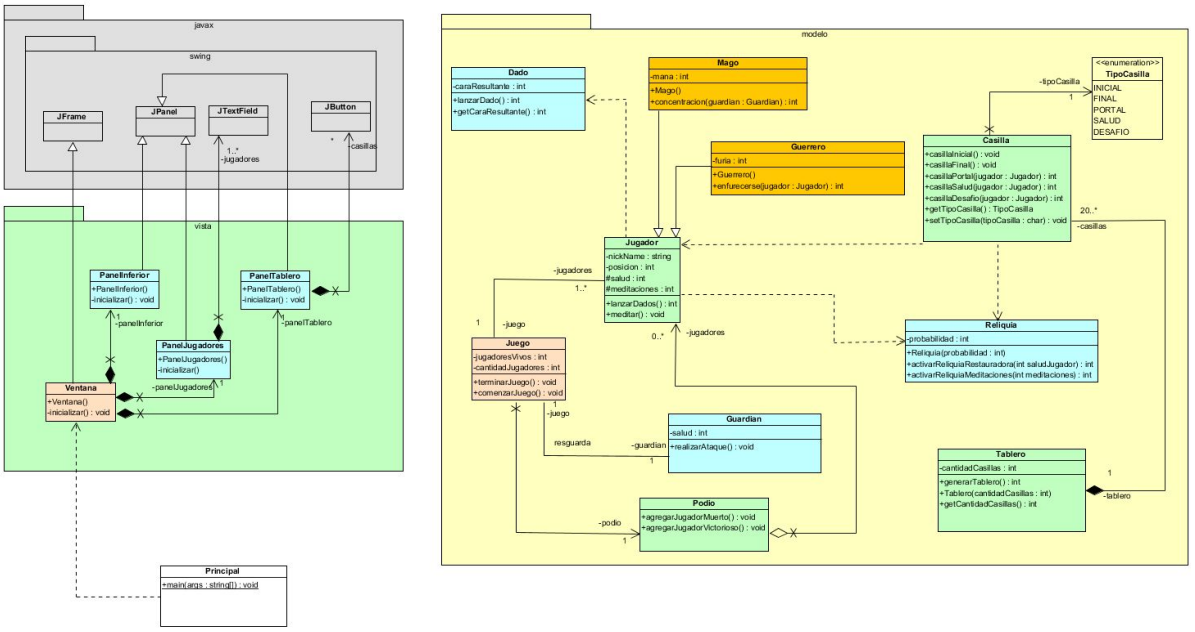
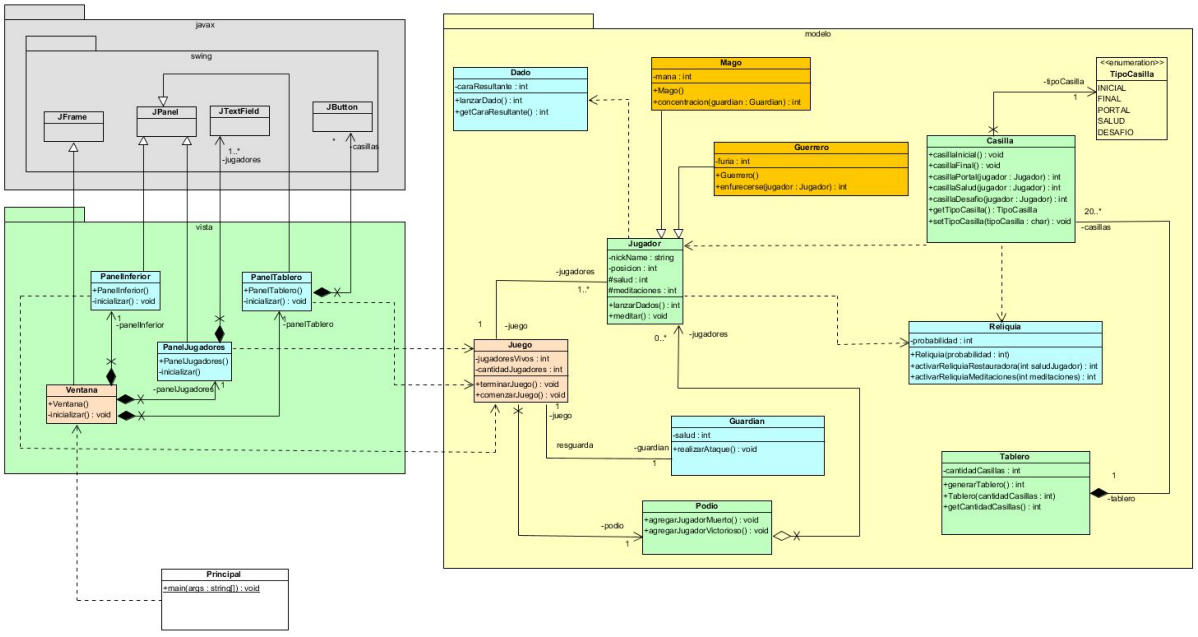
Pauta tarea 10.

En el proceso de convertir el programa en uno con la estructura de tres capas, habíamos logrado construir la capa de vista y la de modelo, sin ninguna conexión por el momento.

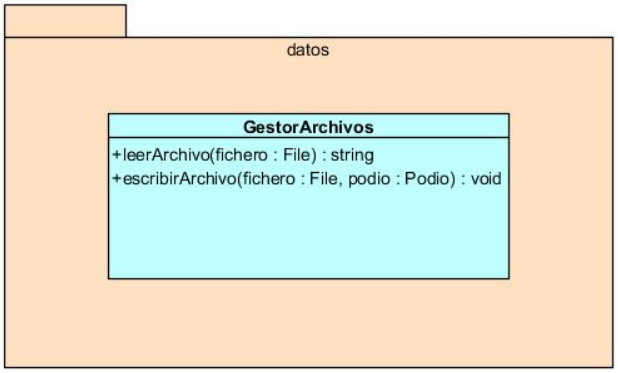


Se debe tener en cuenta que las capas se relacionan entre sí por medio de dependencias, en este caso los paneles son los encargados de llamar al Juego, el cual se comportará como controlador de las demás clases de la capa de modelo.

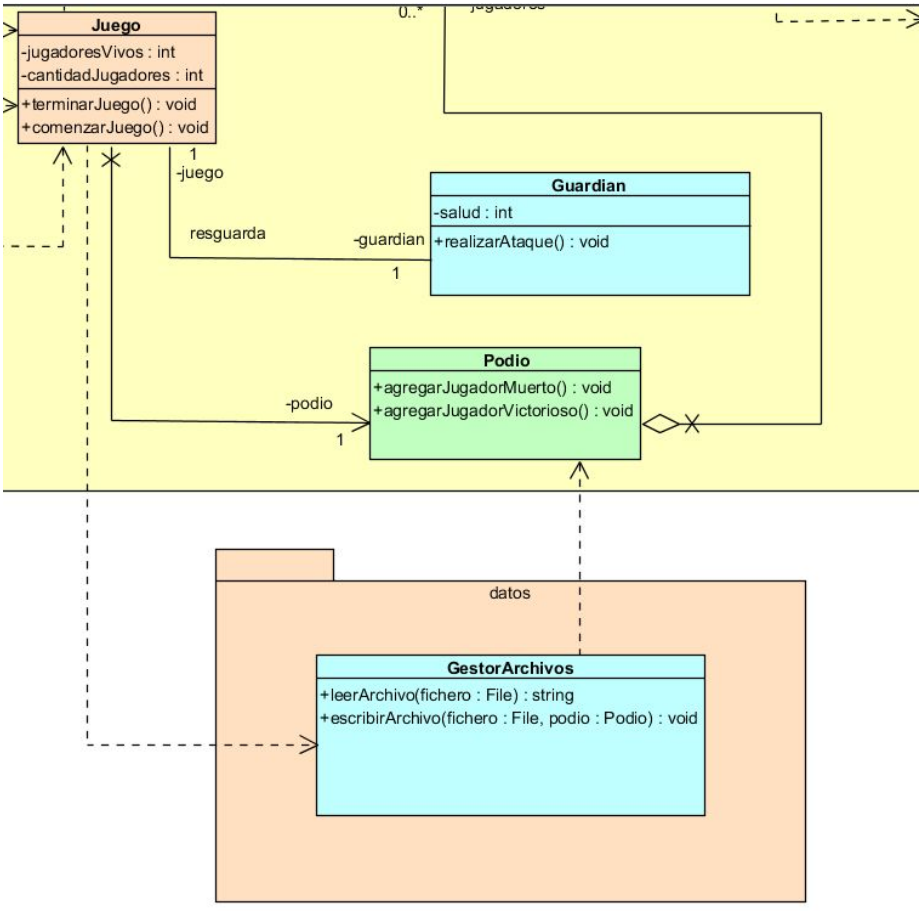


Capa de datos.

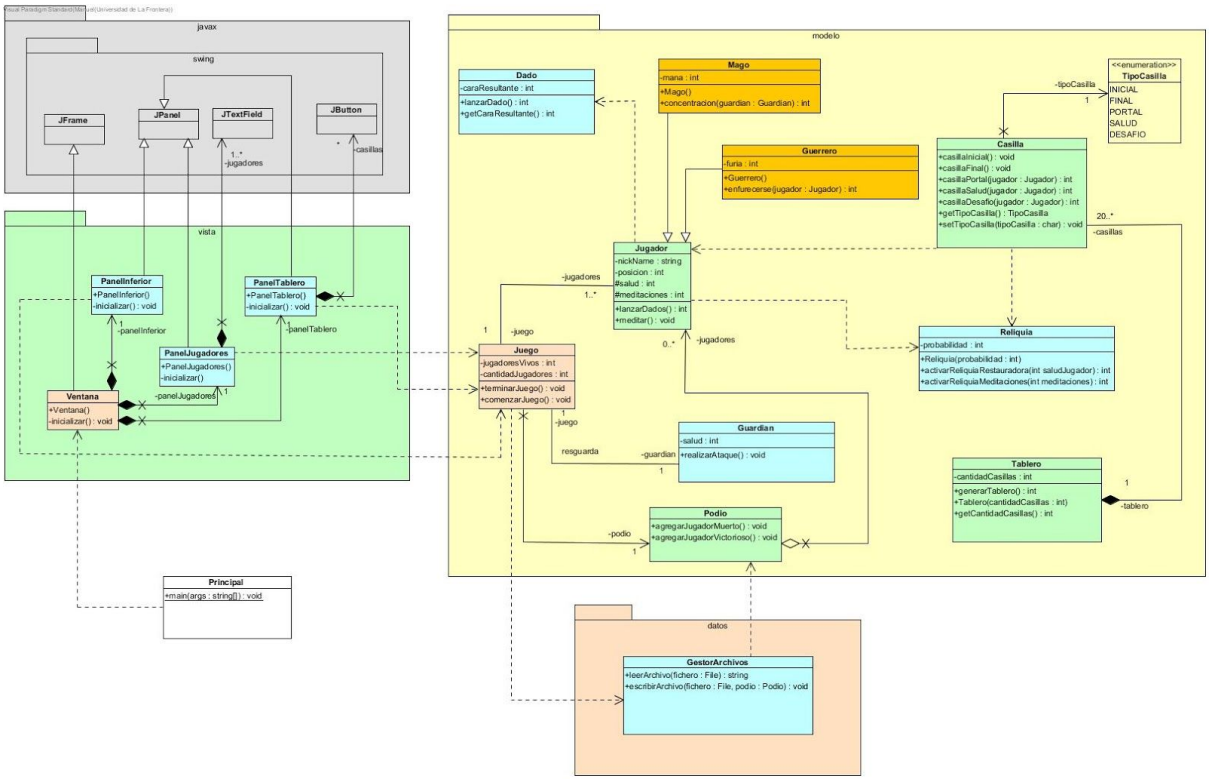
En este caso, la capa de datos consistirá únicamente en una clase, la cual se encargará de leer y escribir el archivo de texto que guardará una especie de historial de partidas terminadas, básicamente con la información del podio y la fecha en que se terminó el juego.



La clase GestorArchivos usa como parámetro un objeto de la clase Podio, es por eso que habrá una dependencia entre estas clases. Por otro lado, la clase Juego es la que se encargará de llamar a la clase GestorArchivos al momento de terminar el juego. Esto se refleja de la siguiente manera en el diagrama:



Finalmente, nuestro diagrama para el juego Infinity Game, utilizando un modelado de tres capas, queda de la siguiente forma:



Código.

Conexión capa de vista con capa de modelo.

La siguiente imagen es un ejemplo de la adaptación de los métodos que se hicieron para la primera tarea respecto a la generación de tablero, para conectarlos con la capa de vista (con el panel de tablero). Estos métodos se encuentran en la clase Tablero, por lo que al momento de iniciar el juego y preguntarle al usuario cuántas casillas habrán en el tablero, se instancia la clase Tablero y se llama al método para generarlo considerando, por ejemplo, los colores que tendrán los botones en el panel.

```java
public static Color[] generarTablero(int cantidadCasillas) {
    Color tablero[] = new Color[cantidadCasillas];

    tablero[0] = Color.WHITE;
    tablero[cantidadCasillas - 1] = Color.BLACK;

    for (int i = 1; i < cantidadCasillas - 1; i++) {
        tablero[i] = casillaRandom();
    }

    return tablero;
}

private static Color casillaRandom() {
    /*
    cyan=blanco, green=portal, yellow=salud, orange=desafio
     */
    Color pool[] = {Color.CYAN, Color.GREEN, Color.YELLOW, Color.ORANGE};

    int indiceRandom = (int) (Math.random() * 4);

    Color selected = pool[indiceRandom];

    return selected;
}
```

Teníamos en la pauta anterior que el botón para hacer una jugada se ubicaba en el panel inferior, a continuación se mostrará de forma general cómo debiera funcionar la conexión de ese botón con la capa de modelo.

En primer lugar tenemos la forma en que se debería armar el evento del botón, junto al JOptionPane para hacer la selección de la jugada a realizar.

```java
@Override
public void actionPerformed(ActionEvent ae) {
    if (this.comenzarTurno == ae.getSource()) {

        String seleccion = elegirTurno();
        Juego juego = new Juego();
        juego.hacerJugada(seleccion, indiceJugadorActual);

    }
}

private String elegirTurno() {
    Object turno = JOptionPane.showInputDialog(null, "Seleccione acción a realizar",
            "Turno", JOptionPane.QUESTION_MESSAGE, null,
            new Object[]{"Lanzar dados", "Meditar", "Habilidad especial"}, "Seleccione");
    return turno.toString();
}
```

Notar que para hacer la jugada como tal se instancia en un objeto de la clase Juego, el cual es nuestro “controlador” en la capa de modelo. La forma en que se maneja el evento desde este controlador, de forma muy general, es la siguiente:

```java
public void hacerJugada(String jugada, int indiceJugador){

    Jugador jugadorActual = jugadores.get(indiceJugador);

    switch (jugada) {
        case "Lanzar dados":
            jugadorActual.lanzarDados();
            //posteriormente se debe hacer la lógica al caer en algún tipo de casilla
            break;
        case "Meditar":
            jugadorActual.meditar();
            break;
        case "Habilidad especial":
            //Habilidad especial según el tipo de jugador (mago/guerrero)
            break;
        default:
            //Cualquier otro caso
            break;
    }
}
```

Capa de datos.

En primer lugar, la clase encargada de la gestión de archivos constará de dos métodos: leer y escribir. En este caso son una adaptación de los métodos que se han estado utilizando en ejemplos anteriores. A continuación se muestra el detalle del código:

```java
public String leerArchivo(File fichero) {
    String texto = "";
    try {
        //Si existe el fichero
        if (fichero.exists()) {
            //Abre un flujo de lectura a el fichero
            BufferedReader bReader = new BufferedReader(new FileReader(fichero));
            String linea;

            //Lee el fichero linea a linea hasta llegar a la ultima
            while ((linea = bReader.readLine()) != null) {
                texto += linea;
            }

            /*Cierra el flujo*/
            bReader.close();
        } else {
            System.out.println("Fichero No Existe");
        }
    } catch (Exception ex) {
        /*Captura un posible error y le imprime en pantalla*/
        System.out.println(ex.getMessage());
    }
    return texto;
}
```

```java
public void escribirFichero(File fichero, Podio podio) {
    try {

        BufferedWriter bw = new BufferedWriter(new FileWriter(fichero));

        String contenido = podio.toString();

        bw.write(contenido);

        bw.close();

    } catch (IOException ex) {
        System.out.println(ex.getMessage());
    }
}
```

La clase Podio, perteneciente a la capa de modelo, se encargará de pasar los datos que se escribirán en el fichero. El método escribirFichero se encarga únicamente de escribir. A continuación se muestra cómo se obtienen los datos que se utilizarán.

En primer lugar, hacemos los métodos toString para la clase jugador.

```java
@Override
public String toString() {
    return "Jugador{" + "nickName=" + nickName + ", posicion=" + posicion +
    ", salud=" + salud + ", meditaciones=" + meditaciones + ", juego=" + juego + '}';
}
```

Ahora se hacen los métodos toString para las clases Mago y Guerrero.

```java
@Override
public String toString() {
    //Toma el toString de la clase Jugador y le agrega el atributo extra del mago
    return super.toString()+" Tipo de jugador: Mago, mana: "+this.mana;
}
```

```java
@Override
public String toString() {
    //Toma el toString de la clase Jugador y le agrega el atributo extra del guerrero
    return super.toString()+" Tipo de jugador: Guerrero, furia: "+this.furia;
}
```

Finalmente, se utilizan estos datos y se ordenan (según se necesiten) en la clase Podio, de la siguiente forma:

```java
private String now(){
    Date now = new Date();
    return now.toString();
}

@Override
public String toString(){

    String contenido="\n";

    contenido += now()+"-";

    int posicion = 1;
    for (Jugador jugador : jugadores) {
        contenido+=posicion+". "+jugador.toString();
        contenido+=";";
        posicion++;
    }
    contenido = contenido.substring(0, contenido.length()-1);

    return contenido;
}
```