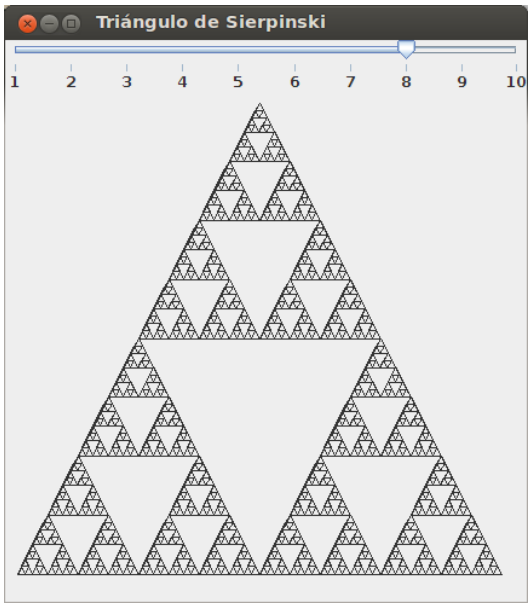
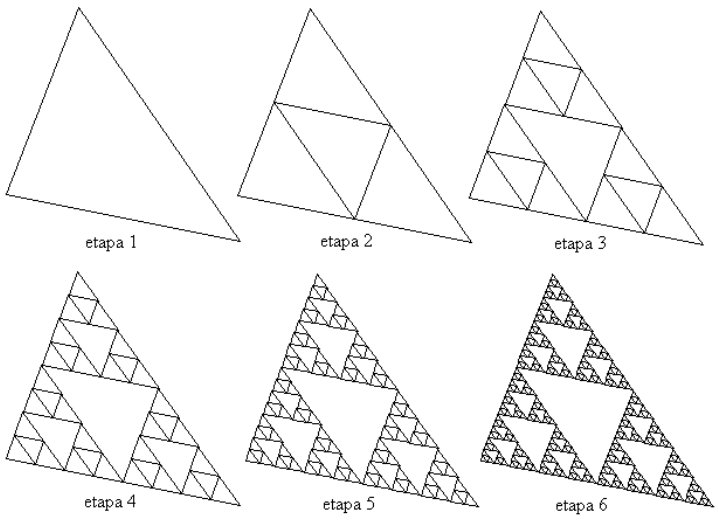
Ejercicio 8.1 (0.7 puntos)

Escribe un programa que dibuje en una ventana el triángulo de Sierpinski, permitiendo controlar el número de iteraciones mediante un componente `JSlider`.



El triángulo de Sierpinski es un fractal que puede dibujarse de manera recursiva: Para dibujar un triángulo de Sierpinski de *n* iteraciones se comienza dibujando el triángulo más externo. Después se toman los puntos medios de cada lado del triángulo. Uniendo esos tres puntos medios se obtienen cuatro triángulos más pequeños. Se dibuja un triángulo de Sierpinski de *n-1* iteraciones en tres de ellos (en el del centro no). La siguiente figura[1] ilustra el procedimiento:



1 Tomada de http://commons.wikimedia.org/wiki/File:Sierpinski_segmentos.png

Indicación: Crear una clase que extienda a `JPanel`, y que reescriba su método `paintComponent(Graphics g)` para dibujar el triángulo. La clase ha de tener un atributo privado que almacene el número de iteraciones.

```
public class PanelSierpinski extends JPanel
{
    private int numIteraciones;

    public PanelSierpinski(int numIteraciones) {
        this.numIteraciones = numIteraciones;
    }

    public void setNumIteraciones(int numIteraciones) {
        this.numIteraciones = numIteraciones;
        repaint();  // Volver a dibujar el triángulo
    }

    public void paintComponent(Graphics g) {
       // Dibujar triángulo (se deberá llamar a otro
       // método privado que sea recursivo)
    }
}
```

La ventana ha de utilizar un `BorderLayout`. En la parte norte se colocará el `JSlider` y en el centro se colocará el `PanelSierpinski`.

Ejercicio 8.2 (1.2 puntos)

Este ejercicio tiene como objetivo poner en práctica, con un sencillo ejemplo, la arquitectura modelo-vista-controlador. Supongamos que queremos mostrar gráficamente los resultados electorales de tres partidos políticos. Nuestro modelo se compondrá de tres números enteros, representando la cantidad de votos obtenidos por cada partido.

```
public class ResultadosElectorales {
 private int numVotosPartido1;
 private int numVotosPartido2;
 private int numVotosPartido3;
 ...
}
```

Por otro lado, nuestro modelo deberá almacenar una lista de observadores, que serán notificados cada vez que haya un cambio en el modelo. Estos observadores deberán implementar una interfaz `ObservadorResultados` con un único método:

```
public interface ObservadorResultados {
 void resultadosCambiados()
}
```

De este modo, nuestro modelo tendrá un atributo adicional, que será la lista de observadores (implementada, por ejemplo, como un `ArrayList`)

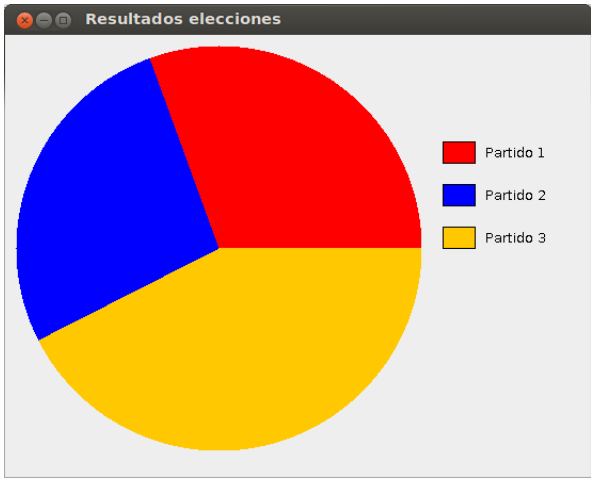
```
private ArrayList<ObservadorResultados> listaObservadores;
```

El modelo deberá proporcionar un método que permita a un observador (es decir, cualquier clase que implemente la interfaz `ObservadorResultados`) registrarse como tal en el modelo, para que el modelo le "avise" cada vez que haya un cambio.

```
public void añadirObservador(ObservadorResultados o)
```

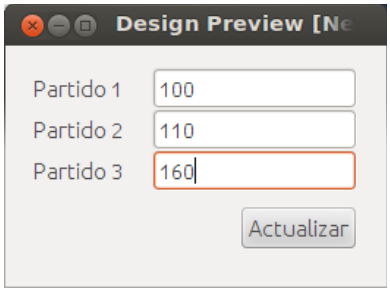
Por último, el modelo deberá proporcionar métodos de modificación a cada uno de los atributos `numVotosPartido1`, `numVotosPartido2`, `numVotosPartido3`. Estos métodos, además de modificar el atributo de la clase correspondiente, deberán avisar a todos los observadores registrados del hecho de que el modelo ha cambiado. Para ello, se recorrerá el `ArrayList` de `ObservadorResultados`, y se llamará al método `resultadosCambiados()` de cada uno de ellos.

Una vez creado el modelo, pasamos a implementar las vistas. En este ejercicio tendremos dos vistas: una de ellas mostrará los resultados de las elecciones en un diagrama de sectores, y la otra los mostrará mediante un diagrama de barras.



Las dos vistas han de extender la clase `JFrame`, y han de implementar la interfaz `ObservadorResultados`. El método `resultadosCambiados()` de cada uno se limitará a llamar al método `repaint()` de la ventana correspondiente, para que se actualicen cada uno de los dibujos.

Por último, se implementará un controlador que contenga tres campos de texto, para introducir los votos de cada partido. Cuando el usuario pulse sobre el botón Actualizar, se modificará el modelo.



Tanto el controlador como las vistas deberán contener una referencia al modelo que modifican/observan. Pueden recibir esas referencias en sus respectivos constructores. El método `main()` es el responsable de crear el modelo y pasárselo a las dos vistas y el controlador, que también serán construidos en este método.