Taller 2 *Fecha: Marzo de 2025*

Indicador de logro a medir: Aplicar los conceptos básicos de la lógica de programación y la orientación a objetos en el desarrollo de una aplicación con interfaz gráfica de usuario basada en el lenguaje ***Java*** y un IDE adecuado.

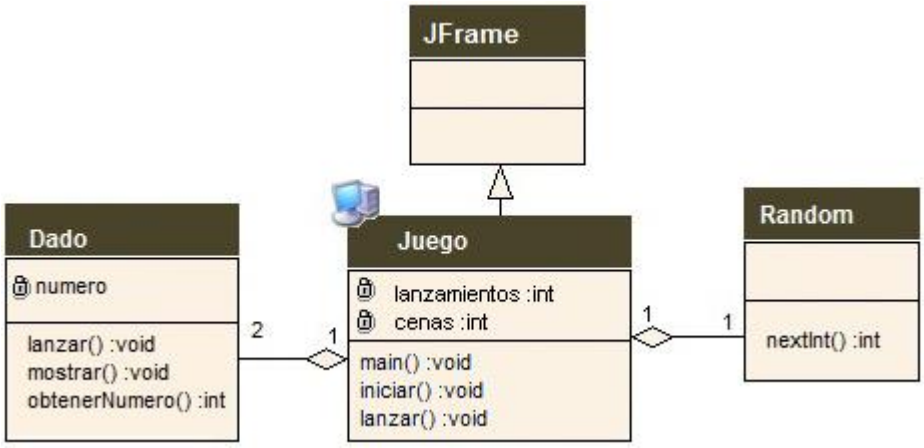
NOTAS:
- Este taller (punto 4) se debe hacer con carácter evaluativo. Representa en total una calificación de 20%.
- Se entregan ejercicios resueltos como ejemplo para el desarrollo de los demás.

Elaborar el diagrama de clases básico (sin las clases correspondientes a la interface de usuario según el lenguaje de implementación) y la respectiva aplicación en un leguaje orientado a objetos para los siguientes enunciados:

1. Un juego de azar corresponde al lanzamiento de 2 dados. De acuerdo a un determinado número de lanzamientos se contabilizan las veces que se obtienen "cenas" (la suma de los dados es 11 ó 12)

R/

- El modelado bajo el paradigma Orientado a Objetos se ilustra en el siguiente diagrama de clases:



En este diagrama es importante comprender lo siguiente:

Una **Clase** se describe normalmente como el molde a partir del cual se crean los objetos. La forma normal de imaginarse las clases es pensando en ellas como la plantilla para hacer un billete, mientras que el objeto propiamente dicho es el billete obtenido con dicha plantilla.

Cuando se crea un objeto a partir de una clase, se dice que el programador ha creado una **Instancia** de dicha clase. Por ejemplo, todos los formularios que se creen en Java son instancias de la clase *JFrame*.

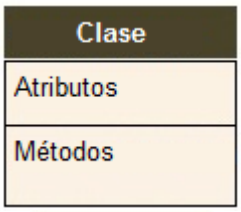
Un **Objeto** es un elemento real o abstracto que tiene un estado y un comportamiento. Un objeto es, pues, una mesa, un alumno, etc., pues son elementos reales y están bien definidos. También lo puede ser un concepto abstracto como un elemento llamado "Lista" que es capaz de recibir un conjunto de números y permitir ordenarlo ascendente o descendentemente.

El **Estado** de un objeto viene determinado por el conjunto de **propiedades** o **atributos** que tiene (Estructura Estática), junto con los valores que pueden asumir cada uno de esos atributos (Estructura Dinámica).

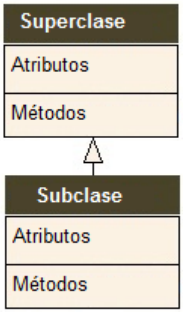
El **Comportamiento** de un objeto viene determinado por su forma de actuar y esta se ve representada por los **métodos** o subprogramas incluidos en su definición (La Clase).

Basado en lo anterior, al modelar una aplicación de computadora basado en el paradigma orientado a objetos, se debe tener en cuenta lo siguiente:

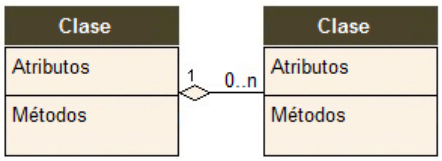
- Cada clase es representada mediante un rectángulo con 3 áreas:



- La relación de **Herencia** que indica que una clase recibe todos los atributos y propiedades de otra clase (denominada **Superclase**) se representa mediante el siguiente diagrama:

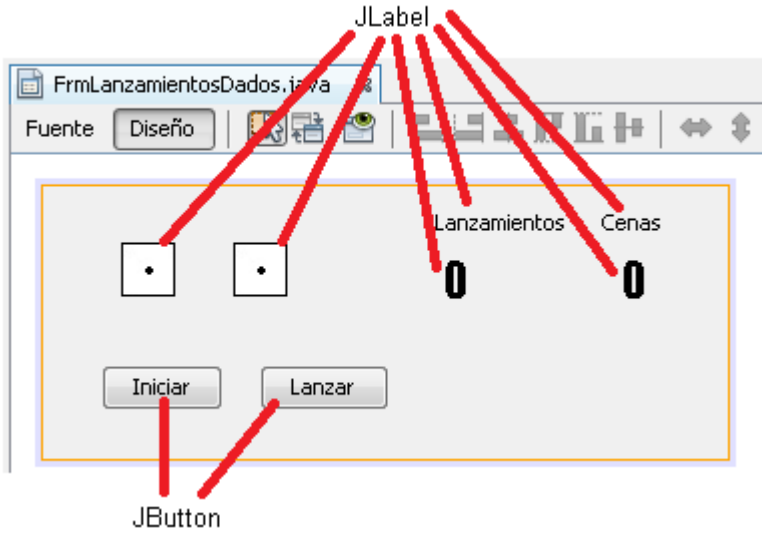


- La relación de **Agregación** la cual indica que una clase contendrá objetos (instancias) de otra clase se representa mediante el siguiente diagrama:



Programa

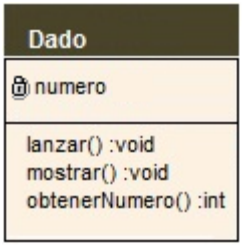
Para la implementación del aplicativo en *Java* se debe comenzar con el diseño de un formulario como el siguiente (Utilizando el *IDE Netbeans*):



Este formulario corresponde a la clase *Juego* del anterior diagrama de clases. Al ser un formulario, hereda de la clase *JFrame*. La siguiente tabla relaciona los objetos a añadir con las propiedades cuyos valores deben ser cambiados:

Clase	Nombre	Otras Propiedades
JLabel	*lblDado1*	Icon="Imágenes\1.jpg" text = ""
	lblDado2	Icon="Imágenes\1.jpg" text = ""
	jLabel1	text = "Lanzamientos"
	jLabel2	text = "Cenas"
	lblLanzamientos	font="Impact 24 simple" text = "0"
	lblCenas	font="Impact 24 simple" text = "0"
JButton	*btnIniciar*	text = "*Iniciar*"
	btnLanzar	text = "*Lanzar*" enabled= false

De acuerdo al modelo de clases planteado, se debe editar la clase *Dado* la cual tendrá la funcionalidad asociada a cada dado. Esta se compone de:



- La propiedad *número* de tipo entero y carácter privado. Solo se muestra su valor mediante el método *obtenerNumero()* y su valor cambia mediante el método *lanzar()* de manera aleatoria.
- El método *lanzar()* recibe un parámetro de la clase *Random* el cual es un generador de números aleatorios. Este se utiliza para generar un entero entre 1 y 6 el cual se almacena en la propiedad *numero*.

- El método *mostrar()* mostrará la cara del dado que corresponda a la propiedad *numero*.

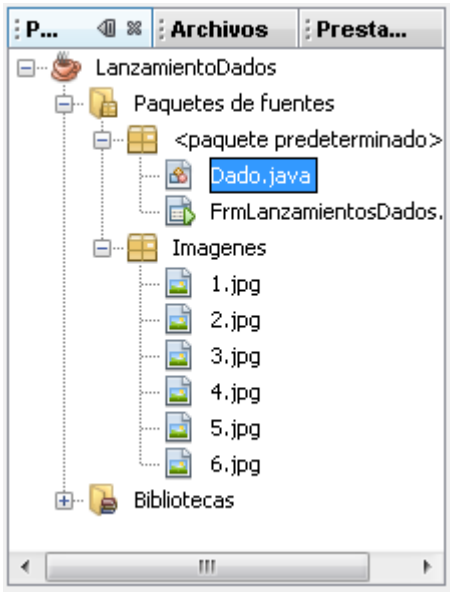
- El método *obtenerNumero()* devuelve el número obtenido mediante un lanzamiento.

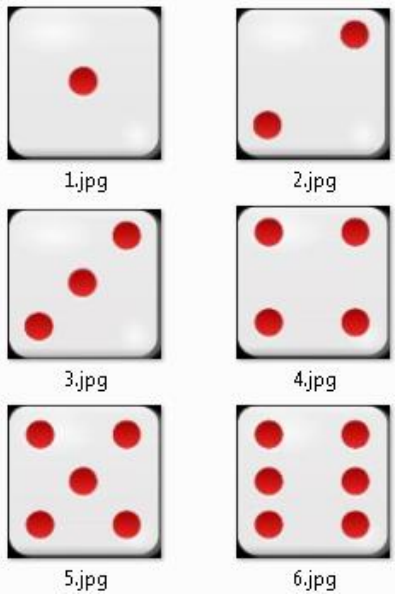
El código respectivo sería el siguiente:

```
import java.util.*;
import javax.swing.*;

public class Dado
{
    private int numero;

    //Metodo constructor
    public Dado()
    {
    }

    //Metodo que simula el lanzamiento del dado
    public void lanzar(Random r)
    {
        numero=r.nextInt(6)+1;
    }

    public void mostrar(JLabel lblDado)
    {
         lblDado.setIcon(new
ImageIcon(getClass().getResource("/Imagenes/"+String.valueOf(numero)+".jpg"))
);
    }

    public int obtenerNumero()
    {
        return numero;
    }
}
```

En este código se pueden apreciar los siguientes detalles de implementación:

- El método *lanzar()* recibe un objeto de la clase *Random*. Esta clase de objetos permiten generar números seudoaleatorios de acuerdo a una semilla, la cual si no se especifica, se basa en el reloj del sistema. En este ejercicio, para generar los números aleatorios entre 1 y 6, se utilizará el método *nextInt()* el cual pide un parámetro de tipo entero correspondiente al mayor entero no incluido en el rango a generar. En este caso se pasa el valor 6 que indica que se generarán enteros entre 0 y 5. Por esta razón, se debe añadir una unidad para que sean los valores 1 al 6.
- El método *mostrar()* recibe un objeto de la clase *JLabel* el cual permite mostrar imágenes. Esto se realiza asignándole un objeto de la clase *ImageIcon*.
  La instancia de dicho objeto se realiza haciendo referencia a imágenes que deben ser agregadas al proyecto. En este caso, se debe agregar una carpeta con las 6 imágenes correspondientes a las 6 caras de un dado.

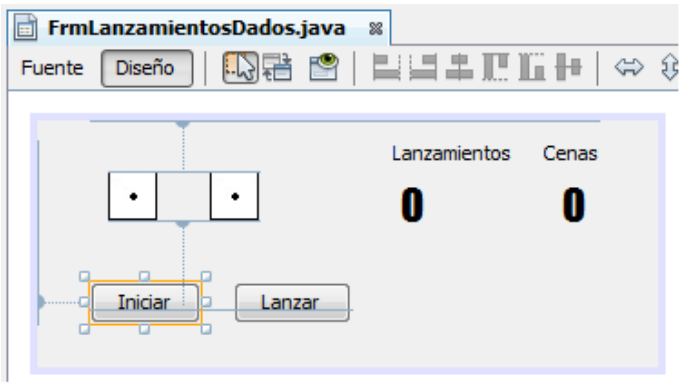


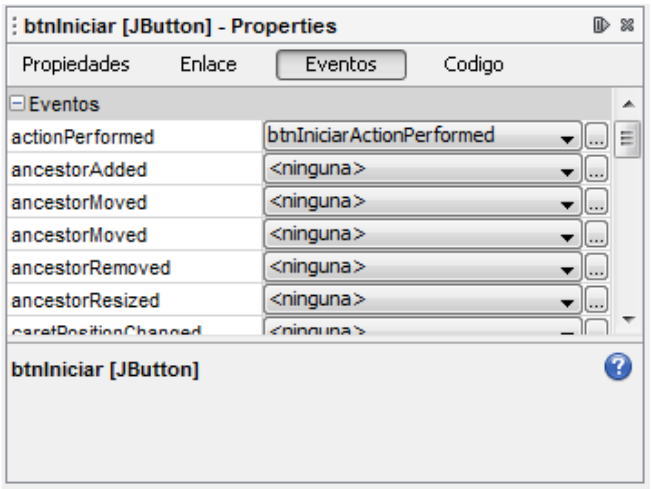


La referencia se realiza mediante la instrucción *getClass().getResource()* a la cual se pasa un parámetro String con el nombre completo del archivo. En este caso, el nombre dependerá del número obtenido mediante el lanzamiento.

Ahora bien, para continuar con la codificación, se deben programar los métodos *lanzar()* e *iniciar()* de la clase *Juego* y que corresponderán a los eventos de los botones de comando agregados al formulario.

El inicio de cada serie de lanzamientos se hará mediante un método que responde al evento ***actionPerformed*** del botón de comando ***btnIniciar***:





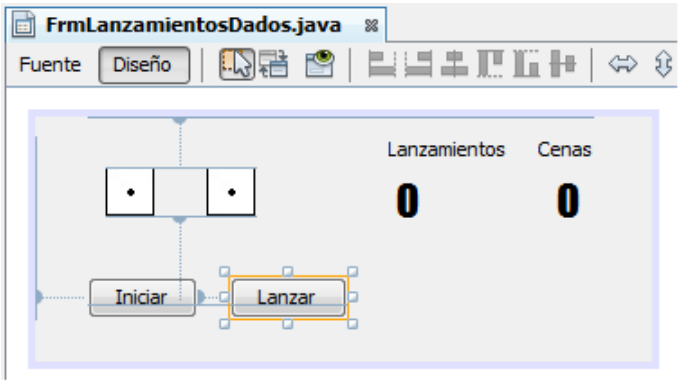
El código respectivo sería el siguiente:

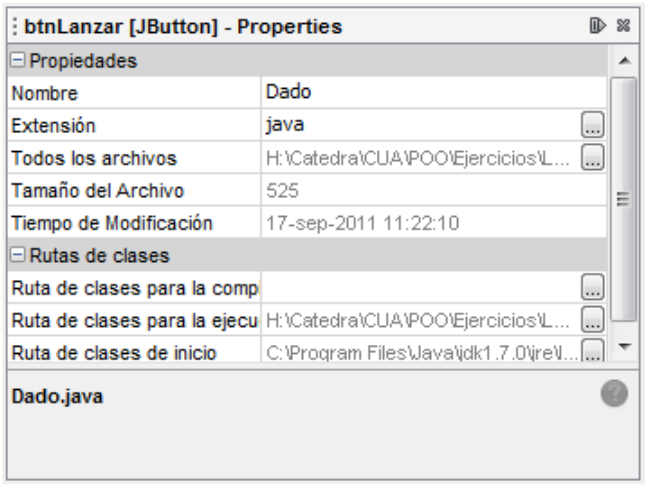
```
private void btnIniciarActionPerformed(java.awt.event.ActionEvent evt)
{
    lanzamientos=0;
    cenas=0;
```

```
    lblLanzamientos.setText("0");
    lblCenas.setText("0");
    btnLanzar.setEnabled(true);
}
```

En este código se inicializan las variables globales *lanzamientos* y *cenas*, las cuales contarán cada lanzamiento y cada vez que ocurra una "cena". Se habilita también el botón *btnLanzar* que inicialmente esta deshabilitado.

El lanzamiento de los dados se simulará mediante el método que responde al evento ***actionPerformed*** del botón de comando ***btnLanzar***:





El código respectivo sería el siguiente:

```
private void btnLanzarActionPerformed(java.awt.event.ActionEvent evt)
{
    //Lanzar los dados
    d1.lanzar(r);
    d1.mostrar(lblDado1);
    d2.lanzar(r);
    d2.mostrar(lblDado2);
    //Incrementar lanzamientos
    lanzamientos++;
    //Si ocurre una cena, incrementar el respectivo contador
    if(d1.obtenerNumero()+d2.obtenerNumero()>=11)
        cenas++;
    //Mostrar los contadores
    lblLanzamientos.setText(String.valueOf(lanzamientos));
    lblCenas.setText(String.valueOf(cenas));
}
```

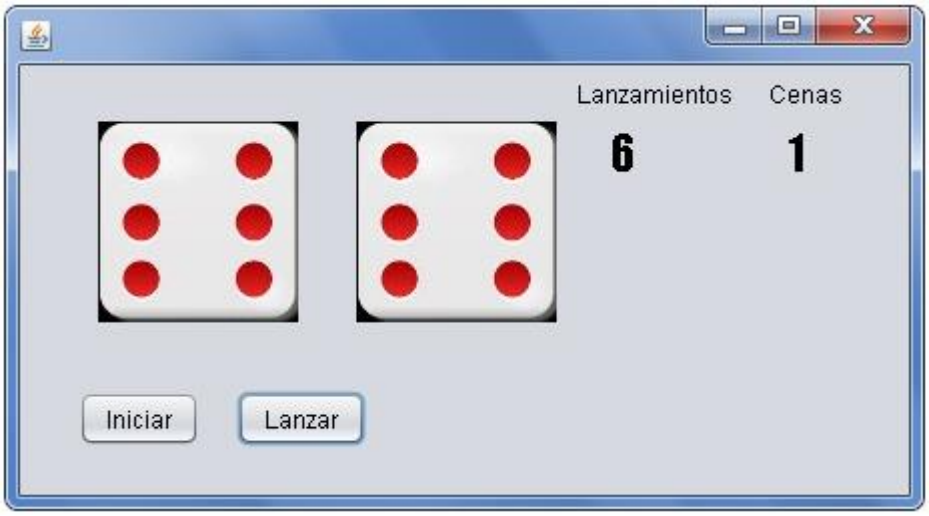
Este código simula el lanzamiento de los dados. Para ello se invocan los métodos de la clase *Dado* mediante las 2 instancias que se debieron declarar globalmente así:

```
import java.util.*;
```

```
public class FrmLanzamientosDados extends javax.swing.JFrame {

    Dado d1=new Dado();
    Dado d2=new Dado();
    Random r=new Random();

    int lanzamientos;
    int cenas;
}
```

Junto a la declaración global de los dados (Objetos *d1* y *d2*) también se realizan la del objeto *Random* que supone la suerte del juego y la de los contadores de lanzamientos y cenas.

Para verificar su funcionamiento, se simulan los lanzamientos necesarios hasta obtener una "cena" cuya pantalla luciría así:



2. El juego de azar "Bingo" se realiza entre varios jugadores y un "cantor". Cada jugador debe tener una tabla que viene dividida en cuadrados numerados. La tabla tiene 5 columnas y 5 filas. Las columnas están encabezadas con las letras "B","I","N","G" y "O", cada una de las cuales alberga debajo 5 números comprendidos entre los siguientes rangos:

   - ***"B"***: 1 a 15
   - ***"I"***: 16 a 30
   - ***"N"***: 31 a 45
   - ***"G"***: 46 a 60
   - ***"O"***: 61 a 75

   A la derecha se tiene un ejemplo de una tabla. Se puede observar que el centro de la tabla no tiene número.

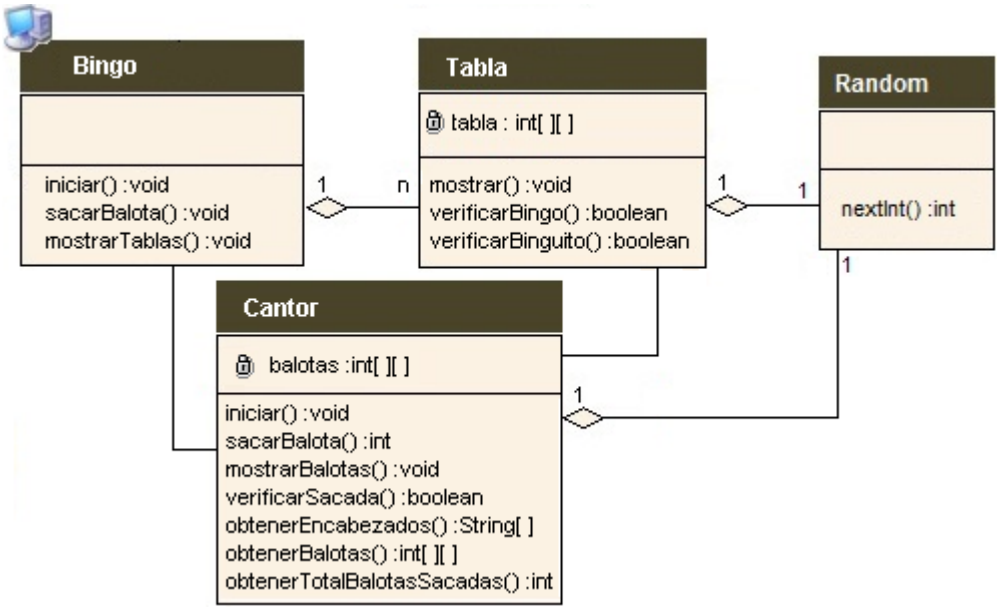
B	I	N	G	O
7	25	44	57	62
15	22	40	50	70
11	30	FREE SPACE	46	74
2	28	37	55	68
10	27	39	59	75

El Bingo es un juego muy similar a la Lotería. La persona denominada "Cantor" saca al azar balotas con números del 1 al 75. Las anuncia y los participantes del juego deben marcar los números si es que los tienen en sus respectivas tablas.

El objetivo de este juego es marcar todos los números de la tabla ("*Bingo*") o completar una línea horizontal, vertical o diagonal (*"Binguito*"). El jugador que lo logre, grita "Bingo" y gana el premio.

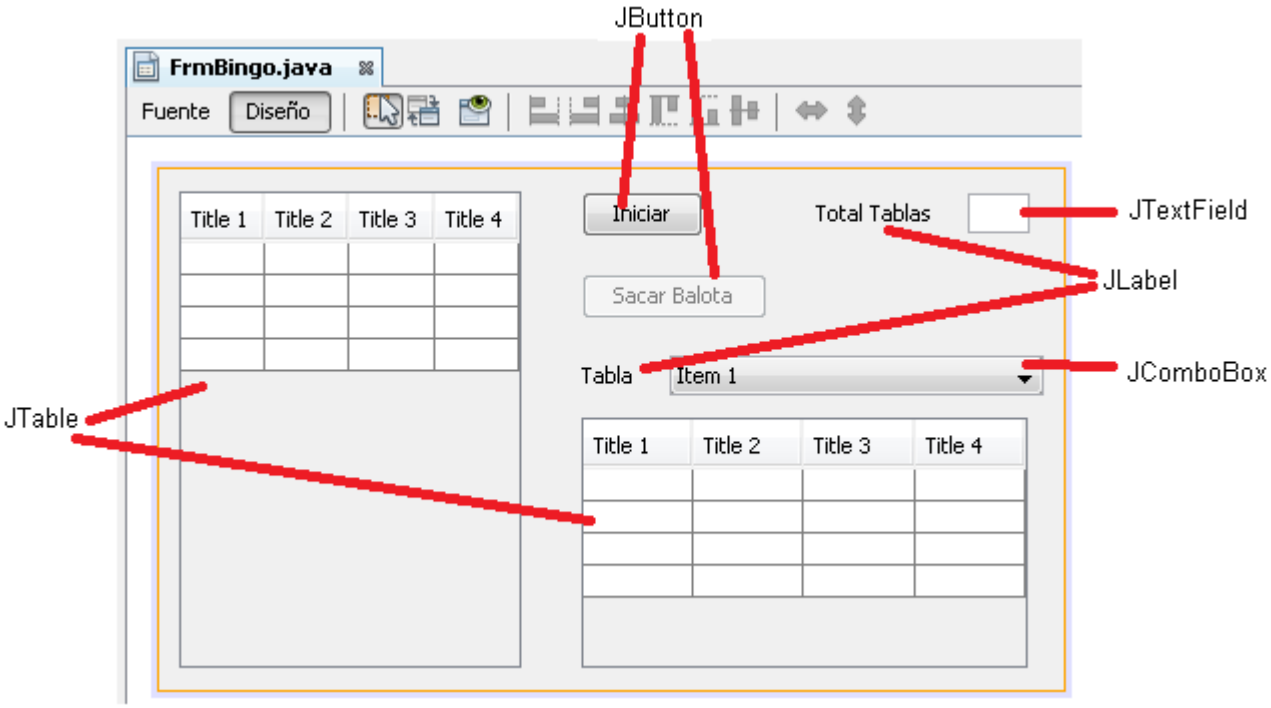
R/

- El modelado del aplicativo bajo el paradigma Orientado a Objetos se ilustra en el siguiente diagrama de clases:



Programa

Para la implementación del aplicativo en *Java* se debe comenzar con el diseño de un formulario como el siguiente (Utilizando el *IDE Netbeans*) y que corresponde a la clase *Bingo* del anterior diagrama de clases:



La siguiente tabla relaciona los objetos a añadir con las propiedades cuyos valores deben ser cambiados:

Tipo Control	Nombre	Otras Propiedades
JLabel	*jLabel1*	text = "*Total Tablas*"
	jLabel2	text = "*Tabla*"
JButton	*btnIniciar*	text = "*Iniciar*"
	btnSacarBalota	text = "*Sacar Balota*" enabled = false
JTable	*tblBingo*	enabled = false
	tblTabla	enabled = false
JTextField	*txtTotalTablas*	text = ""
JComboBox	*cmbTablas*	model = *(vacío)*

De acuerdo al modelo de clases planteado, se debe editar la clase *Tabla* la cual tendrá la funcionalidad asociada a cada tabla que participa en el juego. Esta se compone de:

- La propiedad *tabla* la cual es una matriz de enteros y carácter privado. Solo se muestra su valor mediante el método *mostrar()* utilizando un objeto *JTable*. Los valores de esta matriz se generan aleatoriamente mediante el método constructor de la clase.
- Los métodos *verificarBingo* y *VerificarBinguito()* los cuales devuelven un valor booleano con base en las balotas sacadas.

El código correspondiente a la clase sería el siguiente:

```
import java.util.Random;
```

```
import javax.swing.*;
import javax.swing.table.*;

public class Tabla {
    //Generador de numeros aleatorios
    private Random r;
    //Matriz para la tabla
    private int[][] tabla;

    public Tabla(int numero) {
        /* La semilla del generador de numeros aleatorios
         * depende de la la hora de instancia del objeto
         * y del numero del jugador*/
        r = new Random(System.currentTimeMillis() * numero);
        // Generar la Tabla
        tabla = new int[5][5];
        for (int c = 0; c < 5; c++) {
            for (int f = 0; f < 5; f++) {
                tabla[f][c] = 0;
                //Ubicación diferente del centro del tabla
                if (f != 2 || c != 2) {
                    while (tabla[f][c] == 0) {
                        numero = (c * 15) + r.nextInt(14) + 1;
                        if (!verificarNumero(numero, c)) {
                            tabla[f][c] = numero;
                        }
                    }
                }
            }
        }
    }

    //Metodo que verifica que un numero no este repetido
    private boolean verificarNumero(int numero, int columna) {
        int f = 0;
        boolean encontrado = false;
        while (f < 5 && !encontrado) {
            if (tabla[f][columna] == numero) {
                encontrado = true;
            } else {
                f++;
            }
        }
        return encontrado;
    }

    //Metodo que devuelve los valores que hay en la tabla
    public int[][] obtenerTabla() {
        return tabla;
    }

    //Metodo que muestra los datos de la tabla en un JTable
    public void mostrar(JTable tbl) {
        int f = tabla.length;
```

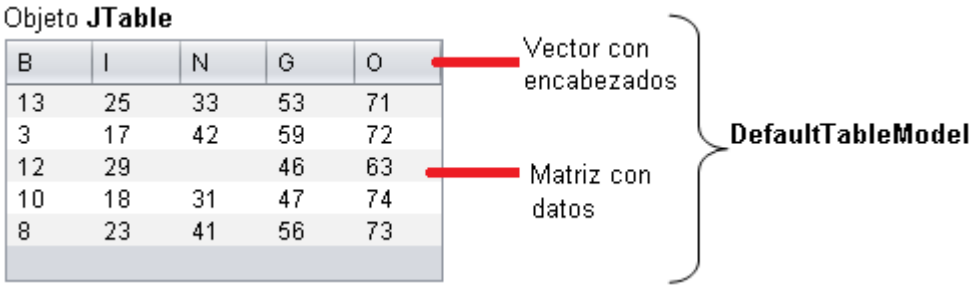
```
        int c = tabla[0].length;
        //Convertir enteros en textos
        String[][] m = new String[f][c];
        for (int i = 0; i < f; i++) {
            for (int j = 0; j < c; j++) {
                if (i != 2 || j != 2) {
                    m[i][j] = String.valueOf(tabla[i][j]);
                    if (Cantor.verificarSacada(tabla[i][j])) {
                        m[i][j] += "*";
                    }
                }
            }
        }
        tbl.setModel(new DefaultTableModel(m, Cantor.obtenerEncabezados()));
    }

    //Metodo que verifica con base en las balotas sacadas
    //si se ha presentado un Bingo, es decir,
    //si todos los números de la tabla han sido sacados
    public boolean verificarBingo() {
        boolean bingo = true;
        int ft = 0;
        while (ft < 5 && bingo) {
            int c = 0;
            while (c < 5 && bingo) {
                if (ft != 2 || c != 2) {
                    bingo = false;
                    int fb = 0;
                    while (fb < 15 && !bingo) {
                        if (tabla[ft][c] == Cantor.obtenerBalotas()[fb][c]) {
                            bingo = true;
                        } else {
                            fb++;
                        }
                    }
                }
                c++;
            }
            ft++;
        }
        return bingo;
    }

    //Metodo que verifica con base en las balotas sacadas
    //si se ha presentado un Binguito, es decir,
    //si todos los números en una fila o columna han sido sacados
    public boolean verificarBinguito() {
        boolean binguito = false;
        //Buscar binguito por fila
        int c, fb;
        int ft = 0;
        while (ft < 5 && !binguito) {
            c = 0;
            binguito = true;
```

```
            while (c < 5 && binguito) {
                if (ft != 2 || c != 2) {
                    binguito = false;
                    fb = 0;
                    while (fb < 15 && !binguito) {
                        if (tabla[ft][c] == Cantor.obtenerBalotas()[fb][c]) {
                            binguito = true;
                        } else {
                            fb++;
                        }
                    }
                }
                c++;
            }
            ft++;
        }
        if (!binguito) {
            //Buscar binguito por columna
            c = 0;
            while (c < 5 && !binguito) {
                ft = 0;
                binguito = true;
                while (ft < 5 && binguito) {
                    if (ft != 2 || c != 2) {
                        binguito = false;
                        fb = 0;
                        while (fb < 15 && !binguito) {
                            if                (tabla[ft][c]                ==
Cantor.obtenerBalotas()[fb][c]) {
                                binguito = true;
                            } else {
                                fb++;
                            }
                        }
                    }
                    ft++;
                }
                c++;
            }
        }
        return binguito;
    }
}
```

En la implementación de dicha clase se debe tener en cuenta lo siguiente:

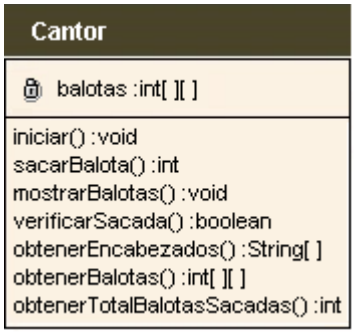
- El método constructor pide un número de identificación de la tabla el cual será usado como semilla del generador de números aleatorios.
- El método *mostrar()* pide como parámetro de entrada un objeto *JTable*, el cual permite mostrar datos provenientes de una matriz de objetos. Para ello se alterará su *Modelo de datos*, es decir, la estructura que contiene los datos a mostrar. Esto se realiza mediante el método *setModel()*. Este método recibirá un nuevo objeto *DefaultTableModel* cuyo constructor recibirá:

- La matriz de números que conforma la tabla, la cual debe ser convertida a textos. Se aprovechará para agregar un distintivo (asterisco) a los números que han sido sacados.
- Los encabezados, los cuales se obtienen de la clase *Cantor* (método *obtenerEncabezados()*).



- El método *verificarBingo()* verifica que todos los valores de la matriz (con excepción de la posición fila 3, columna 3) estén en las balotas sacadas, los cuales se obtienen de la clase *Cantor* (método *obtenerBalotas()*).
- El método *verificarBinguito()* verifica que todos los valores en alguna de las filas o de las columnas de la matriz estén en las balotas sacadas.

Ahora se continúa con la implementación de la clase *Cantor*, la cual será una clase abstracta, es decir, no requiere instancia y sus propiedades y métodos serán estáticos. Esta se compone de:



- La propiedad *balotas* la cual es una matriz de enteros y de carácter privado. Solo se muestra su valor mediante el método *mostrarBalotas()* utilizando un objeto *JTable*. La estrategia es la misma que la del método *mostrar()* de la clase *Tabla*.
- El método *iniciar()* asigna a la matriz de enteros el valor de 0, lo cual indica que ninguna balota aún no ha sido sacada.
- El método *sacarBalota()* genera un número aleatorio entre 1 y 75 simulando la sacada de una balota.
- El método *verificarSacada()* permite evaluar si una posición ha sido sacada durante el juego. Es útil para poder verificar las casillas de las tablas al momento de mostrarlas.
- El método *obtenerEncabezados()* devuelve el vector de textos de encabezado para las columnas tanto de las tablas como de las balotas y corresponde a las letras de la palabra “BINGO”.
- El método *obtenerBalotas()* devuelve la matriz de enteros de las balotas sacadas. De gran utilidad para los métodos *verificarBingo()* y *verificarBinguito()* de la clase *Tabla*.
- El método *obtenerTotalBalotasSacadas()* devuelve cuantas balotas han sido sacadas. Sirve para controlar la duración del juego.

El código de la clase sería el siguiente:

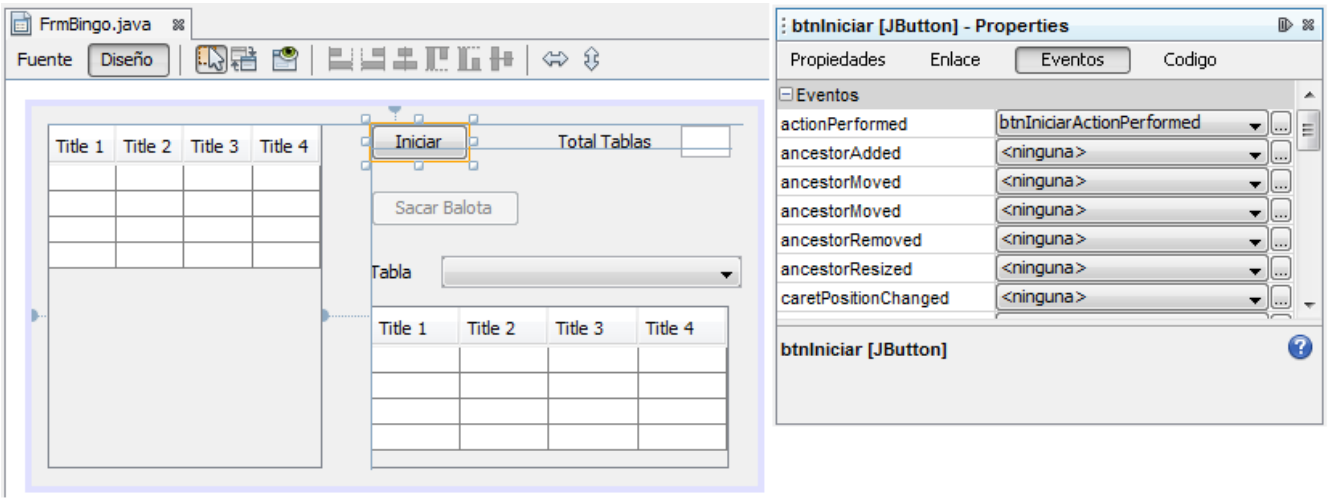
```
import java.util.*;
import javax.swing.*;
import javax.swing.table.*;

public abstract class Cantor {
```

```
    private static String[] encabezados = new String[]{"B", "I", "N", "G",
"O"}; //Encabezados de las tablas
    private static int[][] balotas = new int[15][5]; //Matriz que almacena
las balotas sacadas
    private static int totalBalotasSacadas;
    private static Random r = new Random();

    //Metodo que inicia las balotas como no sacadas
    public static void iniciar() {
        //Iniciar las balotas
        for (int c = 0; c < 5; c++) {
            for (int f = 0; f < 15; f++) {
                balotas[f][c] = 0;
            }
        }
        totalBalotasSacadas = 0;
    }

    //Metodo que simula el sacar una balota de manera aleatoria
    public static int sacarBalota() {
        int numero = 0;
        if (totalBalotasSacadas < 75) {
            boolean sacada = false;
            while (!sacada) {
                //Sacar la balota
                numero = r.nextInt(75) + 1;
                //Fila de la balota
                int f = numero % 15;
                if (f == 0) {
                    f = 15;
                }
                //Columna de la balota
                int c = (numero - f) / 15;
                if (balotas[f - 1][c] == 0) {
                    sacada = true;
                    balotas[f - 1][c] = numero;
                    totalBalotasSacadas++;
                }
            }
        }
        return numero;
    }

    //Metodo para mostrar las balotas que han sido sacadas
    public static void mostrarBalotas(JTable tbl) {
        int f = balotas.length;
        int c = balotas[0].length;
        //Convertir enteros en textos
        String[][] m = new String[f][c];
        for (int i = 0; i < f; i++) {
            for (int j = 0; j < c; j++) {
                if (balotas[i][j] == 0) {
                    m[i][j] = "";
```

```
                } else {
                    m[i][j] = String.valueOf(balotas[i][j]);
                }
            }

        }
        tbl.setModel(new DefaultTableModel(m, encabezados));
    }

    //Metodo que permite verificar si una balota ha sido sacada
    public static boolean verificarSacada(int numero) {
        if (totalBalotasSacadas > 0) {
            int f = numero % 15;
            if (f == 0) {
                f = 15;
            }
            int c = (numero - f) / 15;
            if (balotas[f - 1][c] == 0) {
                return false;
            } else {
                return true;
            }
        } else {
            return false;
        }
    }

    //Metodo que devuelve los encabezados de las columnas de las tablas
    public static String[] obtenerEncabezados() {
        return encabezados;
    }

    //Metodo que devuelve la matriz de las balotas sacadas
    public static int[][] obtenerBalotas() {
        return balotas;
    }

    //Metodo que devuelve la cantidad de balotas sacadas
    public static int obtenerTotalBalotasSacadas() {
        return totalBalotasSacadas;
    }
}
```

Continuando con la codificación del proyecto, se deben programar los métodos *iniciar()*, *sacarBalota()* y *mostrarTablas()* de la clase *Bingo* y que corresponderán a eventos de los botones de comando y lista desplegable agregados al formulario.

El inicio del juego, en el cual se deja que ninguna balota sea sacada y se generen al azar las tablas que participarán se hará mediante un método que responde al evento ***actionPerformed*** del botón de comando ***btnIniciar***:

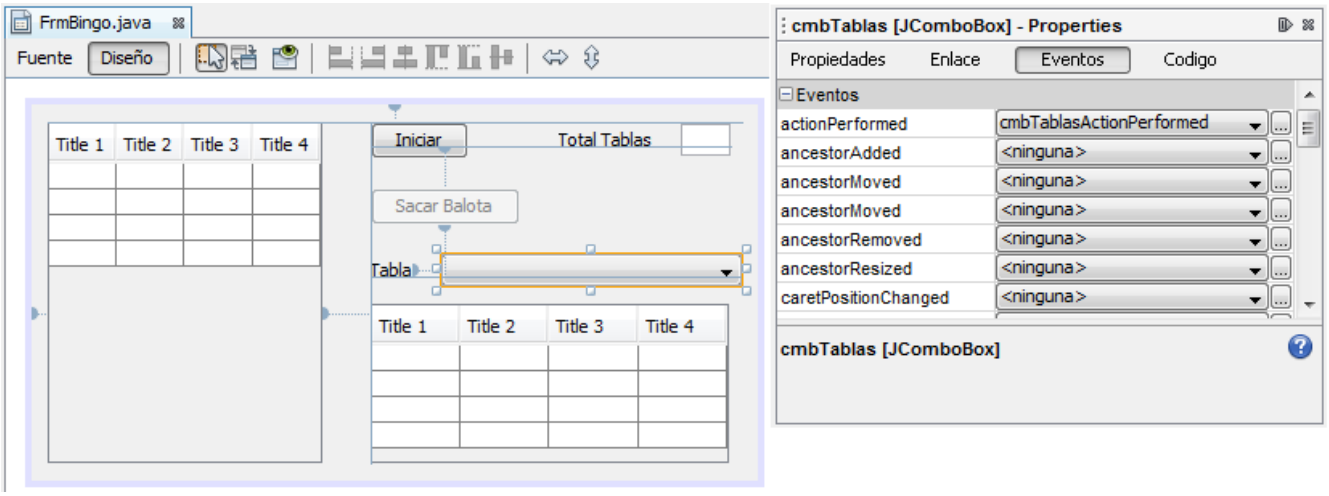


El código respectivo sería el siguiente:

```
private void btnIniciarActionPerformed(java.awt.event.ActionEvent evt) {
   int totalTablas = 2;
    try {
        totalTablas = Integer.parseInt(txtTotalTablas.getText());
    } catch (Exception ex) {
        txtTotalTablas.setText("2");
    }
    tablas = new Tabla[totalTablas];
    cmbTablas.removeAllItems();
    for (int i = 0; i < totalTablas; i++) {
        tablas[i] = new Tabla(i);
        cmbTablas.addItem("Tabla " + (i + 1));
    }
    Cantor.iniciar();
    Cantor.mostrarBalotas(tblBingo);
    btnSacarBalota.setEnabled(true);
}
```

En este código se crean las instancias de las tablas que participarán en el juego (vector *tablas*) con base en un dato de entrada que se lee mediante una caja de texto (*txtTotalTablas*). Si en esta caja de texto se especifica un dato no válido, predeterminadamente se crean 2 tablas. También se invocan los métodos *iniciar()* y *mostrarBalotas()* de la clase abstracta *Cantor* para indicar que ninguna balota ha sido sacada. Observe que a diferencia de la clase *Tabla*, no hubo que crear instancias de objetos (uso del método constructor mediante la instrucción *new*).

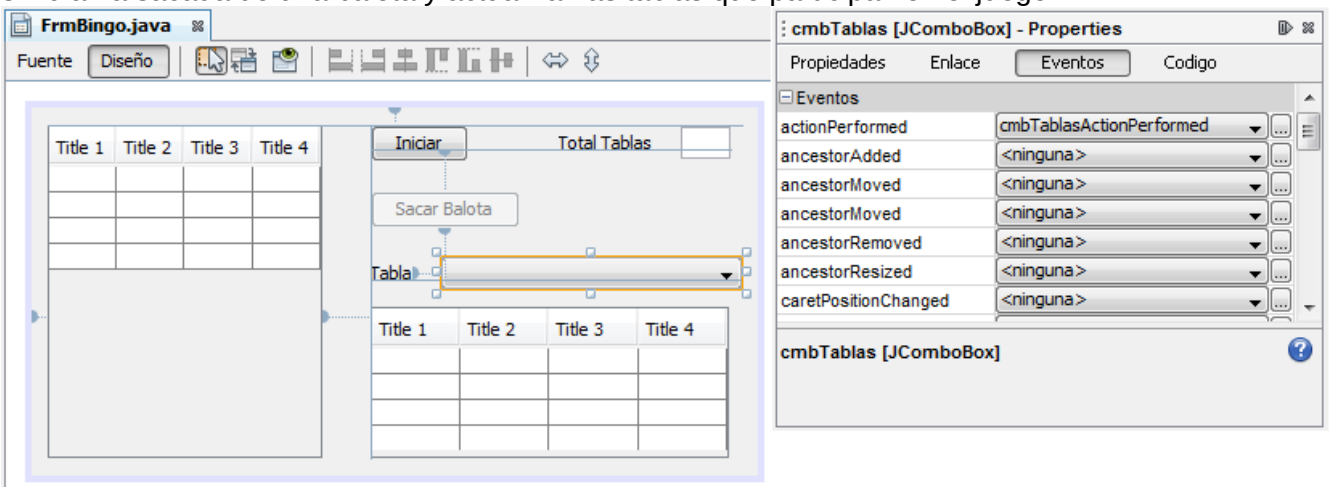
La mostrada de las tablas se realiza mediante el evento ***actionPerformed*** de la lista desplegable ***cmbTablas***. En este caso, sólo se mostrará una tabla a la vez, la cual se selecciona mediante dicha lista:



El código sería el siguiente:

```
private void cmbTablasActionPerformed(java.awt.event.ActionEvent evt) {
    if (cmbTablas.getSelectedIndex() >= 0) {
        tablas[cmbTablas.getSelectedIndex()].mostrar(tblTabla);
    }
}
```

Por último, el evento evento ***actionPerformed*** del botón de comando ***btnsacarBalota*** permitirá simular la sacada de una balota y actualizar las tablas que participan en el juego:



El código sería el siguiente:

```
private void btnSacarBalotaActionPerformed(java.awt.event.ActionEvent evt) {
    if (Cantor.sacarBalota() > 0) {
        Cantor.mostrarBalotas(tblBingo);
        if (cmbTablas.getSelectedIndex() >= 0) {
            tablas[cmbTablas.getSelectedIndex()].mostrar(tblTabla);
        }

        //Verificar bingos y binguitos
```

```
        for (int i = 0; i < cmbTablas.getItemCount(); i++) {
            if (tablas[i].verificarBingo()) {
                JOptionPane.showMessageDialog(new         JFrame("Verificando
Bingo"), "BINGO para el jugador " + (i + 1));
            }
            else if (tablas[i].verificarBinguito()) {
                JOptionPane.showMessageDialog(new         JFrame("Verificando
Binguito"), "BINGUITO para el jugador " + (i + 1));
            }
        }

        if (Cantor.obtenerTotalBalotasSacadas() == 75) {
            btnSacarBalota.setEnabled(false);
            JOptionPane.showMessageDialog(new  JFrame(),  "El  juego  termina
porque no hay más balotas para sacar");
        }
    } else {
        JOptionPane.showMessageDialog(new    JFrame(),    "El    juego    había
terminado");
    }
}
```

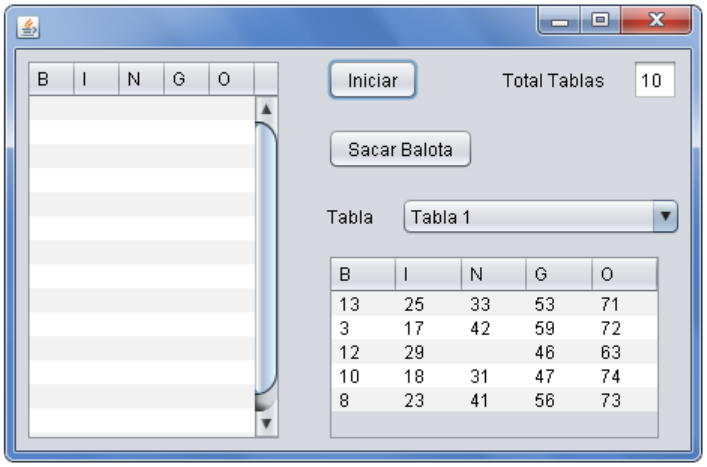
En este código se verifica en primer lugar si se ha podido sacar una balota. En caso afirmativo, se muestran las balotas sacadas, y la tabla seleccionada con los números destacados correspondientes a las balotas sacadas.

Seguido, se verifican los bingos y binguitos que se presentan en todas las tablas y se muestran los respetivos mensajes.

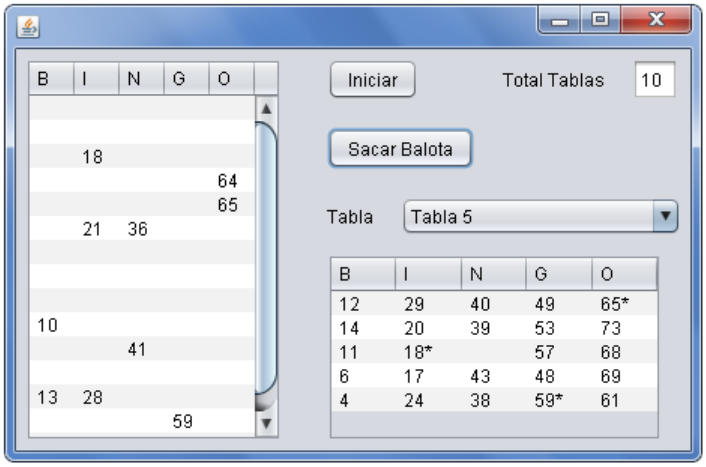
Si se han sacado todas las balotas, se termina el juego, deshabilitándose el botón para sacar balotas.

La ejecución del juego luciría así:

- Cuando se inicia un juego, ninguna balota aparece sacada:

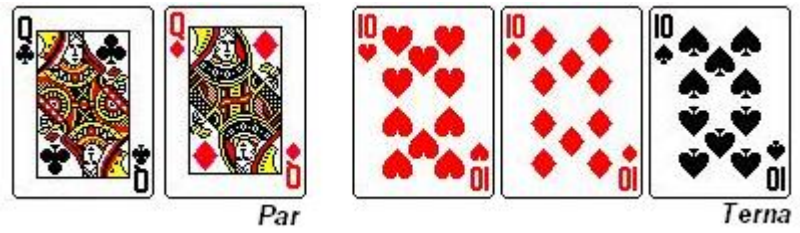


- Cuando se sacan balotas, estas aparecen en el respectivo *JTable* y destacadas en las tablas que participan en el juego (mediante un asterisco):

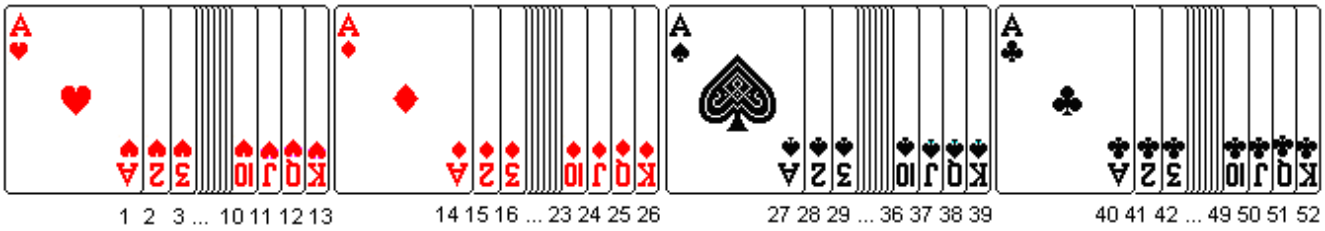


- Cuando se presenta un binguito o bingo en una tabla, aparece el respectivo mensaje:

3. En un juego basado en la baraja inglesa, denominado *"Apuntado",* se reparten a cada jugador 10 cartas. Cada jugador debe identificar las figuras que hay entre las cartas, es decir los agrupamientos por el nombre de la carta. Por ejemplo, si entre las 10 cartas hay 2 *Queen (Q)* independiente de la pinta, hay 1 par de *Q*. Si hay 3 *Diez (10)* con diferentes o iguales pintas, hay una terna de *10*:

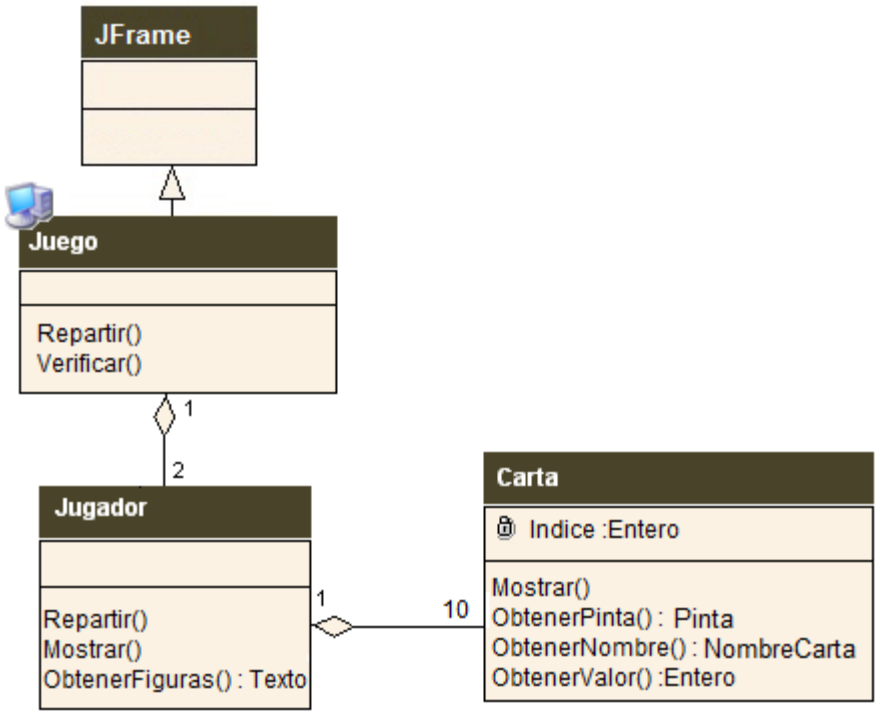


Para quien no conozca la baraja inglesa, esta se compone de un juego de 52 cartas que combina 13 nombres de cartas con 4 tipos de pinta:

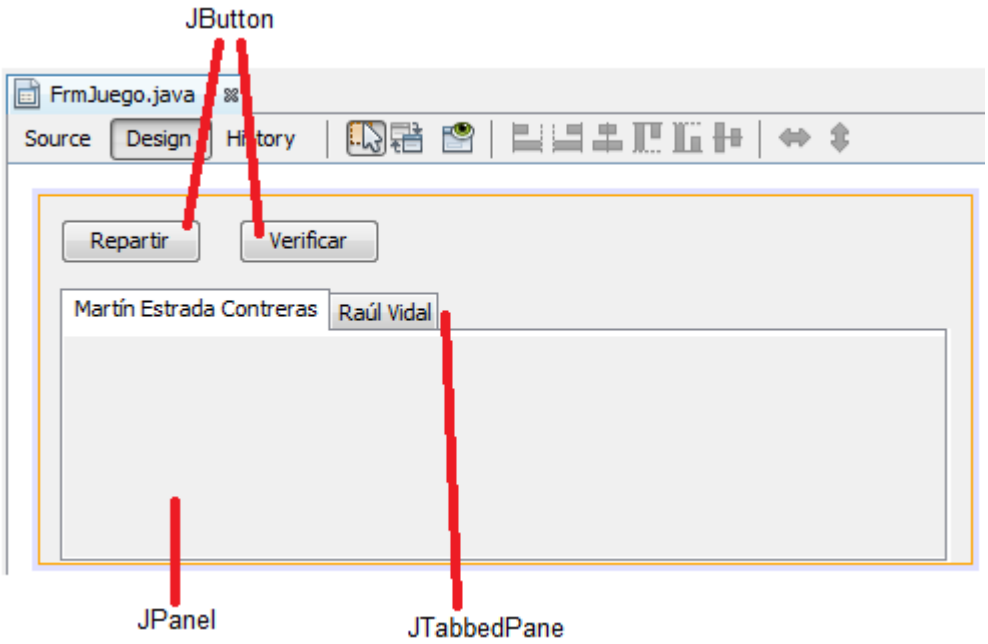


R/

- El modelado bajo el paradigma Orientado a Objetos se ilustra en el siguiente diagrama de clases:



- **Programa**
  Para la implementación del aplicativo en *Java* (utilizando la IDE *Netbeans)* se debe comenzar con el diseño de un formulario como el siguiente:

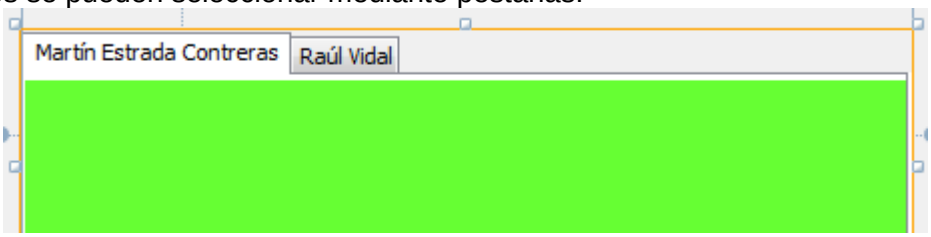


Este formulario corresponde a la clase *Juego* del anterior diagrama de clases. Al ser un formulario, hereda de la clase *JFrame*. La siguiente tabla relaciona los objetos a añadir con las propiedades cuyos valores deben ser cambiados:

Tipo Control	Nombre	Otras Propiedades
JButton	*btnRepartir*	Text = “Repartir”
	btnVerificar	Text = “Verificar”
JTabbedPane	*tpJugadores*	
JPanel	*pnlJugador1*	Text = “Martín Estrada Contreras”
	pnlJugador2	Text = “Raúl Vidal”

En este diseño de formulario aparecen unos nuevos tipos de objetos que se comportan como contenedores de otros objetos:

- ***JTabbedPane:*** Corresponde a un agrupador de contenedores de objetos cuyos contenidos se pueden seleccionar mediante pestañas:



  Cada contenedor asociado a una pestaña es un objeto ***JPanel***

- ***JPanel***: Corresponde a un contenedor de objetos. Predeterminadamente, todo formulario (un objeto *JFrame*) incluye 2 paneles: *ContentPane* y *RootPane*.

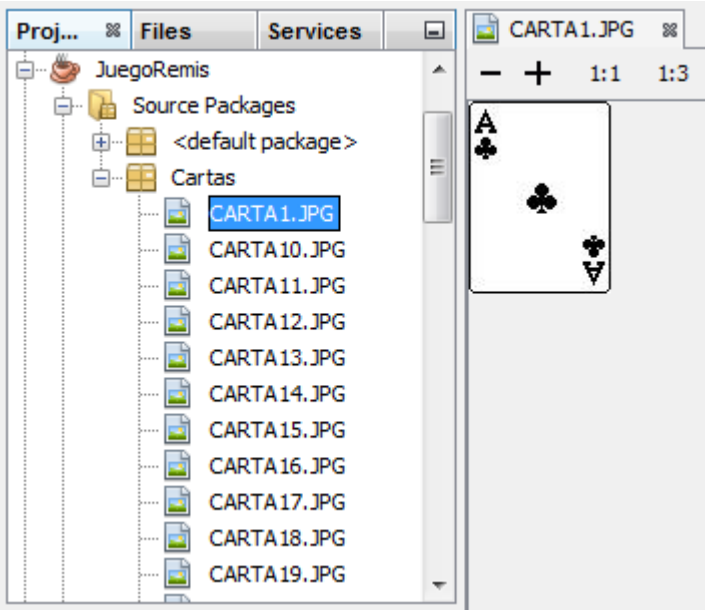
De acuerdo al modelo de clases planteado, se debe editar la clase *Carta* la cual tendrá la funcionalidad asociada a cada carta. Esta se compone de:

- La propiedad *indice* que identificará la carta de acuerdo a la siguiente convención:
  - Valores entre 1 y 13 para las cartas con pinta ♥
  - Valores entre 14 y 26 para las cartas con pinta ♦
  - Valores entre 27 y 39 para las cartas con pinta ♠
  - Valores entre 40 y 52 para las cartas con pinta ♣

  Es de observar que este valor debe coincidir con un conjunto de imágenes correspondientes a las 52 cartas en el siguiente orden:

Pinta	Nombre Carta												
	A	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***J***	***Q***	***K***
♣	1	2	3	4	5	6	7	8	9	10	11	12	13
♠	14	15	16	17	18	19	20	21	22	23	24	25	26
♥	27	28	29	30	31	32	33	34	35	36	37	38	39
♦	40	41	42	43	44	45	46	47	48	49	50	51	52

Para lo cual se incluirá en el proyecto una carpeta denominada *Cartas* la cual incluirá las 52 imágenes cuyo nombre se formará con la palabra "*Carta*" y el respectivo índice:



Para efectos de controlar las listas de pintas y de los nombres de las cartas, se utilizará el tipo **Enumerado**. Esto posibilita que en lugar de utilizar un texto o número por cada pinta o nombre de carta, se utilice un valor constante que puede ser manipulado por su posición, además de otras posibilidades.

La siguiente es la declaración de los enumerados requeridos:

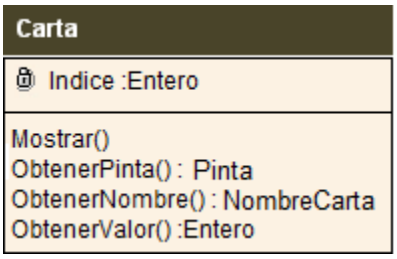
- La lista de pintas

```
public enum PintaCarta {

    NINGUNO,
    TREBOL,
    PICA,
    CORAZON,
    DIAMANTE
}
```

- La lista de nombress

```
public enum NombreCarta {
    NINGUNO,
    ACE,
    DOS,
    TRES,
    CUATRO,
    CINCO,
    SEIS,
    SIETE,
    OCHO,
    NUEVE,
    DIEZ,
    JACK,
    QUEEN,
    KING;
}
```

Algún valor de las anteriores listas, será el que devuelvan algunos de los métodos de la clase, como se podrá verificar a continuación:



- El método *Constructor* (utilizado en la instrucción ***new)***, que como se sabe, es invocado cuando se crea un objeto de la clase, generará aleatoriamente el índice de la carta.
- El método *mostrarCarta()* permite ver la imagen de la carta que corresponda al *indice*.
- El método *obtenerNombre()* devuelve un valor del enumerado *NombreCarta* con base en el *indice*.
- El método *obtenerPinta()* devuelve un valor del enumerado *Pinta* con base en el *indice*.
- El método *obtenerValor()* devuelve el valor de la carta con base en el atributo *índice*. Las cartas (“*Ace”, “Jack”, “Queen”*,y *“King”* valen 10 y el resto, el respectivo número del nombre.

El código respectivo sería el siguiente:

```
import java.util.*;
import javax.swing.*;

public class Carta {

/* Atributo privado para almacenar el numero de la carta
 * 1= As de Trebol, 14= As de Pica, 27= As de Corazon, 40= As de Diamante,
 * 2= 2 de Trebol, 15= 2 de Pica, 28= 2 de Corazon, 41= 2 de Diamante,
 * 3= 3 de Trebol, 16= 3 de Pica, 29= 3 de Corazon, 42= 3 de Diamante,
 * ...
 * 10= 10 de Trebol, 23= 10 de Pica, 36= 10 de Corazon, 49= 10 de Diamante,
 * 11= J de Trebol, 24= J de Pica, 37= J de Corazon, 50= J de Diamante,
 * 12= Q de Trebol, 25= Q de Pica, 38= Q de Corazon, 51= Q de Diamante,
 * 13= K de Trebol, 26= K de Pica, 39= K de Corazon, 52= K de Diamante
 * */
    private int indice;

    public Carta(Random r)
    {
        //El numero de la carta es generado aleatoriamente
        indice=r.nextInt(52)+1;
    }

    public Pinta obtenerPinta()
    {
        /* Obtiene la pinta que corresponde a la carta,
         * basado en el rango en que se ubica el índice
         */
        if(indice<=13)
            return Pinta.TREBOL;
        else if (indice<=26)
```

```
            return Pinta.PICA;
        else if (indice<=39)
            return Pinta.CORAZON;
        else
            return Pinta.DIAMANTE;
    }

    public NombreCarta obtenerNombre()
    {
        //Obtiene el nombre que corresponde al número de la carta
        int numero = indice % 13;
        if (numero == 0) {
            numero = 13;
        }
        return NombreCarta.values()[numero];
    }

    public void mostrarCarta(int x, int y, JPanel pnl, boolean tapada)
    {
        String nombreImagen;
        //Obtener el nombre del archivo de la carta
        if(tapada)
            nombreImagen = "/Cartas/Tapada.jpg";
        else
            nombreImagen = "/Cartas/Carta" + String.valueOf(indice) + ".jpg";

        //Cargar la imagen
        ImageIcon                imagen                =                new
ImageIcon(getClass().getResource(nombreImagen));
        //Instanciar label para mostrar la imagen
        JLabel lblCarta = new JLabel(imagen);
        //Definir posicion y dimensiones de la imagen
        lblCarta.setBounds(x, y,
                           x + imagen.getIconWidth(),
                           y + imagen.getIconHeight());
        //Mostrar la carta en la ventana
        pnl.add(lblCarta);
    }
}
```

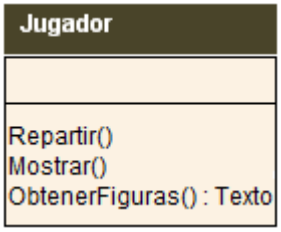
En este código se pueden apreciar los siguientes detalles de implementación:

- El método *mostrarCarta()* recibe como parámetros de entrada: Las coordenadas donde se ubicará el objeto que mostrará la imagen de la carta (un *JLabel*) el cual a su vez se incluirá en un objeto *JPanel,* y un valor booleano que permita decidir si mostrar la imagen de la carta o la imagen de una carta tapada.

  Lo anterior implica que en la carpeta de imágenes también deberá ir la imagen de la carta tapada.

- La divisibilidad por 13 es un factor clave para obtener el nombre de la carta y su respectivo valor a partir del *Indice*. Esto se debe a que el total de cartas por pinta es 13.

Siguiendo con el modelo de clases planteado, se debe editar la clase *Jugador* la cual tendrá la funcionalidad asociada a cada jugador. Se debe tener en cuenta que esta clase es un *agregado* de objetos de la clase *Carta*. Esta se compone de:



- El método *Repartir()* que permite crear las instancias de las 10 cartas, simulando la repartición de cartas. Es preciso tener en cuenta que en cada objeto *Carta* generado, el valor del *indice* que la identifica es aleatorio (Ver el respectivo método constructor).
- El método *mostrarCarta()* permite ver las imágenes de las cartas que tiene el jugador.
- El método *obtenerFiguras()* devuelve un texto que indica las figuras encontradas, es decir, si hubo pares, ternas, cuartas, etc. y de que nombres de cartas

El código respectivo sería el siguiente:

```
import java.util.*;
import javax.swing.*;

public class Jugador {

    private Random r;
    private Carta[] cartas;

    private Figura[] figuras;
    private NombreCarta[] cartasFigura;

    public Jugador()
    {
        //Iniciar el generador de numeros aleatorios
        r=new Random();
    }

    public void repartir()
    {
        //Instanciar las 10 cartas
       cartas=new Carta[10];
       for(int i=0;i<10;i++)
           cartas[i]=new Carta(r);
    }

    public void mostrar(JPanel pnl, boolean tapada)
    {
        //Limpiar el panel
        pnl.removeAll();
```

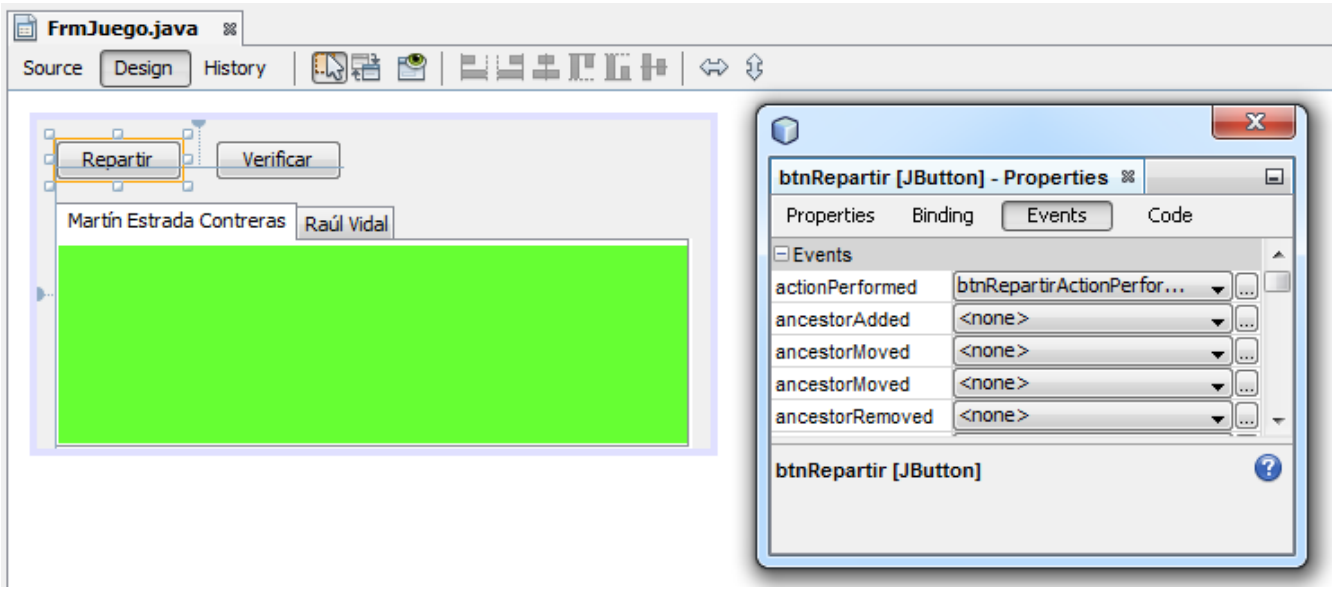
```
        //Mostrar cada carta
        for(int i=0;i<10;i++)
            cartas[i].mostrarCarta(5+i*40, 5, pnl, tapada);
        //Redibujar el panel
        pnl.repaint();
    }

    public String obtenerFiguras()
    {
        //Iniciar la estructura para almacenar las figuras
        figuras=null;
        //Declarar los contadores de cada nombre de carta
        int[] contadores=new int[13];
        //Recorrer cada carta e incrementar el contador
        //del respectivo nombre de carta
        for(int i=0;i<10;i++)
            contadores[cartas[i].obtenerNombre().ordinal()-1]++;
        //Obtener cuantas figuras se encontraron
        int totalFiguras=0;
        for(int i=0;i<13;i++)
            if(contadores[i]>=3)
                totalFiguras++;
        //Instanciar las figuras
        if(totalFiguras>0)
        {
            figuras=new Figura[totalFiguras];
            cartasFigura=new NombreCarta[totalFiguras];
            totalFiguras=0;
            for(int i=0;i<13;i++)
                if(contadores[i]>=3)
                {
                    figuras[totalFiguras]=Figura.values()[contadores[i]];
                    cartasFigura[totalFiguras]=NombreCarta.values()[i+1];
                    totalFiguras++;
                }
        }
        String mensaje="";
        if(figuras==null)
            mensaje="No hay figuras";
        else
        {
            mensaje="El jugador tiene las siguientes figuras:\n";
            for(int i=0;i<figuras.length;i++)
                mensaje+=figuras[i].name()+
                         " de "+cartasFigura[i].toString()+"\n";
        }
        return mensaje;
    }//obtenerFiguras()

}//Clase
```

En este código se pueden apreciar los siguientes detalles de implementación:

- La relación de agregación con la clase *Carta* se representa mediante un vector de estos objetos, lo cual requiere de la respectiva instrucción de declaración
- El método *mostrarCarta()* recibe como parámetros un objeto *JPanel* que contendrá las imágenes de las cartas, y un valor booleano que permita decidir si mostrar las imágenes de las cartas o la imágenes de una carta tapada.
- La estrategia para encontrar las figuras en el método *obtenerFiguras()* utiliza un vector de enteros en el que se cuanta cuanto aparece cada nombre de carta (En total serían 13 contadores). Luego se verifica que contadores son iguales o superiores a 3 para determinar la figura y concatenar los textos que así lo indiquen.

Ahora bien, para continuar con la codificación, se deben programar los métodos *Repartir()* y *Verificar()* de la clase *Juego* y que corresponderán a los eventos de los botones de comando agregados al formulario en la barra de herramientas.

La repartición de las cartas de cada jugador se hará mediante un método que responde al evento ***actionPerformed*** del botón de comando ***btnIniciar***:



El código respectivo sería el siguiente:

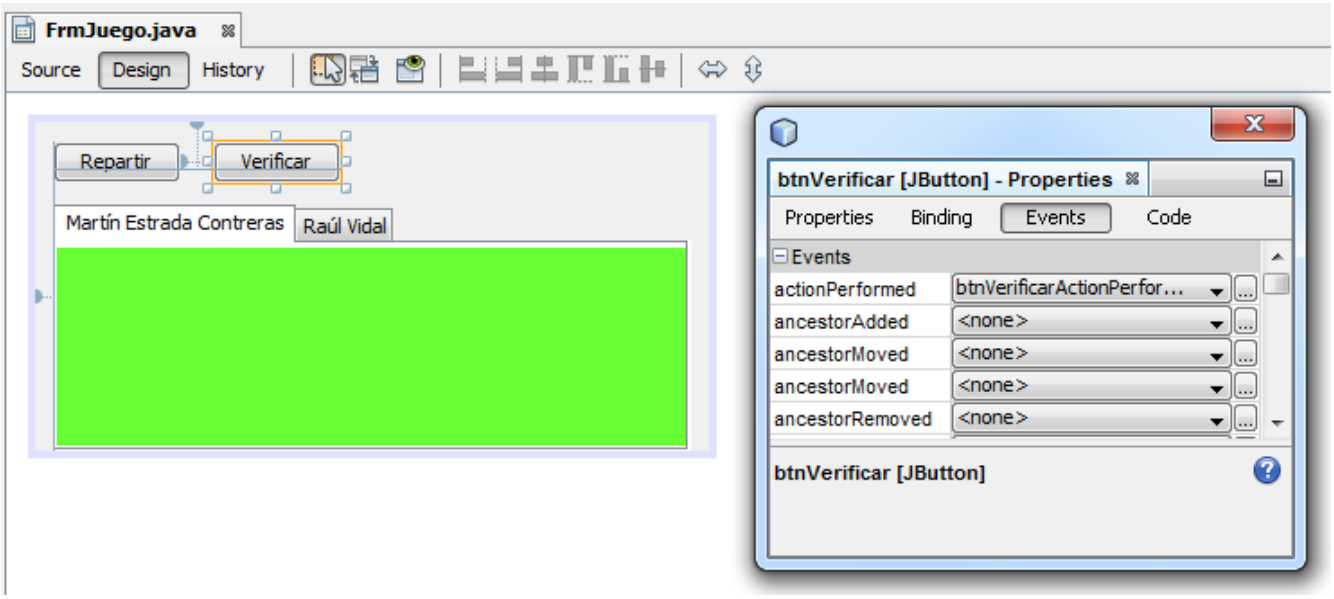
```
private void btnRepartirActionPerformed(java.awt.event.ActionEvent evt) {

        for(int i=0; i<jugadores.length;i++)
            jugadores[i].repartir();
        jugadores[0].mostrar(pnlJugador1, false);
        jugadores[1].mostrar(pnlJugador2, false);
    }
```

En este código se invocan los métodos *repartir()* y *mostrar()* de cada uno de los objetos de la clase *Jugador* cuyas instancias se debieron declarar globalmente así:

```
public class FrmJuego extends JFrame {
```

```
    Jugador[] jugadores=new Jugador[2];

    /**
     * Creates new form FrmJuego
     */
    public FrmJuego() {
        initComponents();

        for(int i=0; i<jugadores.length;i++)
            jugadores[i]=new Jugador();


    }//Constructor
```

La verificación de las figuras que hay en las cartas de los jugadores se hará mediante el método que responde al evento ***actionPerformed*** del botón de comando ***btnVerificar***:



El código respectivo sería el siguiente:

```
private void btnVerificarActionPerformed(java.awt.event.ActionEvent evt) {
        int pestaña=tpJugadores.getSelectedIndex();
        JOptionPane.showMessageDialog(new JFrame(),
                    jugadores[pestaña].obtenerFiguras());

    }
```

En este código se pregunta inicialmente por la pestaña del jugador visible para poder encontrar las figuras de las cartas que son visibles.

La ejecución de la aplicación luciría así cuando se reparten las cartas:

Cuando se verifica las figuras del jugador seleccionado, podría lucir así:

4. Basado en el ejercicio anterior, incluir la funcionalidad que permita:
   - Obtener los grupos en escalera de la misma pinta, es decir, secuencias de cartas que tengan la misma pinta. Por ejemplo: 10, J, Q y K de Pica conforman una cuarta de pica.

   - Calcular el puntaje del jugador con base en el valor de las cartas que no conforman grupos, teniendo en cuenta que las cartas ("*Ace", "Jack", "Queen"*, y *"King"* valen 10 y el resto, el respectivo número del nombre.