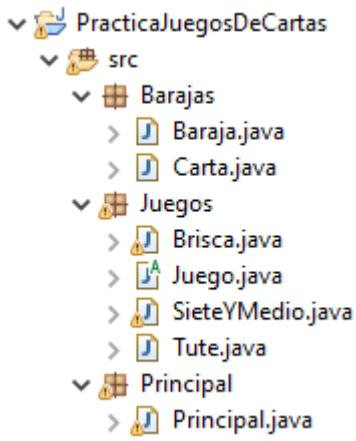
Lo primero que deberíamos hacer antes de ponernos a escribir código es hacer un esquema de nuestro proyecto. Esto es, plasmar los componentes (clases, interfaces, paquetes, etc.) que necesitamos y sus relaciones.

No obstante, en esta práctica ya se os ofrece este esquema en formato UML con un diagrama de clases:



3.2. ESTRUCTURA DE LAS CLASES EN EL IDE

A continuación, en un entorno de desarrollo (en nuestro caso Eclipse) crearemos la estructura de clases y paquetes correspondiente nuestro diseño anterior. Posteriormente iremos implementando cada una de las clases y otros componentes (interfaces, Enum, etc.) de la estructura creada. El explorador de paquetes (*Package Explorer*) de nuestro proyecto en Eclipse debería quedar así:



4. IMPLEMENTACIÓN DEL PROYECTO

En este apartado vamos a ver una descripción de la implementación de cada una de las clases del proyecto.

4.1. CLASE CARTA

Atributos (privados):

- **numero**: un entero (1..10) que nos representa el número de la carta, siendo el 8 la sota, el 9 el caballo y el 10 el rey.
- **palo**: un entero (0, 1, 2, 3) que nos representa el palo de la carta (oros, copas, espadas y bastos).
    - Se valorará que: En vez de que palo sea un número entero implementarlo como un tipo Enum, donde se indican los palos (OROS, COPAS, BASTOS, ESPADAS).

Constructores:

- *Carta(numero, palo)*: nos crea una carta pasándole el palo y el número.
- *Carta(id)*: a la que pasamos un número entre 1 y 40 que representa la carta, siendo 1 el as de oros y 40 el rey de bastos.

Propiedades (Métodos, getters y/o setters):

- **Numero**: de solo lectura, que nos devuelve el número.
- **Palo**: de solo lectura, que nos devuelve el palo.
- **NombreNumero**: de solo lectura, que nos devuelve número de la carta como un string y con letras (1=as, 2=dos, …, 10=rey).
- **NombrePalo**: de solo lectura, que nos devuelve el palo de la carta, en letras (0 = oros, etc.).
- **NombreCarta**: de solo lectura, nos devuelve el nombre completo de la carta (ej: as de oros).
- **ValorTute**: de solo lectura, nos devuelve el valor de la carta en el juego del tute (1 = 11, 3 = 10, sota = 2, caballo = 3, rey = 4).
- **ValorMus**: de solo lectura, nos devuelve el valor de la carta en el juego del mus (1, 2 = 1; 3, sota, caballo y rey = 10, el resto su valor).
- **Valor7ymedio**: de solo lectura nos devuelve el valor de la carta en el juego del Siete Y Medio (figuras: 0.5, el resto su valor). Tendrá que devolver un decimal para poder devolver el medio punto.
- **equals()**: sobreescribir la clase equals() de la clase Object. Dos cartas son iguales si tiene el mismo número y palo.

4.2. INTERFAZ RELACIONABLE

Crear la interfaz Relacionable que definirá los siguientes métodos:

- esMayorQue(Relacionable r)
- esMenorQue(Relacionable r)
- esIgualQue(Relacionable r)

Esta interfaz la debe implementar la clase Carta, de manera una carta se relaciona con otra a través de su número (y también palo en caso de esIgualQue).

4.3. CLASE BARAJA

Atributos (privados):

- **listaCartas**: será una lista de cartas. Habrá que inicializarla en los constructores.

Constructores:

- *Baraja()*: nos crea una baraja vacía.
- *Baraja(int tipoBaraja)*: nos crea una baraja del tipo que le digamos. Por ahora, solo tendremos dos tipos de barajas. La baraja normal de 40 cartas (opción 1) y una baraja doble de 80 cartas (opción 2).
- *Baraja(int tipoBaraja, bool barajar)*: igual que la anterior pero con un booleano que nos dirá si debemos barajar la baraja después de crearla o no.

Propiedades (Métodos, getters y/o setters):

- *numeroCartas*: de solo lectura, nos dice cuántas cartas quedan en la baraja.
- *vacia*: de solo lectura, que será un booleano que nos dice si la baraja está vacía o aún quedan cartas.
- *barajar():* mezcla aleatoriamente las cartas de la baraja. Hay varias formas de hacerlo, yo os propongo crear otra lista de cartas; mientras queden cartas en nuestra baraja, elegimos una al azar (con un Random), la sacaremos y la meteremos en la siguiente baraja. Al terminar, copiamos todas las cartas a nuestra baraja y listos.
- *robar():* roba una carta. Nos devuelve un objeto de tipo carta que se corresponde al primer elemento de la lista que eliminaremos de la misma.
- *cortar(int posicion):* corta la baraja. Consisten en pasar tantas cartas como nos digan desde la primera posición hasta la última. Se recomienda usar para ello los métodos auxiliares que se debes implementar a continuación.
- *insertaCartaFinal(int idCarta):* meteremos una carta nueva al final de la baraja correspondiente al id.
- *insertaCartaPrincipio(int idCarta):* meteremos una carta nueva al principio de la baraja correspondiente al id.
- *insertaCartaFinal(Carta c):* igual que la anterior, pero le pasamos un objeto de tipo carta.
- *insertaCartaPrincipo(Carta c):* igual que la anterior, pero le pasamos un objeto de tipo carta.

4.4. CLASE JUEGO

Esta clase no podrá instanciarse.

Atributos (privados):

- **baraja**: que hace referencia a la clase Baraja.
- **nombreJuego**: cadena de texto que almacena el nombre del juego concreto.
- **nombreJugador1**: cadena de texto que almacena el nombre del jugador 1.
- **nombreJugador2**: cadena de texto que almacena el nombre del jugador 2.

- **puntosJugador1**: un número con decimales que guarda los puntos del jugador 1.
- **puntosJugador2**: un número con decimales que guarda los puntos del jugador 1.
- **numeroMano**: un entero que guarda la mano correspondiente de juego. Se usa en los juegos que requieren saber este dato.

Constructores:

- *Juego(nombreJuego)*: crea el juego en función del nombre del juego (cadena de texto).

Propiedades (Métodos, getters y/o setters):

- *jugar()*: método abstracto que implementará, en la clase del juego correspondiente, el algoritmo de juego.
- *imprimirCabecera()*: método privado que imprimirá el nombre del juego. Se puede usar para mostrarse antes de empezar el juego correspondiente.
- *imprimirFinJuego(nombreJugador, ganapierde)*: imprime el nombre del jugador (cadena de texto) indicando si ha ganado o perdido. El parámetro ganaPierde es una cadena de texto que debería ser "gana" o "pierde".
- *imprimirGanador()*: imprimir quién ha ganado la partida, esto es, que un jugador tiene más puntos que otro. Consejo: se puede usar el método imprimirFinJuego dentro de este método.
- *imprimirPuntuacion()*: imprime los puntos conseguidos por cada jugador.
- *sumarPuntosJugador(puntosCarta1, puntosCarta2, int numJugador)*: acumula los puntos de cada jugador (numJugador = 1 si es el jugador 1, numJugador = 2 si es el jugador 2). Consejo: si solo queremos acumular puntos carta por carta, poner a cero el valor del parámetro puntosCarta2 al llamar al método.

4.5. CLASE PRINCIPAL

Implementar el método main() para la clase Principal.java. Este servirá para lanzar cada unos de los juegos creados a partir de la selección del usuario en el menú principal.

A continuación, se muestra el menú principal que ve el usuario al ejecutarse el programa:

```
************************
 JUEGOS DE CARTAS
************************

1. 7 y medio
2. La Brisca
3. Tute
0. Salir

Opción:
```

Aunque no hayamos implementado el juego de La Brisca ni el Tute, debemos mostrar este menú al usuario. Por ejemplo, podemos aprovechar el método jugar() de cada una de estas clases para mostrar el mensaje "En desarrollo…".

4.6. CLASE SIETEYMEDIO

Descripción de la clase SiesteYMedio:

- Es una subclase de la clase Juego.
- Debe implementar el método abstracto *jugar()*: consistirá en implementar la dinámica del juego del Siete y Medio. Debemos mostrar dos modos de juego al usuario:
    - **Player 1**: la CPU le reparte cartas hasta que el jugador 1 o bien decida plantarse, llega al 7.5 o se pasa. En este modo el jugador gana si saca entre 6 y 7.5, en otro caso pierde.
    - **Player1 VS Player2**: en cada turno la CPU reparte cartas a cada jugador y le pregunta si quiere continuar o plantarse. Gana el que o bien obtenga como puntuación total 7.5 o el que más se acerque a este valor.

      Por ejemplo:

```
**********************************
            7 y medio
**********************************

Player 1: Seis de oros => 6.0 puntos

Player 1: 6.0 puntos
Player 2: 0.0 puntos

Player 1: ¿Continuar (c) o Plantarse (p)? p

Player 2: Sota de copas => 0.5 puntos

Player 1: 6.0 puntos
Player 2: 0.5 puntos

Player 2 ¿Continuar (c) o Plantarse (p)? c

Player 2: Siete de copas => 6 puntos

Player 2: ¿Continuar (c) o Plantarse (p)? p

Player 1: 6.0 puntos
Player 2: 6.5 puntos

**********************************
Player 2 GANA a 7 y medio
**********************************
```

- Como parte avanzada haced que el ordenador juegue también (la implementación de este algoritmo de inteligencia artificial no es para nada compleja). En este caso el modo de juego es el siguiente:
    - **Player1 VS CPU:** la CPU (la banca) reparte cartas hasta que el jugador 1 o bien decida plantarse, llega al 7.5 o se pasa. Luego es el turno de la banca, donde la CPU se reparte cartas a si misma y toma la decisión de plantarse o seguir. Gana el que obtenga una puntuación total de 7.5 o el que más se acerque a este valor. También se puede dar el caso de empate.

Un ejemplo de ejecución del modo Player1 VS CPU sería:

```
**********************************
            7 y medio
**********************************

Player 1: Seis de copas => 6.0 puntos

Player 1: 6.0 puntos
CPU: 0.0 puntos

Player 1: ¿Continuar (c) o Plantarse (p)? p

CPU: Cinco de copas => 5.0 puntos

Player 1: 6.0 puntos
CPU: 5.0 puntos

CPU: Seis de oros => 6.0 puntos

Player 1: 6.0 puntos
CPU: 11.0 puntos

**********************************
Player 1 GANA a 7 y medio
**********************************
```

4.7. CLASE BRISCA

Implementación opcional.

4.8. CLASE TUTE

Implementación opcional.

5. FUTURAS AMPLIACIONES

En esta práctica habrás implementado una estructura de clases que será la base de múltiples juegos de cartas y otras características:

- ✓ Aplicación multilenguaje.
- ✓ Otros juegos con Inteligencia Artificial.
- ✓ Cartas de póker.
- ✓ Trucos de cartas
- ✓ etc.

6. DESARROLLO POR ETAPAS Y CRITERIOS DE CALIFICACIÓN

A continuación, una referencia a la forma de valorar esta práctica:

1. **Estructura del programa (5 puntos):** Se implementan clases (no opcionales) de la estructura del proyecto, así como los métodos de cada clase de forma adecuada. No se puede jugar al Siete y Medio (bien sea por no haber sido implementado o por otros motivos).

2. **Jugar al Siete Y Medio Player 1 (1 puntos):** Teniendo la estructura del programa adecuada, se implementa el juego de Siete y Medio. Se puede jugar en modo Player 1.

3. **Jugar al Siete Y Medio Player 1 VS Player 2 (2 puntos):** Teniendo el modo Player 1 también se da la opción de jugar a Player 1 VS Player 2.

4. **Jugar al Siete Y Medio Player 1 VS CPU (1 puntos):** Se añade la inteligencia artificial para jugar en modo Jugador VS CPU.

5. **Claridad, estilo y eficiencia en el código (1 punto):** mencionados en el siguiente punto.

En el desarrollo del programa se valorará:

- Correcto funcionamiento del juego.
- Formateo de la interacción con el usuario.
- Correcto uso de métodos y evitar repetir código (en la medida de lo posible).
- Uso de mecanismos avanzados como los Enumerados.
- Aplicación de polimorfismo.
- Código ordenado y bien estructurado.
- Modificadores de acceso apropiados.
- Nombres de variables apropiados y auto explicativos.
- Comentarios útiles y breves que ayudan a entender el código.

Tenéis libertad para añadir a las clases métodos que penséis que os puedan ser de utilidad. También se da libertad en el formateo de salida por consola de la dinámica del juego.

7. CONSEJOS Y CONSIDERACIONES

Antes de empezar a programar lee bien el enunciado, subraya lo importante, hazte notas, escribe un borrador de las posibles funciones, etc. Intenta entender el problema y cómo realizarlo antes de empezar a escribir código.

No intentes hacerlo todo de golpe. Una estrategia clave en programación es dividir un problema grande en varios problemas pequeños y luego resolverlos uno a uno (es decir, programa y prueba las funciones una a una).

8. RECONOCIMIENTO DE AUTORÍA

Autor de este documento: José Ramón Simó

Esta práctica es una adaptación de la práctica creada por: José Chamorro Molina