Universidad de los Andes
Ingeniería de Sistemas y Computación
ISIS1205 - Algorítmica y Programación por Objetos II
2006-1 – Hoja de Trabajo: Java2D y MouseListener

Los objetivos de esta hoja de trabajo son:

- Aprender a utilizar las clases de Java que permiten realizar dibujos (Java2D).
- Aprender a realizar aplicaciones que permitan la interacción con el mouse.
- Ejercitar la aplicación de los conceptos construyendo la interfaz de un juego.

Al finalizar la hoja de trabajo, habremos construido un juego de triqui en el cual las figuras se pintarán usando Java2D y que permitirá hacer clic en cualquier lugar del tablero para jugar.

Parte 1: Preparación

1.1

Importe como un proyecto de Eclipse el programa hdt_triqui que acompaña a esta guía.

1.2

Estudie la estructura de la interfaz de la aplicación. En esta hoja de trabajo vamos a cambiar la clase PanelTriqui, que permite la interacción usando únicamente botones, por una nueva clase que va a dibujar el tablero y permitir que se haga clic sobre este.

Parte 2: Preparación del nuevo tablero

En primer lugar hay que crear una nueva clase para mostrar el tablero. Esta clase será la que recibirá los clic del mouse y por lo tanto deberá implementar una interfaz que le permita ser un listener de eventos del mouse.

2.1

Cree un nuevo panel llamado PanelDibujoTriqui. Como este panel recibirá los eventos del mouse tiene que implementar la interfaz MouseListener.

```
public class PanelDibujoTriqui extends JPanel implements MouseListener
```

En este caso solamente nos interesan los eventos del clic del mouse, así que solamente implementaremos el método mouseClicked. Sin embargo es necesario ofrecer una implementación (así sea vacía) para cada uno de los 5 métodos definidos en la interfaz MouseListener (mouseClicked, mousePressed, mouseReleased, mouseEntered y mouseExited).

En la parte 4 de esta hoja de trabajo veremos cómo implementar el método mouseClicked.

2.2

El nuevo panel necesitará comunicarse con la ventana principal de la aplicación para indicarle que se realizó una jugada y con el tablero para saber qué dibujar. El constructor de la clase debe recibir entonces una referencia a la interfaz y al tablero y guardarlos en un par de atributos.

```
public PanelDibujoTriqui(InterfazTriqui ventana, Triqui tablero)
```

Agregue los atributos e implemente el constructor de la clase PanelDibujoTriqui.

2.3

Agregue los métodos dibujarTablero, dibujarLineasTablero y dibujarFichasTablero que serán usados para dibujar el estado actual del juego de triqui.

```
private void dibujarTablero(Graphics2D superficie)
private void dibujarLineasTablero(Graphics2D superficie)
private void dibujarFichasTablero(Graphics2D superficie)
```

El parámetro Graphics2D que reciben los métodos es la superficie donde debe dibujarse el tablero. En la parte 3 de esta hoja de trabajo se estudiará lo que se puede hacer con esta superficie.

2.4

Escriba el método actualizarTablero que solamente llama al método repaint() que ya estaba definido en la clase JPanel.

```
public void actualizarTablero()
{
    repaint();
}
```

El método repaint() se encarga de actualizar la visualización del panel llamando a varios métodos que saben dibujar cada uno de los aspectos del panel, como el borde o los componentes que están dentro del panel.

En el siguiente punto implementaremos uno de los métodos que es llamado por el método repaint().

2.5

Escriba el método paintComponent, que ya estaba definido en la clase JPanel, de la siguiente forma

```
public void paintComponent( Graphics g )
{
    super.paintComponent( g );

    Graphics2D g2D = ( Graphics2D )g;
    dibujarTablero( g2D );
    dibujarLineasTablero( g2D );
    dibujarFichasTablero( g2D );
}
```

El método paintComponent es llamado por la aplicación cada vez que es necesario refrescar la visualización del tablero, por ejemplo, cuando se llama al método repaint dentro de actualizarTablero(). El método paintComponent también se llama cuando hay que repintar el panel porque la aplicación se minimizó o porque otra ventana que la tapaba fue movida.

Como se están utilizando únicamente componentes SWING, se sabe que el parámetro de tipo Graphics que recibe el método paintComponent es una instancia de un objeto de tipo Graphics2D. Esto permite hacer el casting que se muestra en el código con la seguridad de que no se va a presentar una excepción de tipo ClassCastException.

2.6

En la clase InterfazTriqui modifique todos los sitios donde se hacía referencia al viejo panel de tipo PanelTriqui que ahora será de tipo PanelDibujoTriqui.

Parte 3: dibujar el tablero

En esta parte usaremos varias de las características de Java2D para dibujar el tablero de triqui. Teniendo ya la estructura construida en la parte 2, solamente será necesario implementar los métodos ya definidos para que esté lista la visualización del tablero.

3.1

Lo primero que se hará es establecer el color de fondo del tablero. Para lograr esto se dibujará un rectángulo de color cada vez que se vaya a repintar.

Implemente el método dibujarTablero de la siguiente forma.

```
private void dibujarTablero( Graphics2D superficie )
{
    Rectangle2D.Double rectFondo =
        new Rectangle2D.Double( 0, 0, getWidth( ) - 1, getHeight( ) - 1 );

    // Dibujar el fondo del tablero
    Color colorFondo = new Color( 158, 213, 255 );
    superficie.setColor( colorFondo );
    superficie.fill( rectFondo );

    // Dibujar el borde del tablero
    Color colorBordeFondo = Color.BLUE;
    superficie.setColor( colorBordeFondo );
    superficie.draw( rectFondo );
}
```

Lo primero que se hace en este método es construir un rectángulo que ocupa el panel completo: la esquina superior izquierda se encuentra en las coordenadas (0,0) del panel y tanto el ancho como el alto del rectángulo son los mismos del panel.

Como se puede ver, el nombre completo de la clase que representa al rectángulo que se dibuja es Rectangle2D.Double.

Después de tener el rectángulo es necesario dibujarlo sobre la superficie, lo cual se hace de dos formas diferentes. Primero se escoge un color para el fondo, se convierte en el color actual de la superficie (*setColor*) y luego se dibuja el rectángulo usando el método fill. Este método recibe un objeto de tipo Shape (Rectangle2D.Double implementa la interfaz Shape) y dibuja su interior sobre la superficie con el color actual establecido.

Con la tercera parte del método, en la que se llama al método draw, lo que se obtiene es que se pinte el borde de la figura y no su interior.

Después de implementar el método, ejecute la aplicación y vea cómo se ve ahora la superficie del panel. Cambie los colores del panel por los que usted prefiera.

3.2

A continuación se dibujarán las líneas del tablero implementando el método dibujarLineasTablero. En este caso se usará una técnica similar a la del punto anterior, pero en lugar de usar objetos de tipo Rectangle2D.Double se utilizarán objetos de tipo Line2D.Double.

Necesitamos cuatro líneas para dibujar el tablero de triqui, así que lo que primero hacemos es construir dos líneas horizontales y dos líneas verticales que atraviesan el panel dentro del método dibujarLineasTablero.

```
    int ancho = getWidth( );
    int alto = getHeight( );
    Line2D.Double lineaH1, lineaH2, lineaV1, lineaV2;
    lineaH1 = new Line2D.Double( 0, alto / 3, ancho, alto / 3 );
    lineaH2 = new Line2D.Double( 0, 2 * alto / 3, ancho, 2 * alto / 3 );
    lineaV1 = new Line2D.Double( ancho / 3, 0, ancho / 3, alto );
    lineaV2 = new Line2D.Double( 2 * ancho / 3, 0, 2 * ancho / 3, alto );
```

Después de construir las líneas, dibújelas sobre la superficie usando el método draw que usó en el punto anterior.

Ejecute la aplicación y verifique que ahora aparecen las líneas del tablero sobre la superficie del panel.

3.3

Ahora se añadirán algunas características adicionales a las líneas que ya se están pintando. Será necesario usar un objeto de tipo BasicStroke para definir otros atributos de la línea, como el ancho o si es una línea punteada.

Agregue al principio del método dibujarLineasTablero las siguientes líneas:

```
    BasicStroke tipoLinea = new BasicStroke( 5 );
    superficie.setStroke( tipoLinea );
```

Al llamar al método setStroke, se establece que los atributos con los que se pintarán las líneas siguientes son los establecidos por tipoLinea, que por el momento solamente especifica el grueso de

la línea (5 pixeles). Más adelante veremos como establecer otras características de las líneas.

Ejecute la aplicación y verifique que ahora las líneas se ven más gruesas.

3.4

A continuación vamos a dibujar las fichas sobre el tablero, para lo cual se deben agregar un par de métodos adicionales:

```
    private void dibujarX(Graphics2D superficie,
                          int xCentro, int yCentro, int ancho, int alto)
    private void dibujarO(Graphics2D superficie,
                          int xCentro, int yCentro, int ancho, int alto)
```

Para poder verificar los siguientes métodos, agregue al constructor de la clase Triqui las siguientes líneas para que el tablero no esté totalmente en blanco.

```
    casilla3.asignarMarca("X");
    casilla7.asignarMarca("O");
```

3.5

El método dibujarFichasTablero no tiene muchos elementos de Java2D aparte del llamado a los métodos dibujarX y dibujarO. Por este motivo se entrega completo a continuación. Es conveniente que revise los cálculos que se están haciendo para que entienda cómo se ubican las figuras dentro del panel.

```
    private void dibujarFichasTablero( Graphics2D superficie )
    {
        for( int i = 1; i <= 9; i++ )
        {
            if( !triqui.casillaVacia( i ) )
            {
                int anchoCasilla = getWidth( ) / 3;
                int altoCasilla = getHeight( ) / 3;

                int coordX = ( i - 1 ) % 3;
                int coordY = ( i - 1 ) / 3;

                int xCentro = ( coordX * anchoCasilla ) + ( anchoCasilla / 2 );
                int yCentro = ( coordY * altoCasilla ) + ( altoCasilla / 2 );

                if( triqui.obtenerMarcaCasilla( i ) == Triqui.MARCA_PC )
                {
                    dibujarO( superficie, xCentro, yCentro,
                              anchoCasilla, altoCasilla );
                }
                else
                {
```

```
                    dibujarX( superficie, xCentro, yCentro,
                              anchoCasilla, altoCasilla );
                }
            }
        }
    }
```

3.6

El método dibujarX recibe como parámetros las coordenadas del centro de una casilla y el ancho y alto de esa misma casilla. El método debe dibujar una X dentro de la casilla, centrada en las coordenadas dadas. En lo posible el método debería intentar que la X no quede tocando los bordes de la casilla.

Implemente el método usando lo que ya sabe sobre las clases Graphics2D y Line2D.Double.

Ejecute la aplicación y verifique que puede ver una X en la casilla 3.

3.7

El método dibujarO recibe como parámetros las coordenadas del centro de una casilla y el ancho y alto de esa misma casilla. El método debe dibujar un círculo dentro de la casilla, centrado en las coordenadas dadas. En lo posible el método debería intentar que el círculo no quede tocando los bordes de la casilla.

Implemente el método usando lo que ya sabe sobre la clase Graphics2D y usando la clase Ellipse2D.Double. Esta clase tiene un comportamiento muy similar al de la clase Rectangle2D.Double.

Ejecute la aplicación y verifique que puede ver un círculo en la casilla 7.

3.8

En este punto se modificarán los atributos tanto de las X como de los círculos.

En el método dibujarO construya y aplique el siguiente tipo de línea antes de dibujar la elipse.

```
        float[] lineas = new float[]{ 25, 5 };
        BasicStroke tipoLinea =
                    new BasicStroke( 3, BasicStroke.CAP_ROUND,
                                     BasicStroke.JOIN_ROUND, 0, lineas, 0 );
        superficie.setStroke( tipoLinea );
```

Esta vez se está usando un constructor de BasicStroke que tiene más parámetros. El primero de ellos sigue siendo el ancho de la línea; el siguiente indica que los bordes de las líneas se deben redondear; los dos siguientes indican cómo unir segmentos y los dos últimos indican cómo debe ser el punteado de las líneas.

Las zonas de transparentes y de color en las figuras están definidas con el arreglo *lineas*, que en este caso establece que después de 25 pixeles de color, debe haber 5 pixeles transparentes.

En el método dibujarX construya y aplique el siguiente tipo de línea antes de dibujar las líneas.

```
        float[] lineas = new float[]{ 10, 5, 5, 5 };
        BasicStroke tipoLinea =
                    new BasicStroke( 3, BasicStroke.CAP_ROUND,
                                        BasicStroke.JOIN_ROUND, 0, lineas, 0 );
        superficie.setStroke( tipoLinea );
```

Este tipo de línea es bastante parecido al del círculo: la principal diferencia es que las zonas de color son de diferentes tamaños que se van alternando.

Ejecute la aplicación después de realizar estas modificaciones y revise que todo se dibuje de la forma esperada.

Realice las modificaciones que quiera sobre los dibujos de las figuras y verifique que se dibujan tal como usted espera.

Parte 4: captura de eventos del mouse

En esta parte vamos a hacer que la aplicación responda a los eventos del mouse.

4.1

Elimine del constructor de la clase Triqui las líneas que se agregaron para inicializar el tablero con algunas figuras, de tal forma que ahora al ejecutar la aplicación no se vea ninguna figura.

4.2

En el constructor de la clase PanelDibujoTriqui hay que establecer que cada instancia de esta clase va a ser su propio listener para los eventos del mouse. Para esto hay que agregar la línea:

```
    addMouseListener( this );
```

4.3

A continuación se implementará el método mouseClicked(MouseEvent e) de la clase PanelDibujoTriqui.

Este método utiliza los métodos getX(), getY() y getButton() de la clase MouseEvent y que sirven para lo siguiente:

- getX() retorna la coordenada X del punto donde se produjo el evento. Esta coordenada se calcula dentro del objeto sobre el que se hizo clic (en este caso el panel).
- getY() retorna la coordenada Y del punto donde se produjo el evento. Esta coordenada se calcula dentro del objeto sobre el que se hizo clic (en este caso el panel).
- getButton() retorna una constante que identifica al botón que se usó para hacer clic. En este caso nos interesa verificar que el clic se haya hecho con el botón principal del mouse (MouseEvent.BUTTON1) pero en otros casos nos interesa el botón secundario (MouseEvent.BUTTON2) e incluso el tercer botón (MouseEvent.BUTTON3).

Usando estos tres métodos se puede implementar el método que se presenta a continuación, cuyo

objetivo principal es calcular en cual de las casillas se hizo clic para realizar la jugada.

```
    public void mouseClicked( MouseEvent e )
    {
        int xClic = e.getX( );
        int yClic = e.getY( );

        int anchoCasilla = getWidth( ) / 3;
        int altoCasilla = getHeight( ) / 3;

        int coordX = ( xClic < anchoCasilla ) ? 0
                      : ( xClic < 2 * anchoCasilla ? 1 : 2 );
        int coordY = ( yClic < altoCasilla ) ? 0
                      : ( yClic < 2 * altoCasilla ? 1 : 2 );

        int numCasilla = ( coordY * 3 ) + coordX + 1;

        if( e.getButton( ) == MouseEvent.BUTTON1 &&
            triqui.casillaVacia( numCasilla ) )
        {
            principal.procesaJugada( Integer.toString( numCasilla ) );
        }
    }
```

Ejecute la aplicación y verifique que ahora puede jugar triqui haciendo clic sobre las casillas.

Parte 5: actividades adicionales

A continuación se presentan actividades adicionales que se deben realizar para completar la hoja de trabajo.

5.1

Modifique las figuras dibujadas para que ahora se use un rectángulo y un triángulo. Para el triángulo puede usar la clase java.awt.Polygon, que implementa la interfaz Shape.

Cuando haya cambiado las figuras juegue con los parámetros de la clase BasicStroke para ver cómo afectan la forma en la que se dibujan las líneas.

5.2

Modifique el PanelDibujoTriqui para que ahora pueda haber una casilla "marcada". La casilla marcada es una casilla que el jugador señala haciendo clic derecho mientras decide cual va a ser su jugada.

Solamente puede haber una casilla "marcada" a la vez y apenas se realiza una jugada la marca se elimina. La casilla marcada debe dibujarse con un fondo diferente al resto de casillas.