**Universidad ORT Uruguay**

**Facultad de Ingeniería**

**Escuela de Tecnología**

**Obligatorio de Diseño y Desarrollo de Aplicaciones**

**Juego de Dominó**

**Primera Entrega: Primera versión del programa**

**Entrega: 27/10/2016**

Bruno DIAZ – 203056  
Federico SPERONI – 165357

Carrera AP – Turno Matutino  
**2016**

# Indice

Tabla de contenido

[Indice 2](#_Toc465327723)

[Autoevaluación 4](#_Toc465327724)

[Autocalificación 4](#_Toc465327725)

[Datos de Prueba 5](#_Toc465327726)

[Diagrama Conceptual del Dominio del Problema 6](#_Toc465327727)

[Diagrama de Diseño 7](#_Toc465327728)

[Diccionario de Clases 8](#_Toc465327729)

[Clase: Usuario 8](#_Toc465327730)

[Atributos 8](#_Toc465327731)

[Métodos 8](#_Toc465327732)

[Clase: Administrador 8](#_Toc465327733)

[Métodos 8](#_Toc465327734)

[Clase: Ficha 9](#_Toc465327735)

[Atributos 9](#_Toc465327736)

[Métodos 9](#_Toc465327737)

[Clase: Apuesta 9](#_Toc465327738)

[Atributos 9](#_Toc465327739)

[Métodos 9](#_Toc465327740)

[Clase: Movimiento 10](#_Toc465327741)

[Atributos 10](#_Toc465327742)

[Métodos 10](#_Toc465327743)

[Clase: Jugador 11](#_Toc465327744)

[Atributos 11](#_Toc465327745)

[Métodos 11](#_Toc465327746)

[Clase: Partida 12](#_Toc465327747)

[Atributos 12](#_Toc465327748)

[Métodos 12](#_Toc465327749)

[Clase: SubSistemaUsuario 14](#_Toc465327750)

[Atributos 14](#_Toc465327751)

[Métodos 14](#_Toc465327752)

[Clase: SubSistemaPartida 14](#_Toc465327753)

[Atributos 14](#_Toc465327754)

[Métodos 14](#_Toc465327755)

[Clase: Sistema 15](#_Toc465327756)

[Atributos 15](#_Toc465327757)

[Métodos 15](#_Toc465327758)

[Anexo 16](#_Toc465327759)

[Justificaciones de implementación e interpretaciones de la letra 16](#_Toc465327760)

# Autoevaluación

Consideramos que cumplimos con todos los requerimientos solicitados en el sistema, habiendo tenido especial cuidado con la aplicación del GRASP Experto para la delegación de responsabilidades, así como en el correcto uso de una arquitectura M.V.C.

En cuanto a la división física del código, se realizó en cuatro paquetes, según de detallan a continuación:

controladores: Posee todas las clases controladoras de la parte de interfaz de usuario, así como las interfaces contra las que trabajan los mismos.

logica: Posee toda las clases del dominio, así como los sub sistemas y la fachada

utilidades: Contiene la clases ObligatorioException, la cual hereda de Exception y es utilizada para lanzar excepciones personalizadas, así como también contiene la clase interface necesarias para la implementación del patrón del Observador

vista: Posee todo lo relacionado con lo que el usuario visualiza, incluso el botón personalizado (Clase BotonFicha que hereda de JButton), así como la clase Inicio que contiene el main.

## Autocalificación

|  |  |  |  |
| --- | --- | --- | --- |
| **CONCEPTO** | **PTS. POSIBLES** | **CUMPLIMIENTO?** | **AUTOCALIFICACIÓN** |
| Funcionalidad: Implementación de la funcionalidad solicitada en Java. | (-19) | SI | 0 |
| Diagramas de clases y concordancia del código con los diagramas. | 3 | SI | 3 |
| Requerimientos de diseño. | 16 | SI | 16 |
| Auto Evaluación | 1 | SI | 1 |
| **Total:** | **20** |  | **20** |

# Datos de Prueba

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **JUGADORES** | | | | |
| **Cantidad** | **Nombre de Usuario** | **Contraseña** | **Nombre Completo** | **Saldo** |
| 1 | a | a | Alvaro Fernández | 450 |
| 2 | b | b | Bruno Díaz | 400 |
| 3 | c | c | Carlos González | 350 |
| 4 | d | d | Darío Pérez | 300 |
| 5 | e | e | Emiliano Lasa | 250 |
| 6 | f | f | Federico Speroni | 200 |
| 7 | g | g | Gustavo Valverde | 150 |
| 8 | h | h | Hugo Collazo | 100 |
| 9 | i | i | Ismael Espósito | 50 |
| 10 | j | j | Javier Montero | 0 |

Nota: En nuestra solución, la apuesta inicial de todas las partidas es de 100 por jugador, por lo que todos los que poseen saldo menor a 100, no podrán ingresar al juego.

|  |  |  |  |
| --- | --- | --- | --- |
| **ADMINISTRADORES** | | | |
| **Cantidad** | **Nombre de Usuario** | **Contraseña** | **Nombre Completo** |
| 1 | a | a | Analía Pereyra |
| 2 | b | b | Blanca Moreira |
| 3 | c | c | Claudia Tabárez |
| 4 | d | d | Dilma Rousseff |
| 5 | e | e | Emilia Suárez |
| 6 | f | f | Fabiana Guerra |
| 7 | g | g | Graciela García |
| 8 | h | h | Heidy Montero |
| 9 | i | i | Ilda De León |
| 10 | j | j | Judith Barsi |

# Diagrama Conceptual del Dominio del Problema

(Pegar Diagrama)

# Diagrama de Diseño

Notas:

Entendimos que el diagrama a presentar es completo, pero solamente debe incluir las clases correspondiente a la lógica.

Para que el diagrama sea más eficiente arrojando información más concreta, en el mismo se representan las relaciones entre las clases, detallando los atributos y métodos de las mismas en forma de lista.

(Pegar diagrama)

# Diccionario de Clases

## Clase: Usuario

### Atributos

- nombre : String

- password : String

- nombreCompleto : String

### Métodos

+ getNombre() : String

+ getPassword() : String

+ getNombreCompleto() : String

+ Usuario(n : String, p : String, nc : String)

## Clase: Administrador

### Métodos

+ Administrador(n : String, p : String, np : String)

## Clase: Ficha

### Atributos

- valorDerecha : int

- valorIzquierda : int

### Métodos

- rotar() : void

+ getValorDerecha() : int

+ getValorIzquierda() : int

+ sePuedeUnir(lado : String, fichaDescartada : Ficha) : void throws ObligatorioException

+ Ficha(valorIzquierda : int, valorDerecha : int)

## Clase: Apuesta

### Atributos

- valor : double

- jugador : Jugador

### Métodos

+ getValor() : double

+ getJugador() : Jugador

+ setValor(valor : double) : void

+ setJugador(jugador : Jugador) : void

+ Apuesta(valor : double)

## Clase: Movimiento

### Atributos

- ganador : Jugador

- fechaHora : Date

- pozoApuestas : double

- jugador : Jugador

- tablero : ArrayList<Ficha>

### Métodos

+ getGanador() : Jugador

+ getFechaHora() : Date

+ getPozoApuestas() : double

+ getJugador() : Jugador

+ getTablero() : ArrayList<Ficha>

+ setGanador(ganador : Jugador) : void

+ Movimiento(ganador : Jugador, pozoApuestas : double, jugador : Jugador, tablero : ArrayList<Ficha>)

## Clase: Jugador

### Atributos

- saldo : double

- mano : ArrayList<Ficha>

- partidaJugando : Partida

### Métodos

+ setSaldo(saldo : double) : void

+ getSaldo() : double

+ verificarSaldo(monto : double) : void throws ObligatorioException

+ agregarFicha(f : Ficha) : void

+ getFichas() : ArrayList<Ficha>

+ quitarApuesta(valor : double) : void throws ObligatorioException

+ eliminarFicha(ficha : Ficha) : void

+ vaciarMano() : void

+ Jugador(n : String, p : String, np : String, s : double)

## Clase: Partida

### Atributos

- partidaActiva : boolean

- pozoApuestas : double

- jugadores : ArrayList<Jugador>

- turno : Jugador

- ganador : Jugador

- movimientos : ArrayList<Movimiento>

- ultimaApuesta : Apuesta

- libres : ArrayList<Ficha>

- tablero : ArrayList<Ficha>

- primera : Ficha

- ultima : Ficha

### Métodos

+ jugadorAbandonando(jugador : Jugador) : void

+ getLibres() : ArrayList<Ficha>

+ getTurno() : Jugador

+ getTablero() : ArrayList<Ficha>

+ getGanador() : Jugador

+ getPozoApuestas() : double

+ getJugadores() : ArrayList<Jugador>

+ getUltimaApuesta() : Apuesta

+ getMovimientos() : ArrayList<Movimiento>

+ agregarJugador(jugador : Jugador) : void throws ObligatorioException

+ mover(jugador : Jugador, fichaTablero : Ficha, fichaDescartada : Ficha) : void throws ObligatorioException

- controlesAntesJugar(jugador : Jugador) : void throws ObligatorioException

+ robar(j : Jugador) : void throws ObligatorioException

+ apostar(apostador : Jugador, monto : double) : void throws ObligatorioException

+ confirmarApuesta(confirmación : boolean) : void throws ObligatorioException

+ primerJugada(fichaDescartada : Ficha) : void throws ObligatorioException

+ segundaJugada(fichaTablero : Ficha, fichaDescartada : Ficha) : void throws ObligatorioException

- unirFicha(fichaTablero : Ficha, fichaDescartada : Ficha) : void throws ObligatorioException

- restarMontoJugadoresSumarApuestaEnPartida() : void throws ObligatorioException

- repartirFichas() : void

- mezclarFichas() : void

- cambiarTurno() : void

- finalizarPartida(ganador : Jugador) : void

- verificarTurno(jugador : Jugador) : void throws ObligatorioException

- verificarSiSeDescartoTodas() : void

- verificarSiTieneMovimientos(j : Jugador) : boolean

- verificarApuesta(monto : double) : void throws ObligatorioException

- verificarUltimoEnApostar(apostador : Jugador) : void throws ObligatorioException

- sePuedeJugar() : void throws ObligatorioException

- agregarMovimiento() : void

- crearFichas() : void

+ Partida()

## Clase: SubSistemaUsuario

### Atributos

- administradores : ArrayList<Administrador>

- jugadores : ArrayList<Jugador>

- jugadoresLogueados : ArrayList<Jugador>

### Métodos

+ loginJugador(n : String, p : String) : Jugador throws ObligatorioException

+ loginAdministrador(n : String, p : String) : Administrador throws ObligatorioException

+ logoutJugador(jug : Jugador) : void

- boolean estaLogueado(n : String) : boolean

- cargarUsuarios() : void

+ SubSistemaUsuario()

## Clase: SubSistemaPartida

### Atributos

- partidas : ArrayList<Partida>

- proximaPartida : Partida

### Métodos

+ partidaParaJugar() : Partida

+ partidasFinalizadas() : void throws ObligatorioException

+ getPartidas() : ArrayList<Partida>

+ agregarJugador(j : Jugador) : void throws ObligatorioException

+ SubSistemaPartida()

## Clase: Sistema

### Atributos

- ssu : SubSistemaUsuario

- ssp : SubSistemaPartida

- instancia : Sistema <<static>>

### Métodos

+ getInstancia() : Sistema <<static>

+ loginJugador(nombre : String, pass : String) : Jugador throws ObligatorioException

+ loginAdministrador(nombre : String, pass : String) : Administrador throws ObligatorioException

+ partidasFinalizadas() : void throws ObligatorioException

+ getPartidas() : ArrayList<Partida>

+ agregarJugador(j : Jugador) : void throws ObligatorioException

+ getPartidaParaJugar() : Partida

+ logoutJugador(jug : Jugador) : void

- Sistema()

# Anexo

## Justificaciones de implementación e interpretaciones de la letra

En cuanto a los eventos, podríamos tener solo uno, ya que todos realizan las mismas operaciones, pero se crearon diferentes porque en un futuro pueden surgir diferentes acciones a realizar, dependiendo de la acción efectuada.

Diagrama Conceptual del Dominio

No va la clase ObligatorioException

Si hay que especificar quien es observador y quien observable (O mostrando la herencia y la implementación, o con un esteriotipo)

Ver en el diagrama conceptual del dominio a ver si van las relaciones como PrimerFicha y UltimaFicha en Partida… (Y en el otro?)

Diagrama de la Interfaz de Usuario

Alcanza con poner solo los controladores y las vistas, no son necesarias las clases que implementan las interfaces de las vistas

Si hay que especificar quien es observador y quien observable (O mostrando la herencia y la implementación, o con un esteriotipo)

Diagrama Completo del Dominio

No están los Enum (Ver si hay que ponerlos de alguna forma)

Ver si hay que aclarar de alguna forma los métodos que lanzan excepciones personalizadas (para identificarlos, actualmente tienen escrito el throws ObligatorioException. Si no es necesario, borrar eso de aquellos que lo tengan)

Averiguar si los nombres de los parámetros en el diagrama (En este caso, en el Diccionario de CLases) tienen que ser exactamente los mismos que en el código, o si acá pueden ser un poco más nemotécnicos