PROYECTO CONCENTRATE PINTCON

ELABORADO POR:

ANDREA MILENA CAMARGO GONZÁLEZ 20191578121

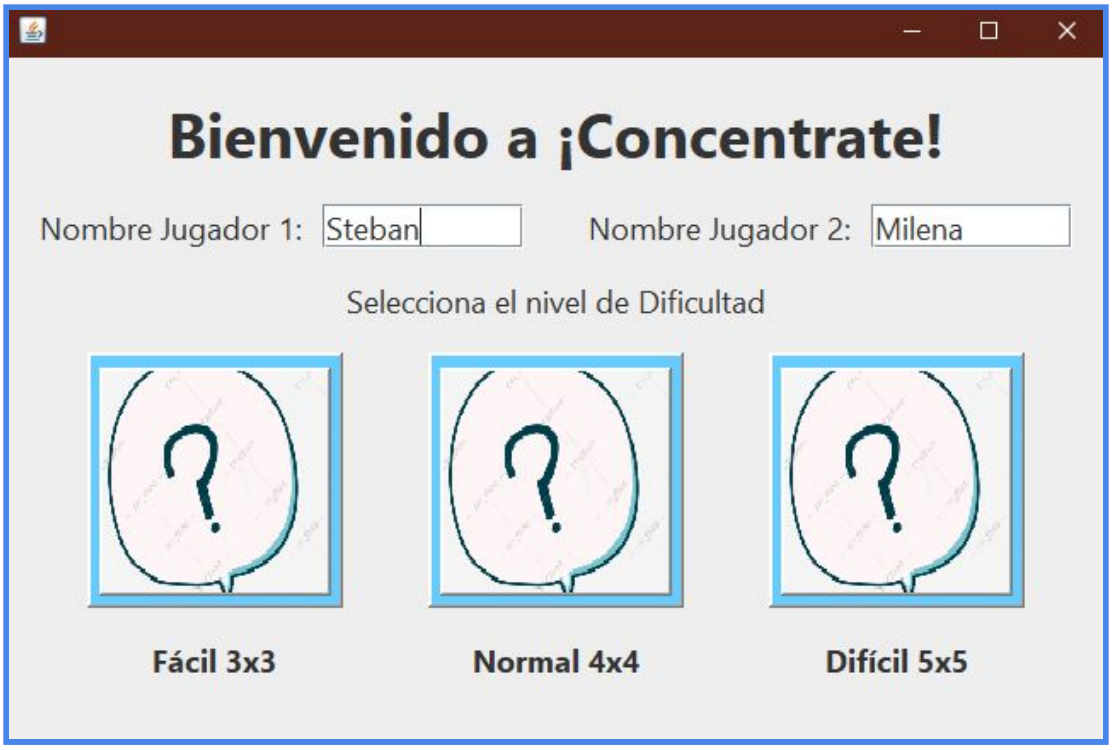
BRAYAN STEBAN CANTOR MUNEVAR 20191578017

PROGRAMACIÓN ORIENTADA A OBJETOS

BOGOTA D.C

2020

OBJETIVOS DEL PROGRAMA



OBJETIVO GENERAL

Diseñar a través de la programación orientada a objetos aplicando el patrón Modelo Vista Controlador (MVC) un programa recreativo, llamado: “Concéntrese PINTCON”, un juego que implementa tres dificultades: Fácil, Normal y Difícil; y correspondientemente, tienen un tamaño paralelo propio a la dificultad, para causar mayor diversión y entretenimiento.

OBJETIVOS ESPECÍFICOS

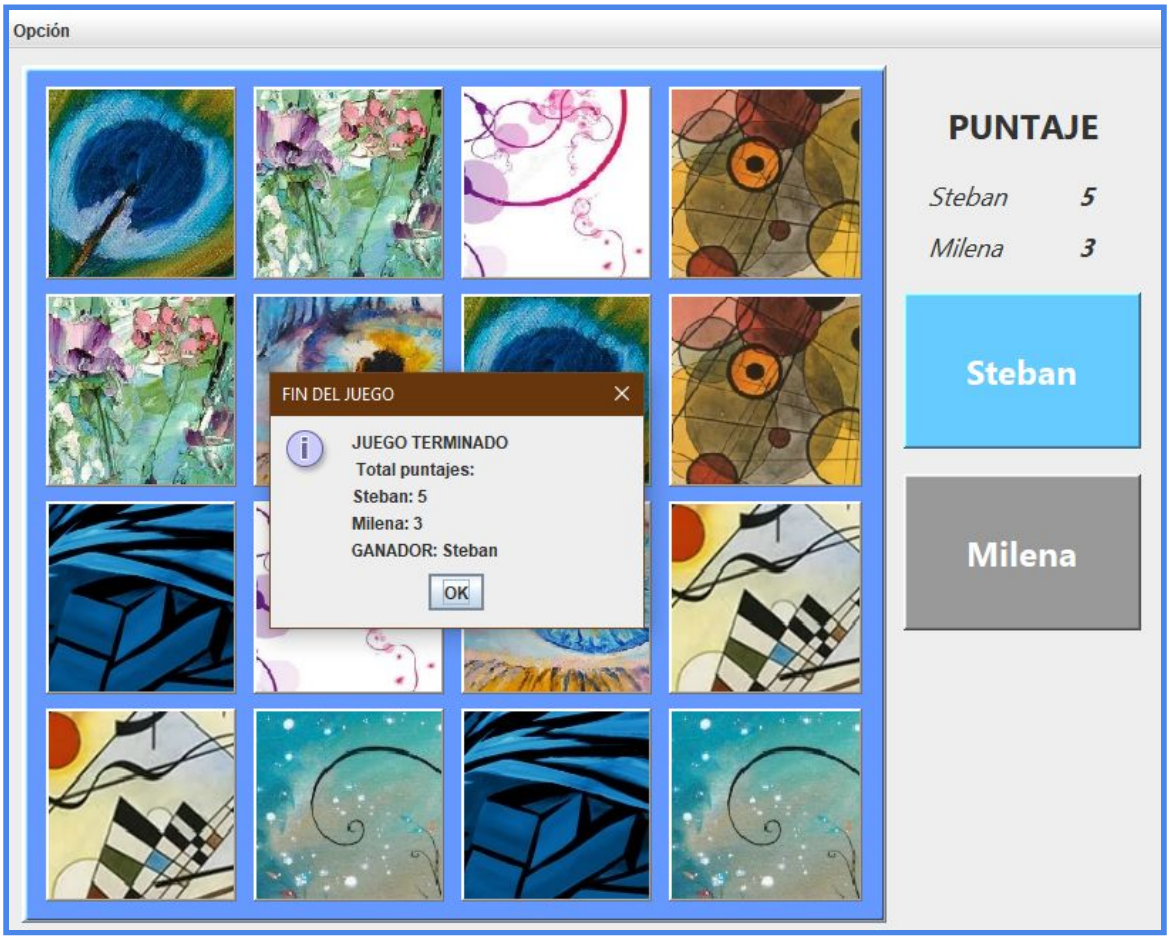
- Desarrollar un programa autónomo en la generación de sistemas aleatorios 3x3, 4x4 y 5x5 que se evidencian según la dificultad del programa.

- Implementar un sistema simple e intuitivo que conceda el juego cooperativo de dos jugadores, además de un sistema de puntaje completo y sencillo.

- Garantizar posibilidades en selección y funcionamiento que causen la customización de acuerdo a los gustos de los usuarios.

EXPLICACIÓN DEL JUEGO

Concentrate PINTCON consiste en un juego interactivo y cooperativo mediante el cual mediante la selección de dificultad variante en el tablero, implica encontrar las parejas de las cartas, sumar puntaje y una competitividad de ambos jugadores por tener más puntos.

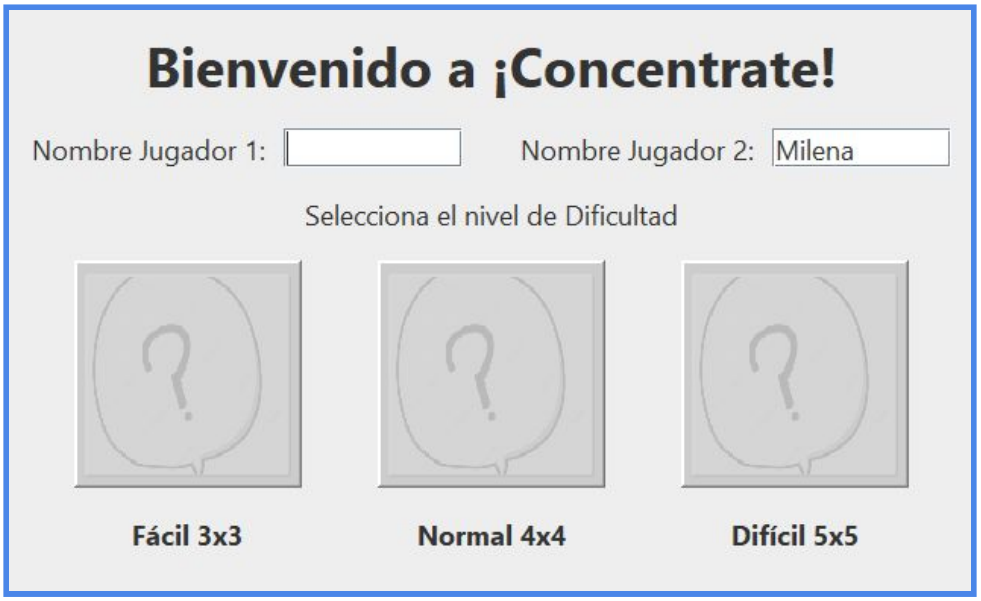


Reglas

- El turno de cada jugador dependerá de si su contrario, se equivoca al buscar las parejas de las cartas.
- Si un jugador encuentra una pareja se sumará un punto en el sistema de Puntaje y puede tener otra posibilidad de elección de pares.
- Si se encuentran todas las parejas, el jugador ganador será quien tenga más puntos en el juego.
- Las reglas del juego cambian al seleccionar la dificultad del tablero.
- La máxima cantidad de puntos y de parejas posibles depende de la dificultad del tablero.

REQUERIMIENTOS PRE-PARTIDA

- Iniciar el ejecutable .java del programa.
- Es necesario que ambos jugadores digiten su nombre en los campos de texto para habilitar los botones de dificultad del menú principal.



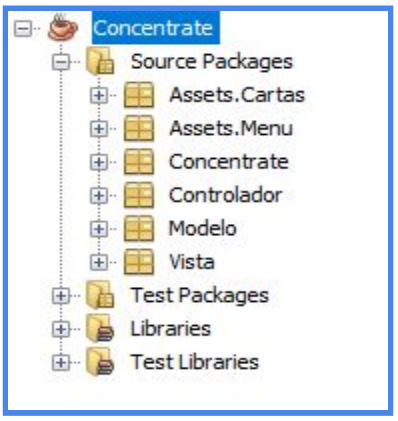
- Es necesario a su vez, que los jugadores seleccionen una dificultad del tablero para la partida.

REQUERIMIENTOS POST-PARTIDA

- Si acaba la partida y desean reiniciarla, deberán dentro del menú opción seleccionar nuevo juego, para reiniciar el sistema.
- Si desean cambiar la dificultad, deberán dentro del menú opción, clicar en seleccionar dificultad para mostrar el menú principal nuevamente.
- Si desean salir del programa, deberán dentro del menú opción, seleccionar salir para cerrar la ventana, o mismamente cerrar la ventana

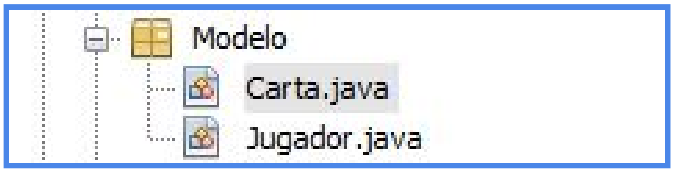
ABSTRACCIÓN DEL PROYECTO

Para comenzar con la abstracción de nuestro juego, primeramente implementamos el patrón Modelo Vista Controlador (MVC), utilizando cada una de sus componentes para el desarrollo del mismo. Así entonces, se procederá explicando cada uno de estos componentes del patrón con sus respectivos elementos.



MODELO

En el paquete Modelo implementamos dos clases, teniendo en cuenta que van existir dos jugadores, cada uno de estos con unos datos específicos, y, asimismo, existirán dos cartas las cuales son las que escoge nuestro jugador, llegando a la conclusión de crear las siguientes dos clases.



- **Clase Jugador**: En esta clase se llevará el control de los datos de nuestro jugador, cada vez que se ingrese se crearán dos objetos correspondientes a la clase.

Entre sus atributos y métodos contamos con:

- **nombre:** String donde se registra el nombre del jugador.

- **turno:** Boolean que llevará el control del turno del jugador, cuando este se encuentre en juego, su valor será True, de lo contrario será False. Esta variable también nos ayuda a determinar a qué jugador sumarle el punto si este acierta.
- **puntaje:** Int el cual será un acumulador que irá aumentando cuando el jugador tenga un acierto, a la hora de escoger las cartas.

- **métodos asociados:** se comportan y se clasifican de acuerdo a **Getters** y **Setters** de los atributos de tipo privado de la Clase. Aquí observamos algunos de ellos:

```
public boolean isTurno() { //Retorna el Turno del Jugador como un Getter de Boolean
    return turno;
}

public void setTurno(boolean turno) { //Captura el turno del Jugador
    this.turno = turno;
}

public int getPuntaje() { //Retorna el Puntaje del Jugador
    return puntaje;
}

public void setPuntaje(int puntaje) { //Captura el Puntaje del Jugador
    this.puntaje = puntaje;
}
```

- **constructor():** predefinidamente cuando crea un nuevo objeto de la clase inicializa el turno como false, puntaje en cero y el nombre como Jugador.

```
private boolean turno;
private int puntaje;
private String nombre;

public Jugador(){
    this.turno = false;
    this.puntaje = 0;
    this.nombre = "Jugador";
}
```

- **Clase Carta:** En esta clase manejaremos todos lo atributos de los botones que en este caso son una concepción de Cartas, ya que serán las que visualizan nuestro jugador. Cada vez que se ingrese se crearán dos objetos correspondientes a la clase que con implementación de controladores, se usan para manejar y comparar si son las instancias adecuadas. Entre sus atributos y métodos contamos con:

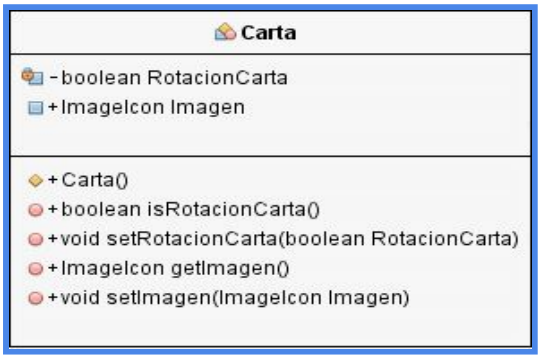
  - **imagen**: ImageIcon, que guarda la imagen de la carta, será un atributo para comparar si las imágenes entre cartas son iguales.

  - **rotación**: Boolean, que controlará que cuando el botón haya sido escogido, su rotación tenga el valor de True, de lo contrario un False, para luego comparar entre las instancias de las cartas.

  - **métodos asociados:** se comportan y se clasifican de acuerdo a **Getters** y **Setters** de los atributos de tipo privado de la Clase.

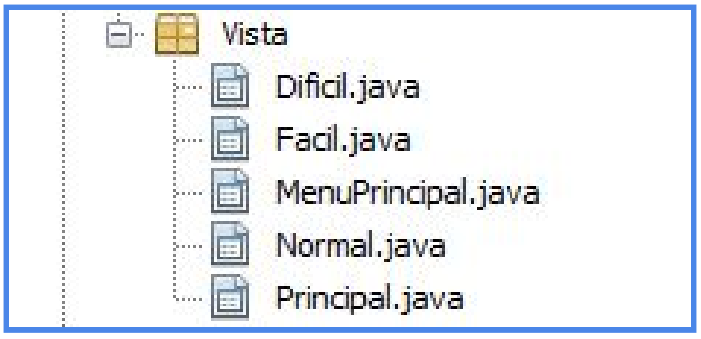
  - **constructor():** predefinidamente cuando crea un nuevo objeto de la clase, inicializa rotacion como false e imagen como nulo.

```
private boolean rotacion;
private ImageIcon imagen;

public Carta(){
    this.rotacion = false;
    this.imagen = null;
}
```



VISTA

En el paquete Vista implementamos cinco clases, teniendo en cuenta que van existir tres niveles de dificultad, cada uno de estos con tableros específicos, asimismo, existirán dos contenedores, uno que guarde los tableros de las dificultades con otros componentes, y otro que permite dar la elección de las dificultades como capturar los nombres de los jugadores, llegando a la conclusión de crear las siguientes cinco clases.



TABLEROS DE DIFICULTAD

Los diversos tableros de dificultad establecidos como Fácil, Normal y Difícil, comparten atributos y métodos de acuerdo a sus específicas clases, pero cambian mínimamente en cuanto al comportamiento y lógica prestada para los controladores. Así entonces dentro de las similitudes encontramos:

- **setCartas()**: Este método no retorna valores siendo entonces tipo void y no recibe parámetros. Dentro de este contaremos un vector de tipo entero (int); el cual guardará los números aleatorios que genera el Controlador Lógica. Se utilizará la clase ImageIcon, la cual nos otorgara distintos métodos para poder acomodar la imagen en el JButton. Por consiguiente, llamaremos el método setDisabledIcon el cual hace parte de la clase JButton; recibe como parámetro una variable de tipo ImageIcon y tiene como funcionalidad establecer una ilustración en el JButton. Dentro de este método creamos una nueva ImageIcon, la cual la capturará correspondientemente al valor de la posición del vector.

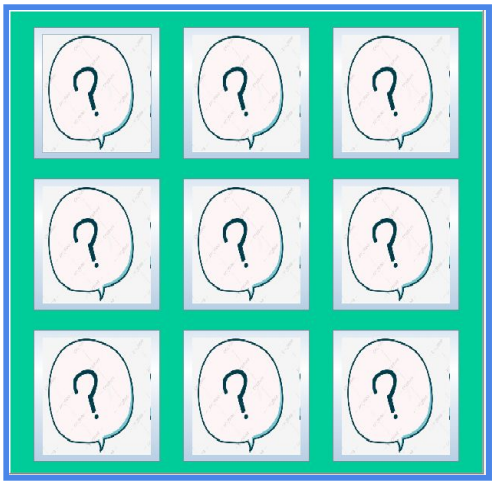
- **JButtonActionPerformed(java.awt.event.ActionEvent evt):** Para controlar cuando se presente una acción en Jbuttons, utilizaremos el método de JButton ActionPerformed, la cual se ejecutará cuando se le de click al botón, consecuentemente, llamaremos una función de nuestro ControladorValidar, llamada: datosBoton(JButton), la cual recibe como parámetro el JButton que presentó la acción, siendo implementando para todos los JButton y el mismo para cualquier dificultad.

```
private void btnf1ActionPerformed(java.awt.event.ActionEvent evt) {
    InstanciaValidar.datosBoton(btnf1);
}
```

- **JButtonMouseExited(java.awt.event.MouseEvent evt):** Se implementa este método a cada uno de los JButton, para que justo en el momento donde el mouse sale del componente accionado, llame a una de las funciones del ControladorValidar, la cual decide si inhabilitar los botones o dejarlos en su estado normal, enviando el nivel de dificultad desde el cual se llama la función. Este evento es aplicado a todos los JButton.

```
private void btnf1MouseExited(java.awt.event.MouseEvent evt) {
    InstanciaValidar.comparar(1);
}
```

- **Clase Fácil:** En esta clase dentro de la interpretación de Vista, hablamos de un tablero con 9 botones que implementan dentro de su comportamiento el manejo de Controladores. Dentro de la clase se crean dos instancias correspondientes a los controladores de la misma. Esta clase corresponde a un nivel de dificultad Fácil.



El tablero contiene diferentes métodos, los cuales son encargados de diferentes acciones que los JButton puedan tener, en este casos se recurre a los controladores para su desarrollo óptimo. Estos métodos ya fueron descritos, pero en cuanto a sus particularidades son los siguientes:

- **setCartas()**: En éste caso, cambia el tamaño de nuestro vector a 9 posiciones, por esto mismo se le asignan imágenes a 9 JButtons y se le envía a la Instancia Lógica el valor de 1 para identificar la dificultad Fácil.

```
public void setCartas(){ //Captura las imagenes en cada Carta

    //Genera un vector de Id de imagenes aleatorio de dificultad fácil
    int[] num= InstanciaLogica.getCartas(1);

    /*El nombre de las imagenes tiene la estructura de img+numero+.jpg
      con ello el vector num que tiene números aleatorios con las posiciones del tablero
      los asigna con getResource concatenandose en el string asignado
    */

    //Guarda las imagenes de la primera fila del Tablero
    btnf1.setDisabledIcon(new ImageIcon(getClass().getResource("../Assets/Cartas/img"+num[0]+".jpg")));
    btnf2.setDisabledIcon(new ImageIcon(getClass().getResource("../Assets/Cartas/img"+num[1]+".jpg")));
    btnf3.setDisabledIcon(new ImageIcon(getClass().getResource("../Assets/Cartas/img"+num[2]+".jpg")));
```

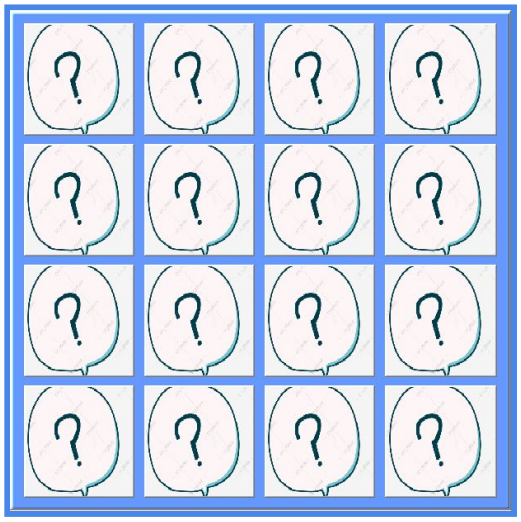
- **JButtonMouseExited(java.awt.event.MouseEvent evt):** Se implementa este método a cada uno de los JButton, comparar() recibe como parámetro 1 para identificar que la dificultad es Fácil.

```
private void btnf1MouseExited(java.awt.event.MouseEvent evt) {
    InstanciaValidar.comparar(1);
}
```

- **Constructor:** Inicializa los componentes Swing con el método initComponents(), y también llama al método setCartas() cuando se instancia un objeto de su clase.

```
//Instancia los Controladores
public ControladorLogica InstanciaLogica= new ControladorLogica();
public ControladorValidar InstanciaValidar= new ControladorValidar();

public Facil() { //Constructor de la Clase
    initComponents(); //Inicializa los componentes Swing
    setCartas(); //Ejecuta el método
}
```

- **Clase Normal:** En esta clase dentro de la interpretación de Vista, hablamos de un tablero con 16 botones que implementan dentro de su comportamiento el manejo de Controladores. Dentro de la clase se crean dos instancias correspondientes a los controladores de la misma. Esta clase corresponde a un nivel de dificultad Normal.



En esta clase también se cuenta con determinados métodos los cuales ya fueron definidos en la clase Fácil, estos métodos son los encargados de las acciones de los JButton, los cuales recurren a los Controladores. Mencionaremos cada método con las diferencias de la clase Fácil.

- **setCartas():** En éste caso, a diferencia del tablero Fácil, cambia el tamaño de nuestro vector a 16 posiciones, por esto mismo se le asigna imágenes a 16 JButtons y se le envía a la Instancia Lógica el valor de 2 para identificar la dificultad Normal.

```
public void setCartas(){  //Captura las imagenes en cada Carta

    //Genera un vector de Id de imagenes aleatorio de dificultad normal
    int[] numbers = InstanciaLogica.getCartas(2);

    /*El nombre de las imagenes tiene la estructura de img+numero+.jpg
      con ello el vector num que tiene números aleatorios con las posiciones del tablero
      los asigna con getResource concatenandose en el string asignado
    */

    //Guarda las imagenes de la primera fila del Tablero
    btn1.setDisabledIcon(new ImageIcon(getClass().getResource("../Assets/Cartas/img"+numbers[0]+".jpg")));
    btn2.setDisabledIcon(new ImageIcon(getClass().getResource("../Assets/Cartas/img"+numbers[1]+".jpg")));
    btn3.setDisabledIcon(new ImageIcon(getClass().getResource("../Assets/Cartas/img"+numbers[2]+".jpg")));
    btn4.setDisabledIcon(new ImageIcon(getClass().getResource("../Assets/Cartas/img"+numbers[3]+".jpg")));
```

- **JButtonMouseExited(java.awt.event.MouseEvent evt):** Se implementa este método a cada uno de los JButton, comparar() recibe como parámetro 2 para identificar que la dificultad es Normal.

```
private void btn1MouseExited(java.awt.event.MouseEvent evt) {
    InstanciaValidar.comparar(2);
}
```

- **Constructor:** Inicializa los componentes Swing con el método initComponents(), y también llama al método setCartas() cuando se instancia un objeto de su clase.

```
//Instancia los Controladores
public ControladorLogica InstanciaLogica= new ControladorLogica();
public ControladorValidar InstanciaValidar= new ControladorValidar();


public Normal() { //Constructor de la Clase
    initComponents(); //Inicializa los componentes Swing
    setCartas(); //Ejecuta el método
}
```

- **Clase Difícil:** En esta clase dentro de la interpretación de Vista, hablamos de un tablero con 25 botones que implementan dentro de su comportamiento el manejo de Controladores. Dentro de la clase se crean dos instancias correspondientes a los controladores de la misma. Esta clase corresponde a un nivel de dificultad Difícil.

En esta clase también se cuenta con determinados métodos los cuales ya fueron definidos en la clase Fácil, estos métodos son los encargados de las acciones de los JButton, los cuales recurren a los Controladores. Mencionaremos cada método con las diferencias de la clase Difícil.

- **setCartas()**: En éste caso, cambia el tamaño de nuestro vector a 25 posiciones, por esto mismo se le asignan imágenes a 25 JButtons y se le envía a la Instancia Lógica el valor de 3 para identificar la dificultad Difícil.

```
public void setCartas(){ //Captura las imagenes en cada Carta

    //Genera un vector de Id de imagenes aleatorio de dificultad dificil
    int[] num= InstanciaLogica.getCartas(3);

    /* El nombre de las imagenes tiene la estructura de img+numero+.jpg
     con ello el vector num que tiene números aleatorios con las posiciones del tablero
     los asigna con getResource concatenandose en el string asignado
    */

    //Guarda las imagenes de la primera fila del Tablero
    btnDificil1.setDisabledIcon(new ImageIcon(getClass().getResource("../Assets/Cartas/img"+num[0]+".jpg")));
    btnDificil2.setDisabledIcon(new ImageIcon(getClass().getResource("../Assets/Cartas/img"+num[1]+".jpg")));
    btnDificil3.setDisabledIcon(new ImageIcon(getClass().getResource("../Assets/Cartas/img"+num[2]+".jpg")));
    btnDificil4.setDisabledIcon(new ImageIcon(getClass().getResource("../Assets/Cartas/img"+num[3]+".jpg")));
    btnDificil5.setDisabledIcon(new ImageIcon(getClass().getResource("../Assets/Cartas/img"+num[4]+".jpg")));
```

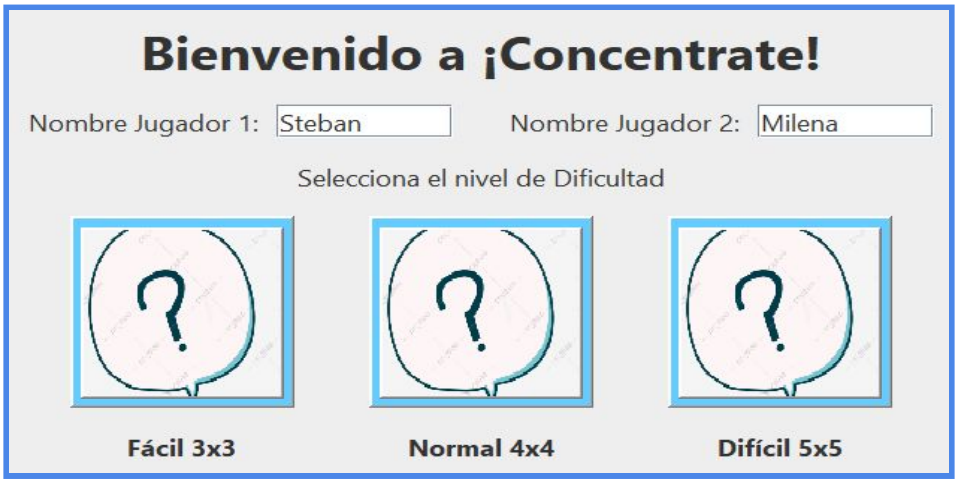
- **JButtonMouseExited(java.awt.event.MouseEvent evt):** Se implementa este método a cada uno de los JButton, comparar() recibe como parámetro 3 para identificar que la dificultad es Difícil.

```
private void btnDificil6MouseExited(java.awt.event.MouseEvent evt) {
    InstanciaValidar.comparar(3);
}
```

- **Constructor:** Inicializa los componentes Swing con el método initComponents(), y también llama al método setCartas() cuando se instancia un objeto de su clase.

```
//Instancia los Controladores
public ControladorLogica InstanciaLogica= new ControladorLogica();
public ControladorValidar InstanciaValidar= new ControladorValidar();


public Dificil() { //Constructor de la Clase
    initComponents(); //Inicializa los componentes Swing
    setCartas(); //Ejecuta el método
}
```

- **Clase Menú Principal:** En esta clase dentro de la interpretación de Vista, hablamos de un tablero, el cual se visualiza al iniciar nuestro juego. Este tablero está compuesto por 8 JLabel (Titulo, Nombres, Selección, y para nombrar cada dificultad), 2JTextField (Nombre Jugador 1, Nombre Jugador 2) y 3 JButton (Fácil, Normal, Difícil). Cuando se ingresen los nombres automáticamente se activarán los botones para que los jugadores seleccionen la dificultad deseada, de lo contrario estos permanecerán inhabilitados.



En esta clase contamos con diferentes atributos y métodos, lo cuales se encargan de el control, del nombre del jugador, las acciones correspondientes a los JButton. Así que encontraremos:

- **campo1:** Boolean, que determina si en el JTextField 1 ya fue llenado con el nombre del primer jugador, si esto se cumple este JTextField será True, de lo contrario False.

- **campo2:** Boolean, determina si en el JTextField 2 ya fue llenado con el nombre el segundo jugador, si esto se cumple este JTextField será True, de lo contrario False

  .
- **constructor():** Inicializa los componentes Swing con el método initComponents(); predeterminada: los botones en su parte setEnabled() como false para que estén inhabilitados, iniciliza campo1 y campo2 en false.

```
public MenuPrincipal() {
    initComponents();
    btnFacil.setEnabled(false);
    btnNormal.setEnabled(false);
    btnDificil.setEnabled(false);
    Campo1 = false;
    Campo2 = false;
}
```

- **validarCampos():** En este método a través de un condicional (if), se evalúa si campo1 y campo2 son verdaderos (true) o sea si ya fueron ingresados los nombres de los jugadores, si es asi los botones Fácil, Normal y Difícil se habilitarán, de lo contrario se inhabilitaran hasta que los nombres sean ingresados.

```
public void validarCampos(){
    if(Campo1 && Campo2 == true){ //Si hay datos en ambos Text Fields
        //Se habilitan los botones de Dificultad
        btnFacil.setEnabled(true);
        btnNormal.setEnabled(true);
        btnDificil.setEnabled(true);
    }else{ //De lo Contrario
        //Se inhabilitan los botones de Dificultad
        btnFacil.setEnabled(false);
        btnNormal.setEnabled(false);
        btnDificil.setEnabled(false);
    }
}
```

- **JButtonActionPerformed(java.awt.event.ActionEvent evt):** Para controlar cuando se presente una acción en los Jbuttons de dificultad, utilizaremos el método de JButton ActionPerformed, la cual se ejecutará cuando se le de click al botón de cada dificultad, consecuentemente se cerrará la ventana de MenuPrincipal, se creará la ventana de principal enviando a su constructor el número correspondiente, si es: Dificil este será 2, si es Normal este será 1 y si es facil este será 0. También se establecerá los nombres de los jugadores en los JLabel y JButton establecidos para cada Jugador en el tablero Principal. Por último se mostrará la ventana Principal. Para Fácil sería:

```
private void btnFacilActionPerformed(java.awt.event.ActionEvent evt) {

    InstanciaEjecutor.setVisible(false); //Se deja de mostrar el Menu Principal

    InstanciaPrincipal = new Principal(0); //Se instancia un nuevo objeto Principal de dificultad Facil

    //Se coloca en los labels de Puntaje los nombres de los Jugadores
    InstanciaPrincipal.getLblNombre1().setText(InstanciaEjecutor.getTxtNombre1().getText());
    InstanciaPrincipal.getLblNombre2().setText(InstanciaEjecutor.getTxtNombre2().getText());

    //Se coloca en los botones del turno los nombres de los Jugadores
    InstanciaPrincipal.getBtnJugador1().setText(InstanciaEjecutor.getTxtNombre1().getText());
    InstanciaPrincipal.getBtnJugador2().setText(InstanciaEjecutor.getTxtNombre2().getText());

    InstanciaPrincipal.setVisible(true); //Se hace visible la vista Principal instanciada
}
```

- **JTextFieldKeyReleased(java.awt.event.KeyEvent evt):** Por medio de este evento se puede establecer cuando se escribe en el JTextField, cuando pase esto campo1 y campo2 serán true, de lo contrario éste será false. Para campo1 sería:

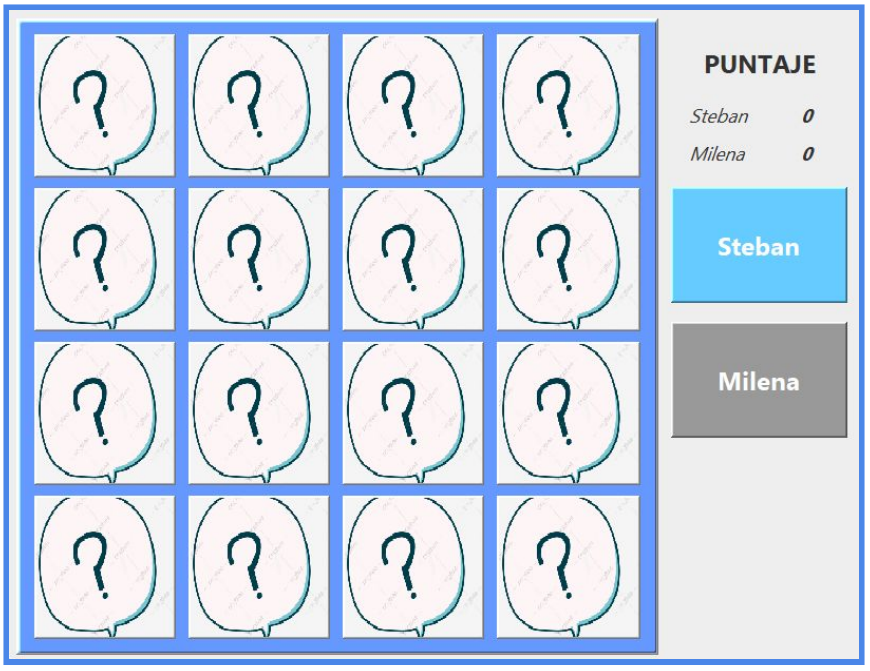
```
private void txtNombre1KeyReleased(java.awt.event.KeyEvent evt) {
    if(!"".equals(txtNombre1.getText())){ //Si hay texto en el Text Field del nombre del Jugador1
        Campo1 = true; //Existen datos en el Text Field
    }else{ //De lo contrario
        Campo1 = false; //No existen datos en el Text Field
    }
    //Valida que hayan datos en los Text Fields de los nombres y habilita los botones de dificultad
    validarCampos();
}
```

- **Métodos asociados:** Por último, se observarán los métodos Setter y Getter, los cuales permiten acceso a las variables privadas.

```
public JTextField getTxtNombre1() { //Retorna el texto del Text Field del Jugador1
    return txtNombre1;
}

public void setTxtNombre1(JTextField txtNombre1) { //Captura el texto del Text Field del Jugador1
    this.txtNombre1 = txtNombre1;
}

public JTextField getTxtNombre2() { //Retorna el texto del Text Field del Jugador2
    return txtNombre2;
}

public void setTxtNombre2(JTextField txtNombre1) { //Captura el texto del Text Field del Jugador2
    this.txtNombre1 = txtNombre1;
}
```

- **Clase Principal:** En esta clase dentro de la interpretación de Vista, hablamos de un tablero, el cual se visualiza al indicar la dificultad deseada. Este Jframe cuenta con 4 JLabel (Titulo Puntaje, Nombre jugador 1, Puntaje jugador 1,

puntaje jugador 2, Nombre Jugador 2), 2 JButton (Turno Jugador 1, Turno Jugador 2) y un JPanel, vacío en el cual serán pintados los tableros de dificultad, Cuenta con un JMenu (Opción), el cual dentro de él contiene 3 JMenuItem (Nuevo juego, Seleccionar Dificultad y Salir).



En esta clase contamos con diferentes atributos y métodos, lo cuales se encargan tanto del JPanel, como del control, del nombre del jugador, las acciones correspondientes a los JButton y a el menú. Los cuales son:

- **InstanciaPrincipal:** Principal, es un objeto de la clase principal el nos sirve para abrir otra ventana del JFrame.

- **Dificultad:** int, en este se guardará la dificultad seleccionada y ayudará a determinar que JPanel se debe pintar en Principal.

- **Constructor:** Inicializa los componentes Swing con el método initComponents(); Se inicializa el método del JFrame setLayout(), creando dentro del él un nuevo BorderLayout, que establecerá un diseño para un contenedor organizado y redimensionando sus componentes.
  Dentro de este constructor existe un Switch, el cual evalúa la dificultad seleccionada. Dependiendo ésta se crea un tablero tipo Fácil, Difícil o Normal, y en el JPanel ya establecido en Principal se llama al método

add(Tablero,BorderLayout.CENTER), al cual se le mandará el tablero según la dificulta y este lo pintara sobre el JPanel.

```
public Principal(int dificultad) {
    initComponents();
    jpnTablero.setVisible(true); //Hace visible el tablero incorporado en Principal
    //Necesario para incorporar en el panel el tablero el nivel de dificultad
    jpnTablero.setLayout(new BorderLayout());
    switch(dificultad){
        case 0:{ //Dificultad Fácil
            Facil Tablero = new Facil(); //Genera un nuevo Tablero Fácil
            jpnTablero.add(Tablero,BorderLayout.CENTER); //Adiciona el Tablero Facil en el panel de Principal
            nivelDificultad=0; //Guarda la dificultad globalmente
            break;
        }
        case 1:{ //Dificultad Normal
            Normal Tablero = new Normal(); //Genera un nuevo Tablero Normal
            jpnTablero.add(Tablero,BorderLayout.CENTER);  //Adiciona el Tablero Normal en el panel de Principal
            nivelDificultad=1; //Guarda la dificultad globalmente
            break;
        }
        case 2:{ //Dificultad Dificil
            Dificil Tablero = new Dificil(); //Genera un nuevo Tablero Normal
            jpnTablero.add(Tablero,BorderLayout.CENTER); //Adiciona el Tablero Normal en el panel de Principal
            nivelDificultad=2; //Guarda la dificultad globalmente
            break;
        }
```

- **setJLabel(int Puntaje):** Recibe como parámetro el puntaje del jugador, y dentro de este método a través de la función ValueOf de String se convierte a un String y se le pasa Al JLabel en su parte SetText(), para que muestre en Principal los cambios del puntaje.

```
public void setLblPuntaje1(int Puntaje1) { //Actualiza la información del Puntaje del Jugador 1
    String Puntaje = String.valueOf(Puntaje1); //Convierte el entero en string
    this.lblPuntaje1.setText(Puntaje); //Guarda el texto
}

public void setLblPuntaje2(int Puntaje2) { //Actualiza la información del Puntaje del Jugador 2
    String Puntaje = String.valueOf(Puntaje2); //Convierte el entero en string
    this.lblPuntaje2.setText(Puntaje); //Guarda el texto
}
```

- **activarBtnJugador(boolean Jugador):** Recibe como parámetro un Boolean el cual nos indica si es verdadero o falso el turno del jugador, dentro de este Método existe un condicional (if) el cual pregunta si Jugador es False, el color de botón del Jugador se tornara gris, si no es así el color será Azul.

```
public void activarBtnJugador1(boolean Jugador) { //Identifica de si es el Turno del Jugador 1
    if(!Jugador){ //Si no es su turno
        btnJugador1.setBackground(new java.awt.Color(153,153,153)); //Backgorund Gris
    }else{ //Si es su turno
        btnJugador1.setBackground(new java.awt.Color(102,204,255)); //Backgorund Azul
    }
}

public void activarBtnJugador2(boolean Jugador) { //Identifica de si es el Turno del Jugador 2
    if(!Jugador){ //Si no es su turno
        btnJugador2.setBackground(new java.awt.Color(153,153,153)); //Backgorund Gris
    }else{ //Si es su turno
        btnJugador2.setBackground(new java.awt.Color(102,204,255)); //Backgorund Azul
    }
}
```

- **mnJuegoActionPerformed(java.awt.event.ActionEvent evt):** En este método podemos identificar cuando en el ItemMenu de reinicio de Juego tiene una acción, si esto sucede dentro de este método hay un Switch, el cual tiene tres casos, estos tres casos determinan la Dificultad, según esta se cerrara la antigua ventana de principal y después de esto se visualizará una nueva ventana con el JPanel reiniciado según la dificultad que se está jugando.

```
private void mnJuegoActionPerformed(java.awt.event.ActionEvent evt) {
    switch(nivelDificultad){  //Casos de acuerdo a nivelDificultad guardado
        case 0: //Dificultad Fácil
            InstanciaPrincipal.setVisible(false); //Deja de mostrar la InstanciaPrincipal
            InstanciaPrincipal = new Principal(0); //Inicializa una nueva Instancia de dificultad Fácil
            break;
        case 1:  //Dificultad Normal
            InstanciaPrincipal.setVisible(false); //Deja de mostrar la InstanciaPrincipal
            InstanciaPrincipal = new Principal(1); //Inicializa una nueva Instancia de dificultad Normal
            break;
        case 2:  //Dificultad Dificil
            InstanciaPrincipal.setVisible(false); //Deja de mostrar la InstanciaPrincipal
            InstanciaPrincipal = new Principal(2); //Inicializa una nueva Instancia de dificultad Dificultad
            break;
    }

    //Se coloca en los labels de Puntaje los nombres de los Jugadores
    InstanciaPrincipal.getLblNombre1().setText(InstanciaEjecutor.getTxtNombre1().getText());
    InstanciaPrincipal.getLblNombre2().setText(InstanciaEjecutor.getTxtNombre2().getText());

    //Se coloca en los botones del turno los nombres de los Jugadores
    InstanciaPrincipal.getBtnJugador1().setText(InstanciaEjecutor.getTxtNombre1().getText());
    InstanciaPrincipal.getBtnJugador2().setText(InstanciaEjecutor.getTxtNombre2().getText());

    InstanciaPrincipal.setVisible(true); //Se hace visible la vista Principal instanciada
```

- **mnSalirActionPerformed(java.awt.event.ActionEvent evt):** Este método determina cuando se oprima la opción salir, se cerrará la ventana y se dará por finalizado el juego.

```
//Botón del menú para cerrar la ventana
private void mnSalirActionPerformed(java.awt.event.ActionEvent evt) {
    exit(1); //Método para cerrar ventana
}
```

- **mnDificultadActionPerformed(java.awt.event.ActionEvent evt):** En este método se determina la acción del ItemMenu Seleccionar Dificultad, se cerrará la ventana presente de Principal y se visualizará la ventana de MenuPrincipal.

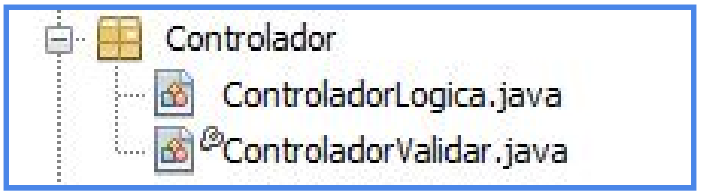
```
//Botón para abrir el menú principal
private void mnDificultadActionPerformed(java.awt.event.ActionEvent evt) {
    InstanciaPrincipal.setVisible(false); //Deja de mostrar la Instancia Principal
    InstanciaEjecutor.setVisible(true); //Muestra la Instancia Ejecutora
}
```

- **Métodos asociados:** Por último, se observarán los métodos Setter y Getter, los cuales permiten acceso a las variables privadas.

```
//Getters de los Botones

public JButton getBtnJugador1() {
    return btnJugador1;
}

public JButton getBtnJugador2() {
    return btnJugador2;
}

//Getters de los Labels

public JLabel getLblNombre1() {
    return lblNombre1;
}

public JLabel getLblNombre2() {
    return lblNombre2;
}
```

CONTROLADOR

En el paquete Controlador implementamos dos clases, teniendo en cuenta que es necesario implementar un controlador que valide comparaciones, puntuaciones, ganadores y el fin del juego,y asimismo, un controlador que genere la lógica de juego dependiendo de las dificultades, los id de las imágenes, y parejas de cartas, llegando a la conclusión de crear las siguientes dos clases.



- **ControladorLógica:** Este controlador asume la complejidad de ser un puente entre los tableros Fácil, Normal o Difícil, hecho con el propósito de generar un lógica aleatoria variable en cuanto la dificultad. En cuanto sus atributos y métodos se encuentran:

    - **cartas:** int[] vector de enteros que se encarga de guardar los id de las cartas generadas en consecuencia de las imágenes establecidas.

    - **imagenes:** int[] vector de enteros que se encarga de guardar los id de las imágenes utilizadas en la dificultad.

    - **contadorCartas:** Int que se encarga de guardar como conteo la posición del momento del vector cartas.

    - **contadorImagen:** Int que se encarga de guardar como conteo la posición del momento del vector imagenes.

    - **numeroAleatorio:** Int que se encarga de guardar un número aleatorio que varía en cuanto a la utilidad de los vectores.

    - **idExist:** Boolean que se encarga de establecer si existe una duplicidad de cartas, o un espacio dentro de los vectores sin datos.

    - **aleatorio:** instancia Random que se encarga de guardar las utilidades de la Clase Random para ser utilizados y generar posibilidades aleatorias.

- **constructor():** predefinidamente cuando crea una nueva instancia de la clase, inicializa los vectores como nulos, contadorCartas como 1, contadorImagen como 0, numeroAleatorio como 0, idExist como false, y aleatorio como una nueva instancia de la clase Random.

```
public ControladorLogica(){
    this.cartas = null;
    this.imagenes = null;
    this.contadorCartas = 1;
    this.contadorImagen = 0;
    this.numeroAleatorio = 0;
    this.idExist = false;
    this.aleatorio = new Random();
}
```

```
public int[] cartas;
public int[] imagenes;
public int contadorCartas;
public int contadorImagen;
public int numeroAleatorio;
public boolean idExist;
public Random aleatorio;
```

- **getCartas(int):** función que comprende todo el funcionamiento del controlador, que recibe como parámetro entero la dificultad del Tablero, se compone de 2 partes asimiladas como **"Ciclo de generación del vector imagenes"**, que dentro de sí contiene una "validación de duplicidad entre un id aleatorio y los espacios del vector imagenes", y un **"Ciclo de generación del vector cartas"**, que dentro de sí contiene una "validación del vector alimentado entre un espacio aleatorio y los espacios sin datos del vector cartas". Visualizando su funcionamiento de una forma más específica y completa, comprendemos así por ejemplo la dificultad Fácil:

- **Primera parte:** inicializamos dependiendo de la dificultad entrante de la función, los dos vectores con espacios variables en cuanto al tablero establecido. Con ello inicializamos un ciclo que varía en cuanto al número de posiciones del vector imagenes, y dentro de sí se valida continuamente, que no existan id´s repetidos. En la validación se genera un id aleatorio de 25 imágenes posibles, y con un nuevo ciclo supervisar que las anteriores posiciones del vector no se hayan generado con el mismo id aleatorio.

```
cartas = new int[9]; //Establece un vector cartas con 9 posiciones que se usan en el tablero
imagenes = new int[5]; //Establece un vector idImagenes con 5 posiciones que se usan en el tablero

for(int i=0;i<5;i++){ //Ciclo que permite generar un vector imagenes aleatorio
    do{
        idExist = false; //Inicialmente no existe una duplicidad
        //Genera un id de imagen aleatorio dentro del limite de la cantidad de imagenes (25)
        numeroAleatorio = aleatorio.nextInt(25)+1;
        for(int j=0;j<5;j++){ //Ciclo que reconoce si hay duplicidad del id generado
            if(imagenes[j] == 0){ //Si encuentra que no hay datos siguientes
                break; //Sale del Ciclo
            }
            if(imagenes[j] == numeroAleatorio){ //Si encuentra alguna duplicidad
                idExist = true; //Reconoce como verdadera la duplicidad
            }
        }
    }while(idExist == true); //Valida si un id generado ya existe en el vector imagenes
    imagenes[i] = numeroAleatorio; //Guarda el id en el vector idImagenes
    System.out.println("id imagen "+i+": "+imagenes[i]);
}
```

- **Segunda parte:** instauramos un ciclo tipo while que permite recorrer la cantidad de espacios del vector cartas que varía de acuerdo a la dificultad, inicializamos el booleano idExist en nulo para cada momento en que comienza de nuevo el ciclo, generamos un número dentro de los límites de la cantidad de espacios del tablero y preguntamos si la posición aleatoria generada en el vector no tiene datos; si no tiene datos, guardamos en la posición aleatoria generada del vector, un id del vector imágenes de acuerdo al contadorImagen del momento, con ello corroboramos que la posición aleatoria no tenía datos y sale de la validación, caso contrario lo hace hasta que encuentre una posición sin datos. Por último si el contadorCartas del momento es par, implica que se ha generado una nueva pareja de cartas y por ende cambia el id de la imagen que guardaremos en el vector.

```
while(contadorCartas <= 9) { //Ciclo que reconoce que se haya generado los espacios del vector cartas
    idExist = false; //Inicialmente no existe en la posición aleatoria algún dato
    do{
        //Genera una posición aleatoria dentro de los limites de la cantidad de cartas del tablero (9)
        numeroAleatorio = aleatorio.nextInt(9);
        if(cartas[numeroAleatorio] == 0){ //Si encuentra que la posición no tiene datos
            System.out.println("posicion: "+numeroAleatorio+" para id:"+imagenes[contadorImagen]);

            //Se guarda en la posición aleatoria el id de la posición del momento del vector idImagenes
            cartas[numeroAleatorio] = imagenes[contadorImagen];

            idExist = true; //Reconoce como verdadero que ya existe en la posición aleatoria el dato.
        }
    //Valida que la posición aleatoria generada se guarde en algún espacio del vector cartas
    }while(idExist == false);
    //Si el contadorCartas es par implica que se han generado una pareja de cartas
    if(contadorCartas % 2 == 0){
        System.out.println("Contador: "+contadorCartas+"\n");
        contadorImagen++; //Se empieza a generar una nueva pareja de cartas
    }
    contadorCartas++; //Se autoincrementa el contador
}
```

- **ControladorValidar:** Este controlador asume la complejidad de ser un puente entre la vista Principal, y el modelo Carta y Jugador, hecho con el propósito de supervisar que se validen comparaciones, puntuaciones, ganadores y el fin del juego teniendo presente la dificultad. En cuanto sus atributos y métodos se encuentran:

    - **jugador1:** objeto Jugador que guarda la información del primer jugador de la partida.
    - **jugador2:** objeto Jugador que guarda la información del segundo jugador de la partida.

    - **carta1:** objeto Carta que guarda la información de la primera carta seleccionada por un jugador.
    - **carta2:** objeto Carta que guarda la información de la segunda carta seleccionada por un jugador.

    - **btnLista:** JButton[], vector de Botones seleccionados.

    - **contadorParejas:** Int que actúa como contador validando que todas las parejas de la dificultad han sido validados

    - **constructor():** predefinidamente cuando crea una nueva instancia de la clase, inicializa los jugadores como nuevos objetos de su clase, guarda la información de los nombres tomados por los Text Field de MenuPrincipal, condicionando de que no sean nulos, inicializa las cartas como nuevos objetos de su clase, inicializa el turno de jugador 1 como True y el del jugador 2 como False, crea una nueva lista de botones de 2 espacios en btnLista e inicializa el contadorParejas en 0.

```
public ControladorValidar() {//Constructor de la Clase

    this.jugador1 = new Jugador();
    if(InstanciaEjecutor != null && InstanciaEjecutor.getTxtNombre1() != null){
        this.jugador1.setNombre(InstanciaEjecutor.getTxtNombre1().getText());
    }
    this.jugador2 = new Jugador();
    if(InstanciaEjecutor != null && InstanciaEjecutor.getTxtNombre2() != null){
        this.jugador2.setNombre(InstanciaEjecutor.getTxtNombre2().getText());
    }
    this.carta1 = new Carta();
    this.carta2 = new Carta();
    this.jugador1.setTurno(true);
    this.jugador2.setTurno(false);
    this.btnLista= new JButton[2];
    this.contadorParejas = 0;
}
```

- **datosBoton():** este método permite verificar el cambio de Imagen y rotación de las cartas con un botón entrante visto en principal.

```
//Permite verificar el cambio de Imagen y rotación de las Cartas con un botón como parámetro entrante
public void datosBoton(JButton btn){
    if(!carta1.isRotacion()) {// Si la primera carta no ha sido volteada
        btn.setEnabled(false);//Se inabilita el botón de la Carta
        carta1.setImagen((ImageIcon) btn.getDisabledIcon());//Se cambia la Imagen de la carta
        btnLista[0] = btn;//Se guarda el botón de la carta en un vector de botones
        carta1.setRotacion(true);//Se vuelve verdadera en la Carta 1 ya que se tienen los datos de la carta
        carta2.setRotacion(false);//Se iguala false en la Carta 2, para poder obtener los datos de la segunda carta
    }else { //De lo contario, implica que la primera carta ha sido volteada
        btn.setEnabled(false);//inhabilita el segundo botón entrante que se escogió
        carta2.setImagen((ImageIcon) btn.getDisabledIcon());//Se cambia la Imagen de la carta
        btnLista[1] = btn;//Se guarda el segundo botón de la carta en un vector de botones
        carta2.setRotacion(true);//Se vuelve verdadera en la Carta 2 para su comparacion.
    }
}
```

- **comparar(int):** esta función de acuerdo al tablero de dificultad verifica que las cartas que han sido seleccionadas sean distintas o similares. Aquí es posible visualizar con comentarios el funcionamiento del método.

```
//Verifica de acuerdo a la dificultad, que las cartas seleccionadas sean distintas o similares
public void comparar(int Dificultad) {
    if (carta1.isRotacion() && carta2.isRotacion()) {//Se pregunta si las dos cartas son verdaderas

        //Compara mediante las imagenes de que ambas sean similares
        if (carta1.getImagen().getDescription().compareTo(carta2.getImagen().getDescription()) != 0) {
            //Vuelven y se habilitan ya que no son las mismas
            btnLista[0].setEnabled(true);
            btnLista[1].setEnabled(true);

            if (jugador1.isTurno()) { //Si es el turno del Jugador1
                jugador1.setTurno(false); //No es el turno del Jugador1
                jugador2.setTurno(true); //Es el turno del Jugador2
            } else { //Si es el turno del Jugador2
                jugador1.setTurno(true); //Es el turno del Jugador1
                jugador2.setTurno(false); //No es el turno del Jugador2
            }

            //Activa backgrounds diferentes para identificar en el tablero el turno del Jugador
            InstanciaPrincipal.activarBtnJugador1(jugador1.isTurno());
            InstanciaPrincipal.activarBtnJugador2(jugador2.isTurno());
```

```
        } else { //Si las cartas son similares

            if (jugador1.isTurno()) { //Si es turno del Jugador1
                //Incremente el Puntaje del Jugador1
                jugador1.setPuntaje( jugador1.getPuntaje() + 1);
                //Actualiza el Puntaje del Jugador1 en el Tablero
                InstanciaPrincipal.setLblPuntaje1(jugador1.getPuntaje());

                System.out.println(jugador1.getNombre()+":" + jugador1.getPuntaje());

            } else { //Si es turno del Jugador2
                //Incremente el Puntaje del Jugador2
                jugador2.setPuntaje(jugador2.getPuntaje() + 1);
                //Actualiza el Puntaje del Jugador2 en el Tablero
                InstanciaPrincipal.setLblPuntaje2(jugador2.getPuntaje());

                System.out.println(jugador2.getNombre()+":" + jugador2.getPuntaje());
            }
            contadorParejas++; //Se incrementa la cantidad de parejas encontradas en el Tablero
        }
        carta1.setRotacion(false); //Primera carta se vuelve false, para empezar otra vez con la validación

        Ganador(); //Verifica el Ganador de la Partida
```

```
        switch(Dificultad){
            case 1: //Dificultad Fácil
                if(contadorParejas==4){ //Si hay 4 parejas encontradas
                    juegoTerminado(); //Ejecutar la función JuegoTerminado
                }
                break;
            case 2: //Dificultad Normal
                if(contadorParejas==8){//Si hay 8 parejas encontradas
                    juegoTerminado(); //Ejecutar la función JuegoTerminado
                }
                break;
            case 3: //Dificultad Difícil
                if(contadorParejas==12){ //Si hay 12 parejas encontradas
                    juegoTerminado(); //Ejecutar la función JuegoTerminado
                }
                break;
        }
    }
```

- **juegoTerminado():** este método imprime en pantalla con un JOption los valores totales de la partida.

```
public void juegoTerminado(){ //Imprime en pantalla los valores
    System.out.println("FINALIZADO");
    if("Empate".equals(Ganador())){ //Si es un empate
        //Imprima en un JoptionPane el total de los puntajes, y que ha sido un empate
        JOptionPane.showMessageDialog(InstanciaPrincipal, "JUEGO TERMINADO \n Total puntajes:
    }else{ //Si no es un empate
        //Imprima en un JoptionPane el total de los puntajes, y el ganador de la partida
        JOptionPane.showMessageDialog(InstanciaPrincipal, "JUEGO TERMINADO \n Total puntajes:
    }
}
```

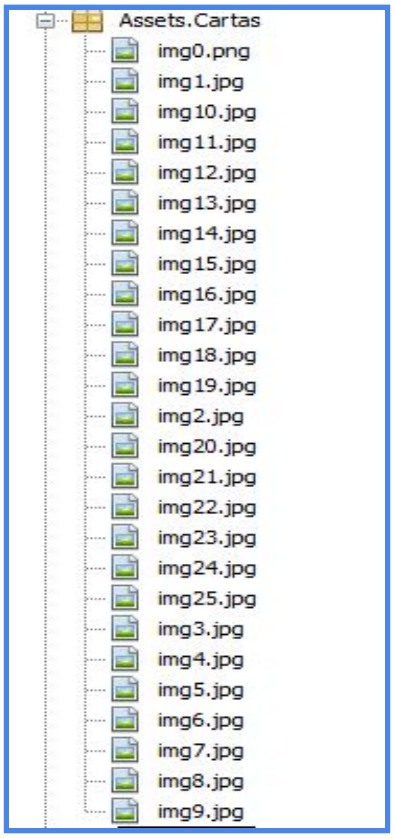
- **ganador():** este método retorna como string el nombre del ganador de la partida o dado caso que haya sido empate

```
public String Ganador(){ //Retorna en string el nombre del ganador o empate
    if(jugador1.getPuntaje()>jugador2.getPuntaje()){ //Si es mayor el puntaje del Jugador1
        return jugador1.getNombre(); //Retorne el nombre del Jugador1
    }
    if(jugador1.getPuntaje()<jugador2.getPuntaje()){  //Si es mayor el puntaje del Jugador2
        return jugador2.getNombre(); //Retorne el nombre del Jugador2
    }
    return "Empate"; //Retorne un empate
}
```

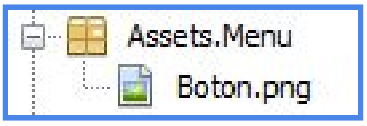
ASSETS

En el paquete Assets implementamos las imágenes las cuales se establecieron en cada uno de nuestros tableros, variando cada una de estas en cada uno de los juegos.

- **Assets.Cartas:** Aqui estaran todas las imágenes utilizadas para nuestros JButton.

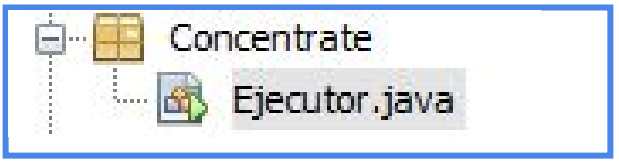


Assets.Menu: Contiene una imagen la cual corresponde al icono de los JButton de Fácil, Normal y Difícil de MenuPrincipal.



CONCÉNTRESE

En este paquete Concéntrese, implementamos nuestra Clase ejecutor. Teniendo en cuenta que es necesario que nuestro programa inicie desde nuestro main, nuestro ejecutor cumple con este papel, teniendo un vínculo con MenuPrincipal para iniciar todo.



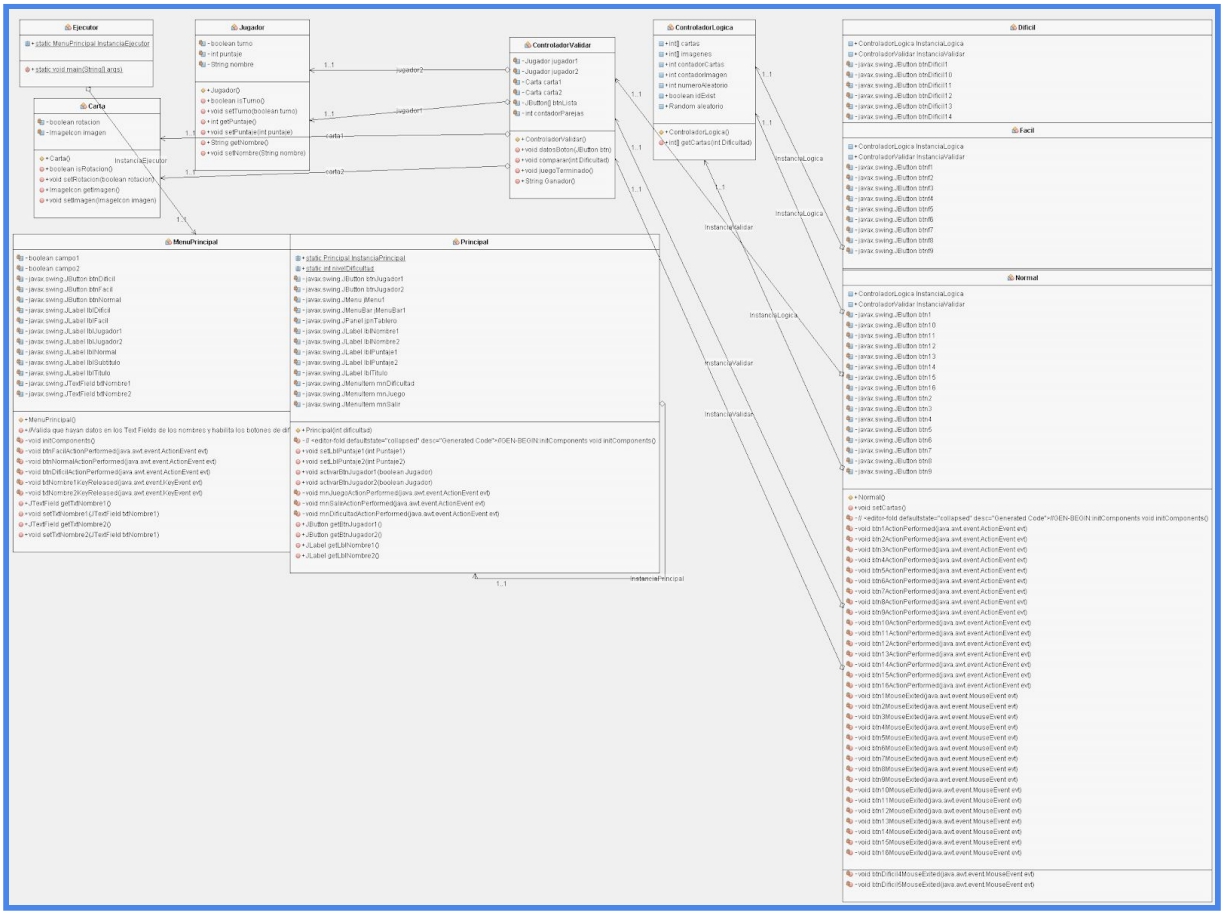
- **Ejecutor:** Es el main de nuestro proyecto, en donde se declara un atributo global el cual será de tipo MenuPrincipal. En el main de clase se creará un nuevo espacio en MenuPrincipal y así se hará la ventana visible a través del método setVisible(true), enviándole un Boolean, el cual hará que nuestra ventana aparezca en pantalla.

```
package Concentrate;

import Vista.MenuPrincipal;

public class Ejecutor {

    public static MenuPrincipal InstanciaEjecutor;

    public static void main(String[] args) {
        InstanciaEjecutor = new MenuPrincipal();
        InstanciaEjecutor.setVisible(true);
    }

}
```

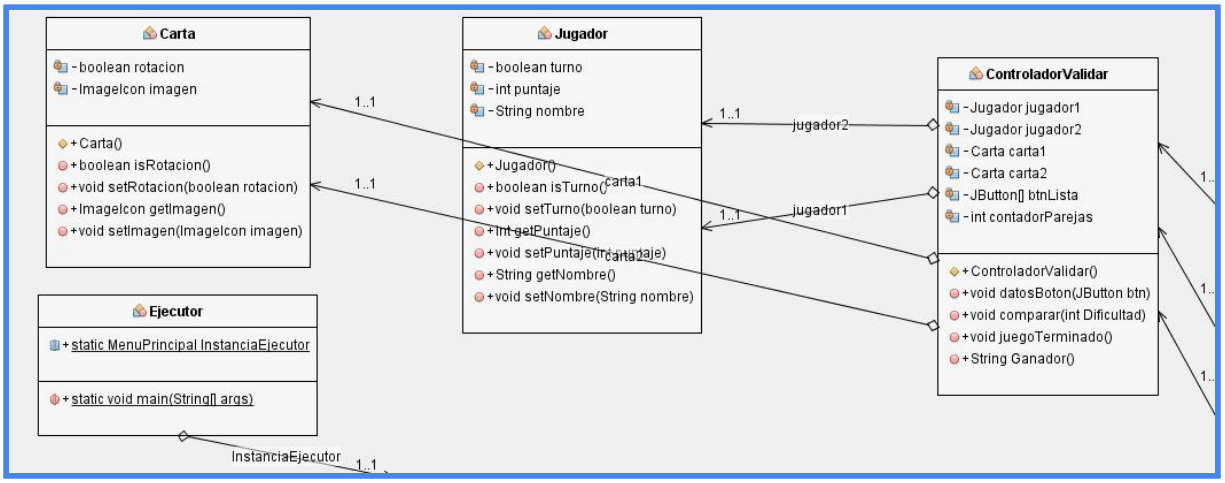
DIAGRAMA DE CLASES UML

El diagrama de clases completo utilizado en este proyecto contemplando tanto las vistas, controladores y modelos generados se comprende de la siguiente forma:

- Las clases Jugador y Carta contemplan una relación con el controladorValidar en cuanto a la generación de Jugador1 y Jugador2, y Carta1 y Carta2.
- Las clases Facil, Normal y Dificil se relacionan con el controladorValidar, al instanciar un objeto de su clase llamado InstanciaValidar
- Las clases Facil, Normal y Dificil se relacionan con el controladorLogica, al instanciar un objeto de su clase llamado InstanciaLogica
- La clase Ejecutor se relaciona con MenuPrincipal, al instanciar un objeto de su clase llamado InstanciaEjecutor.
- La clase MenuPrincipal se relaciona con Principal, al instanciar un objeto de su clase llamado InstanciaPrincipal.
- Las clases Facil,Normal y Dificil se relacionan con Principal, al instanciarse como objetos de su clase en un jpanel dentro de la InstanciaPrincipal.

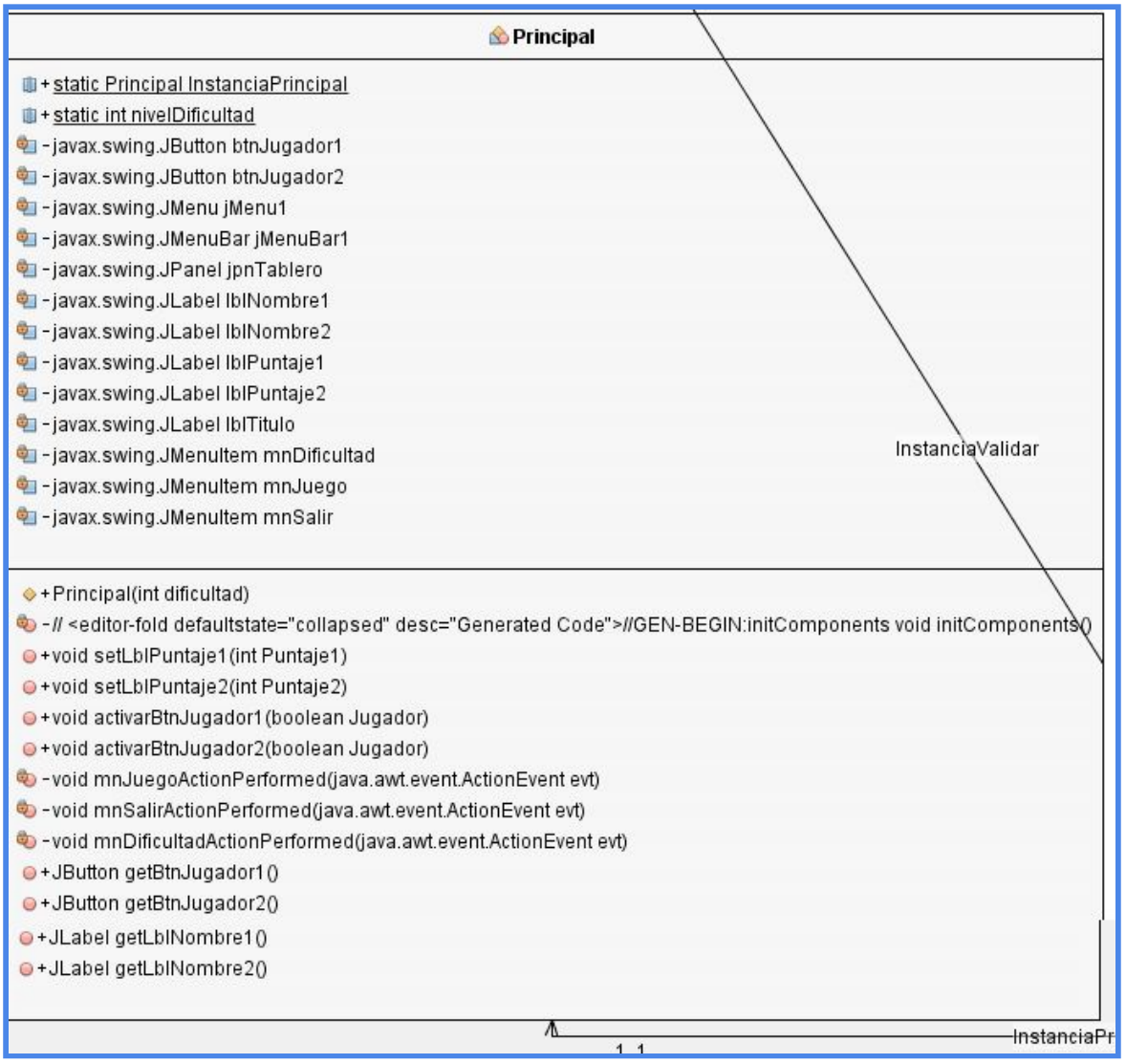


Explicado de una forma sencilla:

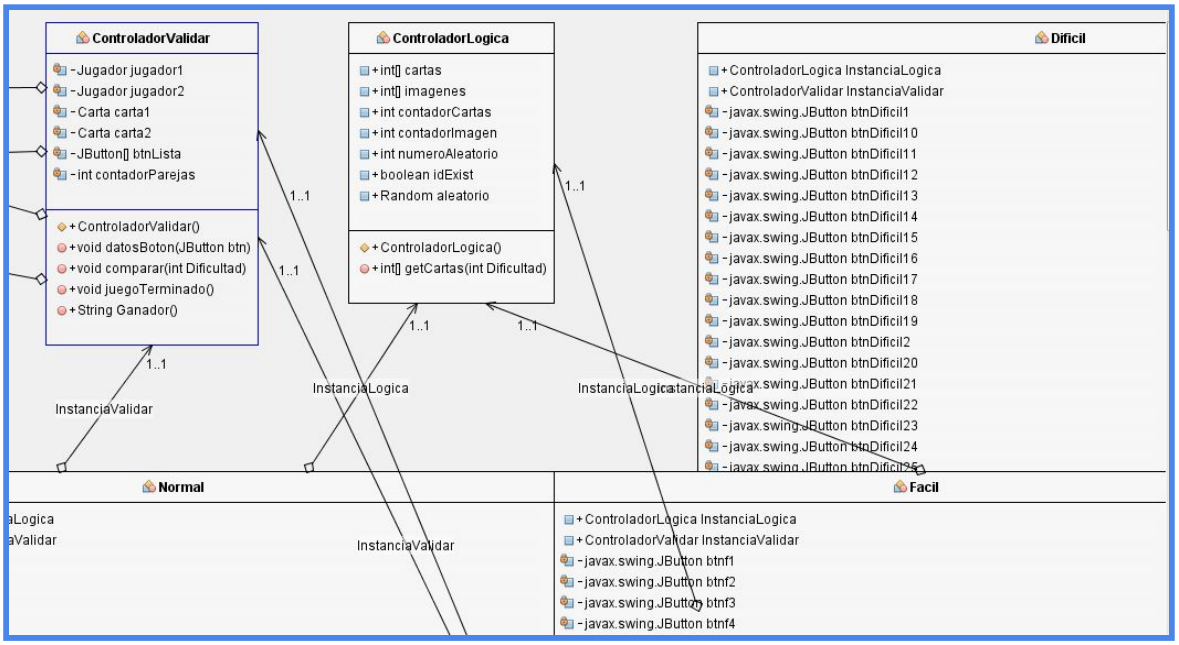


- Aquí observamos los atributos y métodos de la Clase MenuPrincipal

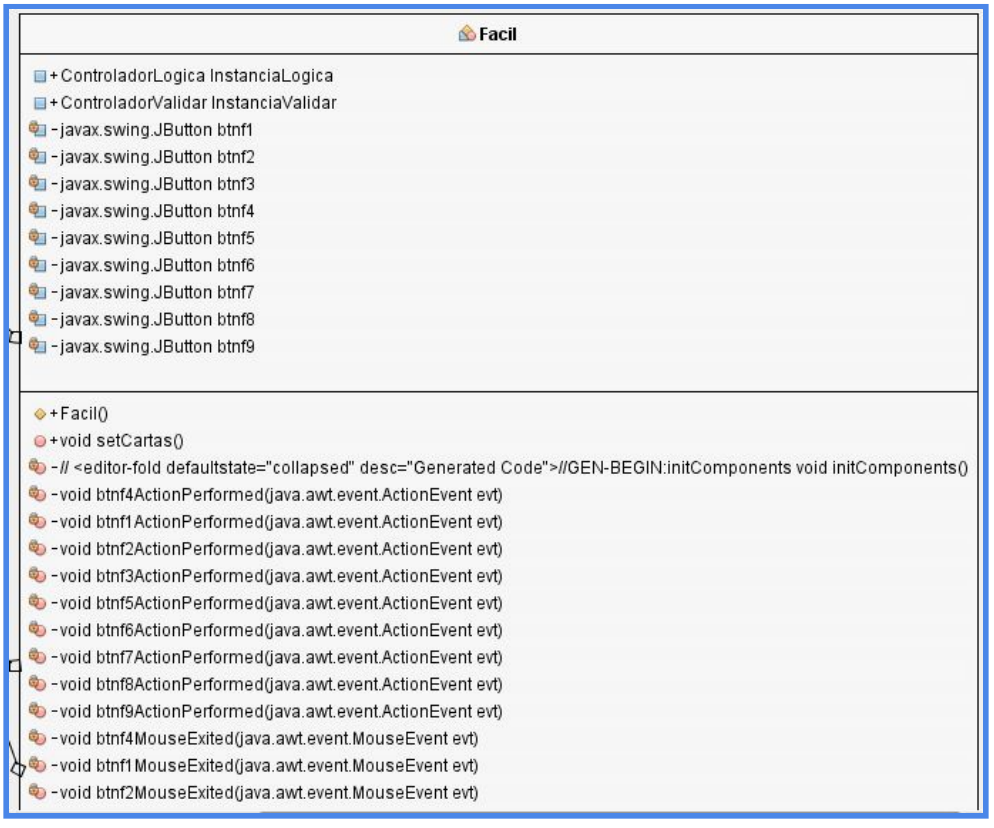
- Aquí observamos los atributos y métodos de la Clase Principal



- Aquí observamos las relaciones de los controladores con los tableros facil, normal y dificil.



- Aquí observamos los atributos y métodos de la Clase Fácil.



- Aquí observamos los atributos y métodos de la Clase Normal.

- Aquí observamos los atributos y métodos de la Clase Dificil simplificada.

CONCLUSIONES

- La abstracción de nuestro juego se hizo a través del Modelo Vista Controlador, donde se implementó la vista de distintas maneras, obteniendo una vista para nuestro menú principal y el inicio de nuestro juego, otra vista Principal para el desarrollo del juego y por último 3 vistas para cada juego que se pueda desarrollar, con una mayor organización.

- La Programación Orientada a Objetos, nos permite el buen manejo de nuestro proyecto y que su implementación sea de manera adecuada al utilizar Java, podemos notar que la buena aplicación de los principios de P.O.O, nos permite utilizar cada clase con total eficacia.

- Adaptar el patrón Modelo Vista Controlador, optimiza y organiza de una forma mucho más eficaz el distanciamiento y jerarquización del código de nuestro proyecto, representando a su vez una asimilación que prevalece en distinguir mediante la abstracción lo que significa ser un modelo (las variables oportunas de una clase distinguida), controlador (los métodos que son un puente entre el modelo y la vista) y una vista (la estética funcional de un programa).

- Diseñar este proyecto nos favoreció a entender, aplicando la teoría, cómo se comporta en la práctica de la programación orientada objetos con MVC a la hora de organizar código de proyectos, y así presentar proyectos de calidad enmarcados a un próximo futuro.