EMPEZAMOS A USAR IDE

Usaremos Visual Studio Code, con sus cosas buenas y malas pero es en definitiva el que vamos a usar. Será el que usemos en principio para los exámenes, aunque esto no quita que adicionalmente para alguna cosa utilicemos otro IDE distinto(por ejemplo intelliJ para kotlin).

En la carpeta compartida de videos tienes un video para la instalación y configuración mínima, otro para hacer un primer proyecto de prueba "hola mundo" y otro para montar el proyecto al que alude el siguiente ejercicio.

EJERCICIO: Modificar el proyecto SieteYMedia para que siga la arquitectura de dos capas.

concepto de paquete

Quizá ya estudiaste este concepto en el módulo de Contornos. No corresponde a este boletín. Simplemente, como en el siguiente ejercicio se usan, basta con saber que igual que cuando tenemos muchos ficheros queremos organizarlos en carpetas, si tenemos muchas clases las queremos organizar en paquetes. Para indicar a qué paquete pertenece una clase se utiliza como primera instrucción del fichero .java correspondiente la sentencia *package*. En el siguiente ejercicio podrás comprobar la existencia de esta sentencia.

juego siete y media

Las siguientes clases implementan una versión muy reducida del famoso juego de cartas *siete y media*. En esta versión, hay dos jugadores, el usuario y la banca. La banca es el ordenador (nuestro programa). Respecto al juego original, en esta versión no hay apuestas y sólo se juega una mano. El funcionamiento concreto se aprecia fácilmente ejecutando el programa que incluye al principio una mini explicación.

```
//Palo.java
package recursos;
public enum Palo{
   BASTOS,COPAS,ESPADAS,OROS;
}
```

```
//Carta.java
package recursos;
public class Carta {

   private Palo palo;
   private int numero;

   public Carta(Palo palo, int numero) {
      this.palo = palo;
      this.numero = numero;
   }

   public String getPalo() {
      return palo.toString();
   }

   public int getNumero() {
      return numero;
   }
```

```
    @Override
    public String toString() {
        return "(" + palo + ", " + numero + ')';
    }

}
```

```
//Baraja.java
package recursos;

import java.util.Random;

public class Baraja {

    //baraja española de 40 cartas. No hay 8 y 9.
    private final int NUM_CARTAS = 40;
    private Carta[] cartas = new Carta[NUM_CARTAS];
    int primeraMazo;//el indice de la primera carta sin dar. A las cartas sin dar le llamo mazo.

   public Baraja() {
        //crea una baraja ordenada por palos y números
        int ultimaCarta=0;
        for (Palo p:Palo.values()) {
            for (int j = 0; j < 12; j++) {
                if((j==7 || j==8)){
                    continue;
                }
                cartas[ultimaCarta] = new Carta(p,j+1);
                ultimaCarta++;
            }
        }
    }


    public void barajar() {
        //baraja el mazo, es decir, la cartas sin dar
        Random r = new Random();
        for (int i = primeraMazo; i < cartas.length; i++) {
            int posicionAzar = r.nextInt(cartas.length-primeraMazo)+primeraMazo;
            Carta temp = cartas[i];
            cartas[i] = cartas[posicionAzar];
            cartas[posicionAzar] = temp;
        }

    }




    public Carta[] darCartas(int numCartasDar) {
        //coge cartas del mazo para dar
        //si no hay suficientes cartas o el mazo está vacío se devuelve array vacio
        Carta[] cartasParaDar;

        int cartasEnMazo = cartas.length - primeraMazo;

        if (cartasEnMazo < numCartasDar) {
            cartasParaDar = new Carta[0];
        } else {
            cartasParaDar = new Carta[numCartasDar];
            int i = 0;
            for (; i < cartasParaDar.length; i++) {
                cartasParaDar[i] = cartas[i + primeraMazo];
            }
            primeraMazo = i + primeraMazo;
```

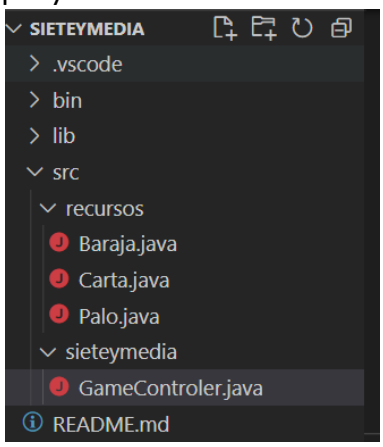
```
        }
        return cartasParaDar;
    }
}
```

```
//GameControler.java
package sieteymedia;

import java.util.Scanner;
import recursos.Baraja;
import recursos.Carta;

public class GameControler {
    Baraja baraja;
    Carta[] cartasJugador;
    Carta[] cartasBanca;
    Scanner sc = new Scanner(System.in);

    public GameControler() {
        baraja = new Baraja();
        baraja.barajar();
        // se van pidiendo cartas al jugar pero matemáticamente a partir de 15 siempre
        // nos pasamos
        // hay 12 cartas de medio puntos, si sacara estas 12 luego cartas con valor 1
        // vemos que a partir de 15 cartas siempre se pasas
        cartasJugador = new Carta[15];
        cartasBanca = new Carta[15];
        presentarJuego();
        jugar();
    }

    public static void main(String[] args) {
        new GameControler();
    }

    void presentarJuego() {
        System.out.println("- El usuario es el jugador y el ordenador la  banca.");
        System.out.println("- No hay en la baraja 8s y 9s. El 10 es la sota, el 11 el caballo y el 12 el Rey.");
        System.out.println("- las figuras (10-sota, 11-caballo y 12-rey) valen medio punto y, el resto, su valor.");
        System.out.println(
                "- Hay dos turnos de juego: el turno del jugador y el turno de la banca. Se comienza por el turno
del jugador.");
        System.out.println("- El jugador va pidiendo cartas a la banca de una en una.");
        System.out.println("- El jugador puede plantarse en cualquier momento.");
        System.out.print("- Si la suma de los valores de las cartas sacadas es superior ");
        System.out.println("a 7 y medio, el jugador 'se pasa de siete y medio' y  pierde.");
        System.out.println(
                "- Si el jugador no se pasa, comienza a sacar cartas la banca y ésta  está obligada a sacar cartas
hasta empatar o superar al jugador.");
        System.out.println(
                "- Si la banca consigue empatar o superar la puntuación del jugador 'sin pasarse de siete y medio',
gana la banca.");
        System.out.println(
                "- La banca no se puede plantar y tiene que empatar o superar la puntuación del  jugador sin
pasarse.");
        System.out.println(
                "- En este proceso puede ocurrir que la banca 'se pase' y entonces pierde la banca y gana el
jugador.");
        System.out.println("\nEmpecemos!!!\n");
    }

    void jugar() {
        turnoJugador();
        turnoBanca();
        System.out.println("Adios");
    }

    void turnoJugador() {
        char opc = 'C';
```

```
    // obligamos a que como mínimo se tenga 1 carta
    System.out.println("Como mínimo recibes una carta, luego puedes decidir si seguir o plantarte");
    while (valorCartas(cartasJugador) < 7.5 && opc == 'C') {
        Carta c = baraja.darCartas(1)[0];
        // insertamos c en las cartas del jugador
        insertarCartaEnArray(cartasJugador, c);
        // mostramos cartas y su valor, si se pasa se sale del bucle
        System.out.println("Éstas son tus cartas jugador:");
        mostrarCartas(cartasJugador);
        double valor = valorCartas(cartasJugador);
        System.out.println("\n\tValor de cartas: " + valor);
        if (valor < 7.5) {
            // suponemos que el usuario teclea bien !!!
            System.out.println("\n¿Pides [C]arta o te [P]lantas?");
            opc = sc.next().trim().toUpperCase().charAt(0);
        }

    }

}

void turnoBanca() {
    // lo primero es consultar el valor que alcanzó el jugador en su turno
    double valorCartasJugador = valorCartas(cartasJugador);
    if (valorCartasJugador > 7.5) {
        System.out.println("Jugador, te has pasado en tu jugada anterior, gana la banca");
        return;
    }
    System.out.println("\n\nTurno de banca ...");

    // juega hasta empatar o superar
    while (valorCartas(cartasBanca) < valorCartasJugador) {
        Carta c = baraja.darCartas(1)[0];
        insertarCartaEnArray(cartasBanca, c);
    }
    System.out.println("Éstas son mis cartas:");
    mostrarCartas(cartasBanca);
    System.out.println("\nValor de  mis cartas(banca): " + valorCartas(cartasBanca));
    if (valorCartas(cartasBanca) > 7.5) {
        System.out.println("me pasé, ganas tú,jugador");
    } else {
        System.out.println("Gana la banca");
    }
}

double valorCartas(Carta[] cartas) {
    double total = 0.0;
    int val;
    int i = 0;
    while (cartas[i] != null) {
        val = cartas[i].getNumero();
        total += (val > 7) ? 0.5 : val;
        i++;
    }

    return total;
}

void insertarCartaEnArray(Carta[] cartas, Carta c) {
    // inserta al final detectando el primer null
    int i = 0;
    while (cartas[i] != null) {
        i++;
    }
    cartas[i] = c;

}

void mostrarCartas(Carta[] cartas) {
    int i = 0;
    while (cartas[i] != null) {
```

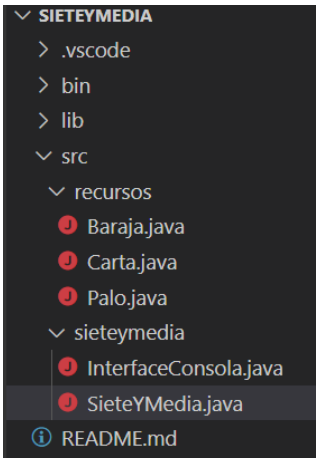
```
            System.out.print("\t" + cartas[i]);
            i++;
        }
    }

}
```

Las tres clases puedes montarlas para probar la aplicación en el siguiente proyecto VSC



Observa que la clase GameControler mezcla **la lógica de negocio y la lógica de presentación, y por lo tanto, claramente no cumple el principio de responsabilidad única**.

SE PIDE: Reestructurar el proyecto SieteYMedia con las siguientes clases y paquetes.



El paquete recursos es el mismo. El paquete sieteymedia es lo que tienes que escribir de forma que haga exactamente lo mismo que Gamecontroler.java pero ahora repartiendo el trabajo y responsabilidad de GameControler en dos clases:

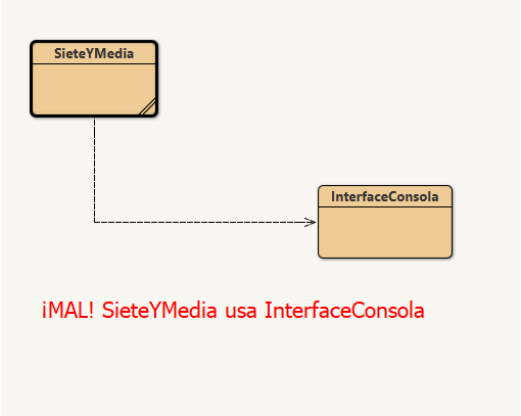
- SieteYMedia.java se ocupe de la lógica de negocio
- InterfaceConsola de la lógica de presentación.

Para conseguir el efecto deseado observa que:

1. La aplicación ahora es más compleja, se divide una clase en dos y surgen nuevos métodos necesarios para la comunicación entre ellas. ¿Merece la pena?. Vamos a suponer que sí porque queremos hacer más adelante de este juego su versión gráfica, de esta manera sólo tendremos que volver a escribir la Interface
2. El teclado y la pantalla sólo los maneja InterfaceConsola. ¡ni un println ni un Scanner en SieteYMedia!
3. ¿Cómo devuelve los datos SieteYMedia a InterfaceConsola?. Supongamos que InterfaceConsola quiere imprimir las cartas que tiene en un momento dado el Jugador ¿Cómo le pasa SieteYMedia esta información a interfaceConsola?. SieteYMedia puede devolver un simple String para que lo imprima InterfaceConsola o bien una estructura más compleja como un array de cartas. La primera es muy sencilla y clara pero la segunda es más flexible ya que no limita a InterfaceConsola a imprimir un String concreto. Usa la segunda.
4. Debes de crear un objeto SieteYMedia y un objeto InterfaceConsola. Esta pequeña App se puede desarrollar perfectamente sin crear objetos y usando metodos static de clase pero EVITA ESTO ¡estamos aprendiendo POO!. Usar static no es pecado mortal pero su abuso anula los principios de POO (más sobre esto más adelante)
5. Tienes que tener totalmente clara la relación entre las clases y por tanto entre los objetos que creas de esas clases

SieteYMedia "usa" InterfaceConsola y esto no es correcto

La flecha de abajo que relaciona las clases se refiere a la relación "usa". El siguiente gráfico indica que SieteYMedio "usa" InterfaceConsola y justamente esto SE DEBE EVITAR para conseguir que la lógica del juego sea independiente de la E/S. Fíjate bien en el sentido de la flecha.



Si un objeto SieteYMedia invoca a un objeto InterfaceConsola() pasa a ser dependiente de esa clase y no es reutilizable para otras aplicaciones que usen otro tipo de entrada salida.

```
class SieteYMedia{
        …..
     InterfaceConsola miconsola= new InterfaceConsola()
        …..
    miconsola.presentarJuego() //¡mal!
        …….

}
```

si hacemos invocaciones a métodos static en lugar de crear un objeto la dependencia persiste y además recuerda que en esta práctica debemos evitar el uso de static, o sea, todavía peor

```
class SieteYMedia{
        …..
       InterfaceConsola.presentarJuego(); //mal
       ……..

}
```

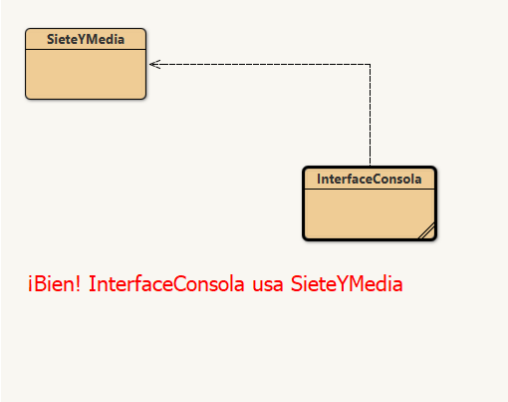
SieteYMedia hace funciones de E/S encubiertas

Otro error que impide la separación en dos capas es que SieteYMedia genere Strings que enmascaran realmente println() de tal forma que InterfaceConsola recibiría unos Strings que puede no desear para comunicarse con el usuario, por ejemplo, se los manda en castellano y los quiere en "Galego" o no le gusta la redacción o lo que sea.

```
 class SieteYMedia{
        …..
      String presentar(){
           return "hola jugador bla bla bla ….";

      }
      …….

}
class InteraceConsola
        …..
     SieteYMedia juego= new SieteYMedia();
      …….
     System.out.println(juego.presentar());//¡pero yo no quiero este texto!
    ……

}
```

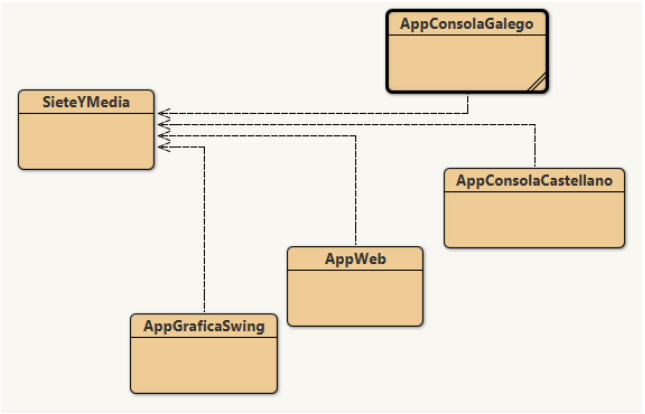
InterfaceConsola "usa" SieteYMedia y es OK

Ahora la flecha cambia de sentido de forma que InterfaceConsola "usa" SieteYMedia




```
class InterfaceConsola{
        …….
        SieteYMedia miObjetoSieteYMedia= new SieteYMedia();
        ……
        System.out.println(miObjetSieteYMedia.valorCartasJugador());
        ……

}
```

podríamos tener muchos clases que desarrollan distintos tipos de interfaces Y que todos utilicen la misma lógica del juego ¡conseguimos reutilizar código!. Para conseguir esto SieteYMedia no puede hacer referencia a ningún tipo de E/S



FORMATO DE ENTREGA: En tarea moodle que se abrirá a la vuelta de vacaciones se subirán dos ficheros pdf:

- InterfaceConsola.pdf que contiene el código de tu solución para la clase InterfaceConsola
- SieteYMedia.pdf, lo mismo para la clase SieteYMedia

En estos ficheros se va a observar simplemente que se cumple la arquitectura a dos capas, no que la lógica de funcionamiento es correcta, de todas formas, procura que funcione bien pues es muy probable que este ejercicio tenga una segunda parte ampliada.