**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](#_Toc465334072)

[Autoevaluación 4](#_Toc465334073)

[Autocalificación 4](#_Toc465334074)

[Datos de Prueba 5](#_Toc465334075)

[Diagrama Conceptual del Dominio del Problema 6](#_Toc465334076)

[Diagramas de Diseño 7](#_Toc465334077)

[Diagrama de Diseño - Lógica 7](#_Toc465334078)

[Diagrama de Diseño – Interfáz de Usuario 8](#_Toc465334079)

[Diccionario de Clases 9](#_Toc465334080)

[Clase: Usuario 9](#_Toc465334081)

[Atributos 9](#_Toc465334082)

[Métodos 9](#_Toc465334083)

[Clase: Administrador 9](#_Toc465334084)

[Métodos 9](#_Toc465334085)

[Clase: Ficha 10](#_Toc465334086)

[Atributos 10](#_Toc465334087)

[Métodos 10](#_Toc465334088)

[Clase: Apuesta 10](#_Toc465334089)

[Atributos 10](#_Toc465334090)

[Métodos 10](#_Toc465334091)

[Clase: Movimiento 11](#_Toc465334092)

[Atributos 11](#_Toc465334093)

[Métodos 11](#_Toc465334094)

[Clase: Jugador 12](#_Toc465334095)

[Atributos 12](#_Toc465334096)

[Métodos 12](#_Toc465334097)

[Clase: Partida 13](#_Toc465334098)

[Atributos 13](#_Toc465334099)

[Métodos 13](#_Toc465334100)

[Clase: SubSistemaUsuario 15](#_Toc465334101)

[Atributos 15](#_Toc465334102)

[Métodos 15](#_Toc465334103)

[Clase: SubSistemaPartida 15](#_Toc465334104)

[Atributos 15](#_Toc465334105)

[Métodos 15](#_Toc465334106)

[Clase: Sistema 16](#_Toc465334107)

[Atributos 16](#_Toc465334108)

[Métodos 16](#_Toc465334109)

[Anexo 17](#_Toc465334110)

[Justificaciones de implementación e interpretaciones de la letra 17](#_Toc465334111)

# 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** |

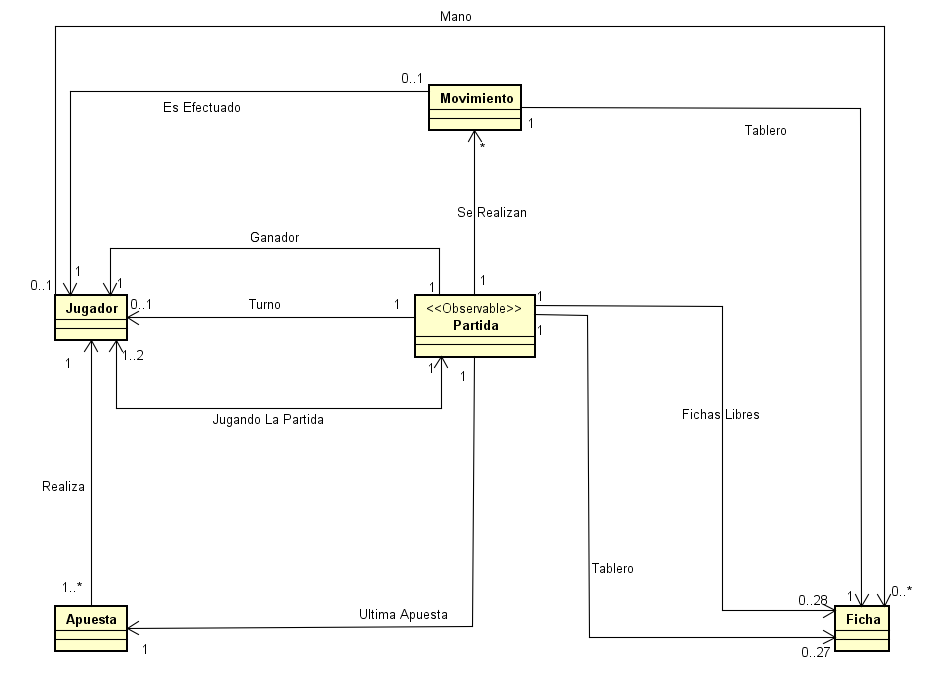
# Datos de Prueba

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 |

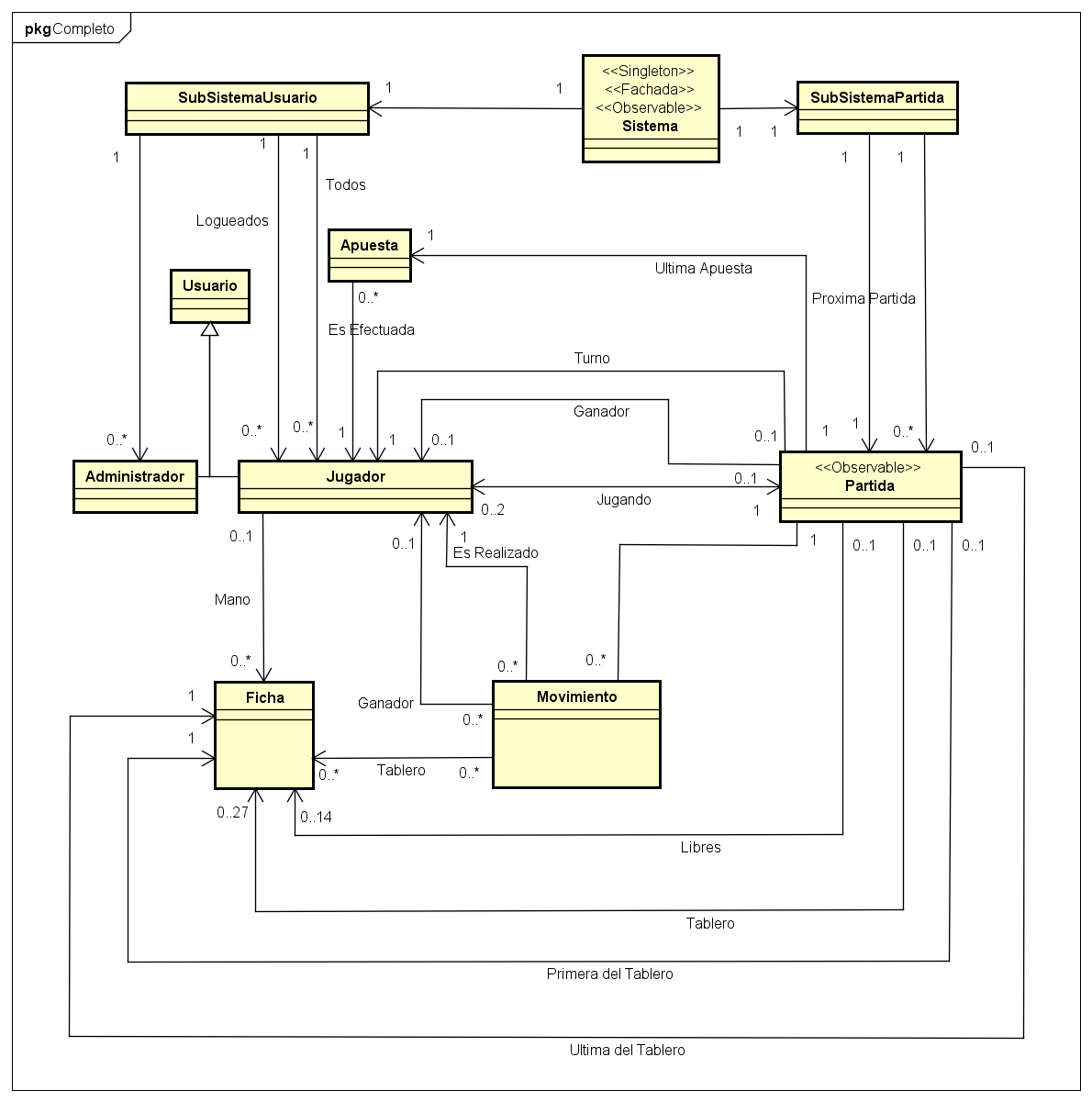
# Diagrama Conceptual del Dominio del Problema



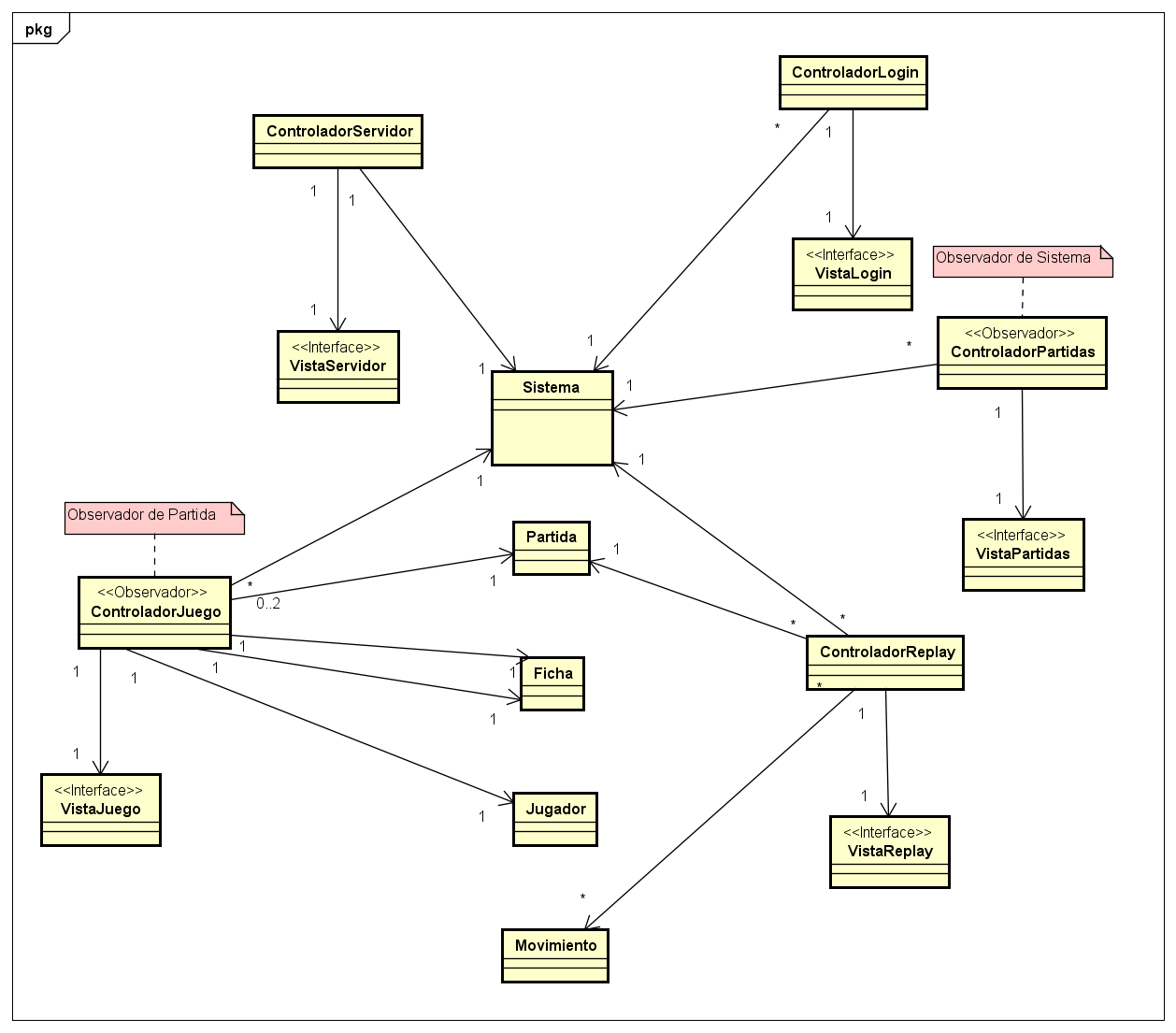
# Diagramas de Diseño

Nota: 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 un Diccionario de Clases presente en este documento.

## Diagrama de Diseño - Lógica



## Diagrama de Diseño – Interfáz de Usuario



# 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

+ 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

+ agregarFicha(f : Ficha) : void

+ getFichas() : ArrayList<Ficha>

+ quitarApuesta(valor : double) : void

+ 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

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

- controlesAntesJugar(jugador : Jugador) : void

+ robar(j : Jugador) : void

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

+ confirmarApuesta(confirmación : boolean) : void

+ primerJugada(fichaDescartada : Ficha) : void

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

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

- restarMontoJugadoresSumarApuestaEnPartida() : void

- repartirFichas() : void

- mezclarFichas() : void

- cambiarTurno() : void

- finalizarPartida(ganador : Jugador) : void

- verificarTurno(jugador : Jugador) : void

- verificarSiSeDescartoTodas() : void

- verificarSiTieneMovimientos(j : Jugador) : boolean

- verificarApuesta(monto : double) : void

- verificarUltimoEnApostar(apostador : Jugador) : void

- sePuedeJugar() : void

- agregarMovimiento() : void

- crearFichas() : void

+ Partida()

## Clase: SubSistemaUsuario

### Atributos

- administradores : ArrayList<Administrador>

- jugadores : ArrayList<Jugador>

- jugadoresLogueados : ArrayList<Jugador>

### Métodos

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

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

+ 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

+ getPartidas() : ArrayList<Partida>

+ agregarJugador(j : Jugador) : void

+ SubSistemaPartida()

## Clase: Sistema

### Atributos

- ssu : SubSistemaUsuario

- ssp : SubSistemaPartida

- instancia : Sistema <<static>>

### Métodos

+ getInstancia() : Sistema <<static>

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

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

+ partidasFinalizadas() : void

+ getPartidas() : ArrayList<Partida>

+ agregarJugador(j : Jugador) : void

+ 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.