

Proyecto - iteración 3.  
Juego James Bond

Fabián Orozco Chaves - B95690  
Daniel Escobar Giraldo - C02748  
Manuel Arroyo Portilla - A80675  
Wilmer Araya Rivas - B80538

### Justificación de diseño MARDA

<div data-bbox="118 707 759 1039"><p style="text-align: center;"><b>&lt;&lt;interface&gt;&gt; IArbitro</b></p><hr/><p>+ <i>decidirGanador(): JugadorMarda</i> + <i>validarJugada(JugadorMarda): Boolean</i> + <i>cambiarTurno()</i> + <i>asignarTurnoInicial(String)</i></p></div>	<p>Debido a que habían grupos con validadores y árbitros, decidimos implementar una <b>interfaz árbitro</b> con <b>métodos reutilizables</b> que fueran <b>separados del concepto del tablero</b> para que sea más fácil de utilizar por aparte. Este <b>permite</b> decidir el ganador del juego, validar una jugada, cambiar de turno entre jugadores y asignar el turno inicial en el juego. <b>Cada equipo los implementa en su juego</b> concreto a conveniencia.</p> <p>Principios: SRP, OCP, ISP.</p>
<div data-bbox="108 1290 769 1644"><p style="text-align: center;"><b>&lt;&lt;Rol controlador, director&gt;&gt; JuegoMarda</b></p><hr/><p># jugador1 : JugadorMarda # jugador2 : JugadorMarda # tablero : ContenedorDeCartasMarda # turnoActual: JugadorMarda</p><hr/><p>+ <i>start()</i> + <i>getJugadorMarda(jugador : int)</i> + <i>guardar(IConstructorSerializadorAbstracto, JuegoMarda)</i> + <i>cargar (IConstructorDeserializadorAbstracto, JuegoMarda) : boolean</i></p></div>	<p>Como parte de las cosas en común, cada juego contaba con <b>dos jugadores</b> donde uno tenía el <b>turno activo</b> y otro pasivo, además de un <b>tablero</b> donde se realizaban las jugadas con cartas. Por eso implementamos una <b>clase controladora abstracta</b> de la que pueden extender juegos de este tipo. El método <b>cargar y guardar el juego son concretos</b> de la clase por lo que el juego que la extienda no tiene que implementarlo (utiliza el <b>patrón constructor</b>).</p> <p>Principios: SRP, OCP, DIP.</p>
<p>Quisimos implementar un <b>método plantilla</b> encargado del flujo del juego, sin embargo para nuestro juego <b>se imposibilita utilizarlo</b> debido a que utiliza temporizadores, por lo que no lo incluimos en el diseño.</p>	

<div> <div>GrupoDeCartasMarda</div> <div> -cartas : Vector&lt;Carta&gt; </div> <div> + agregarCarta(carta : Carta)  + quitarCarta(indice : int): Carta  + getCarta(indice : int): Carta  + buscarCarta(numero : int, palo : char): int  + barajarCartas()  + getCartas() : Vector&lt;Carta&gt;  + clear()  + isEmpty() Boolean </div> </div>	<p>De las cosas en comunes entre los grupos encontramos que varias clases tenían funciones muy similares de manejo de cartas como por ejemplo Mazo, Pila, Mano, etc; por lo cual decidimos implementar <b>una clase capaz de crear y manejar</b> estos diferentes grupos de cartas.</p> <p>Principios: SRP, OCP.</p>
--	--

<div> <div>ContenedorDeCartasMarda</div> <div> -gruposDeCartas: Vector&lt;GrupoDeCartas&gt; </div> <div> + agregarGrupoDeCartas(grupo : GrupoDeCartas)  + quitarGrupoDeCartas(indice : int)  + getGrupoDeCartas(indice : int): GrupoDeCartas </div> </div>	<p>De las cosas en comunes entre los grupos encontramos tableros y jugadores decidimos implementar una clase capaz de contener grupos de cartas con <b>métodos reutilizables</b>. Esto <b>permite</b> manejar uno o varios grupos de cartas, en el mismo ente. Esto permite que un tablero pueda mostrar una pila de descarte o cartas comunes y que un jugador posea más de una mano.</p> <p>Principios: SRP, OCP, DIP.</p>
--	--

<div> <div>JugadorMarda</div> <div> -nombre: String </div> <div> + setNombre(nombre : String)  + getNombre(): String </div> </div>	<p>De las cosas en comunes entre los grupos encontramos jugadores y decidimos crear la clase jugador la cual <b>hereda</b> de <b>ContenedorDeCartasMarda</b> y así seguir usando sus <b>métodos reutilizables</b> junto con los agregados en el hijo. Principios: SRP, OCP, LSP.</p>
--	--

<pre> classDiagram     class IConstructorSerializadorAbstracto {         &lt;&lt;Rol constructor abstracto, interface&gt;&gt;         +inicioObjeto(nombreObjeto: String)         +finObjeto()         +obtenerSerializacion(): String         +serializarJuego(JuegoMarda)         +serializarTablero(ContenedorDeCartasMarda)         +serializarJugador(JugadorMarda)         +serializarGrupoDeCartas(GrupoDeCartas)         +serializarCarta(Carta)     }     class ConstructorSerializadorJSON {         &lt;&lt;Rol constructor concreto&gt;&gt;         -serialización : String         + ConstructorSerializadorJSON()         + inicioObjeto()         + finObjeto()         + obtenerSerialización(): String         -jsonFormatWithComa(String, String): String         -jsonFormat(String, String): String         -sQts(String): String         + serializarJamesBond()         + serializarTablero(Tablero)         + serializarJugador(Jugador)         + serializarGrupoDeCartas(GrupoDeCartas)         + serializarCarta(Carta)     }     IConstructorSerializadorAbstracto &lt; -- ConstructorSerializadorJSON     JuegoMarda --&gt; IConstructorSerializadorAbstracto </pre>	<p>Para cumplir con la función de <b>serializar y deserializar</b> (guardar y cargar) el estado del juego, se implementó el <b>patrón Constructor</b>, donde quien cumple el rol <b>director</b> es el controlador <b>JuegoMarda</b>, esto con el objetivo de que se exista la posibilidad de que a futuro se puedan implementar diferentes tipo de representaciones del juego (XML, csv, json, etc)</p>
--	--

y estas ser elegidas dinámicamente sin afectar el funcionamiento del juego.

Principios: SRP, LSP y DIP, ISP.

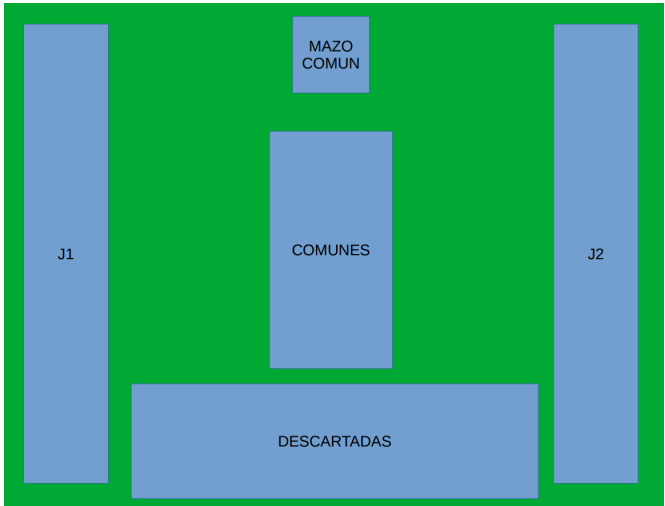
<<Rol Vista>>  
**VistaTableroMarda**

- alturaVentana : int
- anchoVentana : int
- estructura : BordePane
- seccionEste : VBox
- seccionOeste : VBox
- seccionCentro : VBox
- seccionNorte : HBox
- seccionSur : HBox

- + VistaTablero(altura : int, ancho : int)
- + destruirTablero()
- + agregarElemento(seccion : Pane, elemento : Node)
- + getSeccionEste() : VBox
- + getSeccionOeste() : VBox
- + getSeccionCentral() : VBox
- + getSeccionNorte() : HBox
- + getSeccionSur() : HBox

Esta clase se encarga de manejar la construcción de la **interfaz gráfica del tablero**. Para usarla se debe llamar al método *construirTablero(Stage)*. Este inicializa las secciones del tablero pero no le agrega elementos. El usuario puede agregar elementos a la sección que desea con el método *agregarElemento(Pane, Node)*. La sección se obtiene con los métodos get y se puede agregar todo elementos gráfico de javaFX como botones, texto, imágenes, etc.

Si se construye un tablero se crea un tablero vacío pero que tiene las siguientes secciones:



El desarrollador hereda de esta clase para crear un tablero concreto que le agrega la funcionalidad que sea necesaria.

Principios: SRP, ISP.

<table><tr><th>Carta</th></tr><tr><td>-imagen: Image -palo: char -numero: int</td></tr><tr><td>+Carta(numero: int, palo: char) +getNumero(): int +getPalo: char +getImagen(): Image</td></tr></table>	Carta	-imagen: Image -palo: char -numero: int	+Carta(numero: int, palo: char) +getNumero(): int +getPalo: char +getImagen(): Image	<p>Marda ofrece la clase carta la cual contiene el valor y el palo de la misma. Además esta clase genera una dirección de donde extraer la imagen que utilizará el vista carta con el siguiente formato:</p> <p style="text-align: center;"><i>/img/palo_numero.png</i></p> <p>Por lo tanto, el desarrollador quiere cambiar el tiempo de mazo que se va a utilizar, es suficiente con asignar a las cartas los palos y números correspondientes del tipo de mazo y en la carpeta img/ poner el nombre de las imágenes siguiendo el formato indicado para que se carguen correctamente.</p> <p>De esta clase no es necesario heredar y agregar funcionalidad porque ya realiza lo necesario. No obstante el desarrollador es libre de hacerlo.</p> <p>Principios: SRP, OCP, DIP.</p>
Carta				
-imagen: Image -palo: char -numero: int				
+Carta(numero: int, palo: char) +getNumero(): int +getPalo: char +getImagen(): Image				

<div>&lt;&lt;Rol vista, Adaptador&gt;&gt; <b>VistaCarta</b></div> <div>-imagenCarta : Image -imagenVistaCarta : ImageView</div> <div>+VistaCarta(Carta) +getView(boolean): ImageView +getImageView() : ImageView +resaltarCarta() +normalizarCarta()</div>	<p>Esta clase de marda ya realiza lo necesario para ser utilizada por un desarrollador. Se puede heredar de esta y agregar funcionalidad, pero si solo se van a mostrar cartas ya hace lo necesario. El constructor de esta recibe una carta y crea una vista a partir de esta la cual puede ser agregada al tablero para visualizarla.</p> <p>Las imágenes utilizadas para crear las vistas se encuentran en la carpeta src/img . El desarrollador puede cambiar estas imágenes para crear vistas de diferentes mazos de cartas.</p> <p>Principios: SRP, OCP.</p>
--	--

#### Justificación DIP:

Los módulos Carta, ContenedorDeCartasMarda y JuegoMarda no dependen uno del otro ya que sin importar la especialización gracias a la conexión con grupo de cartas.