# PRÁCTICA 6.- Librería para Graficación

## Objetivo

Crear una librería que provea las clases necesarias para llevar a cabo la graficación de figuras en 2D.

Objetivos específicos:

- Generar y habilitar una librería con los elementos necesarios para realizar gráficos en 2D.
- Usar dicha librería en un ejemplo de graficación.

## Introducción

Una librería es una colección de clases que pertenecen a una misma categoría y que quedan disponibles para ser usadas en cualquiera de nuestras aplicaciones con tan solo incluirla en el proyecto. Esto quiere decir que si generamos una clase que funcione, libre de errores, ya no se hace necesario tener que volverla a escribir en una nueva aplicación, solamente incorporarla y hacerla parte de nuestro proyecto actual.

Las librerías son una forma de reutilización de código facilitando a los desarrolladores de aplicaciones el avanzar en otras partes del proyecto. De hecho, las librerías pueden compartirse y al usuario solo le interesará saber los tipos de constructores de la clase, los métodos quela clase a ser usada tiene, los parámetros necesarios para utilizar dichos métodos y el tipo de dato de retorno.

El lenguaje Java nos provee de un número importante de librerías para facilitarnos el trabajo. En todos los programas que desarrollamos, ya sea que nosotros incluyamos alguna librería directamente, también el compilador nos ayuda incorporando de manera automática aquellas que se necesiten.

En esta práctica vamos a ver cómo crear nuestras propias librerías y además dejarlas disponibles para ser usadas en otras aplicaciones. En específico trabajaremos con un grupo de clases que nos permitan dibujar figuras tales como círculos, rectángulos y triángulos, las agruparemos dentro de una librería y luego, usaremos nuestra librería, en una aplicación.

## Correlación con los temas y subtemas del programa de estudio vigente

La práctica propuesta se relaciona directamente con la Unidad III del temario "Componentes y Librerías" y específicamente con el punto 3.5. Creación y uso de paquetes/librerías", ya que se creará una librería y se usará en un proyecto.

## Material y equipo necesario

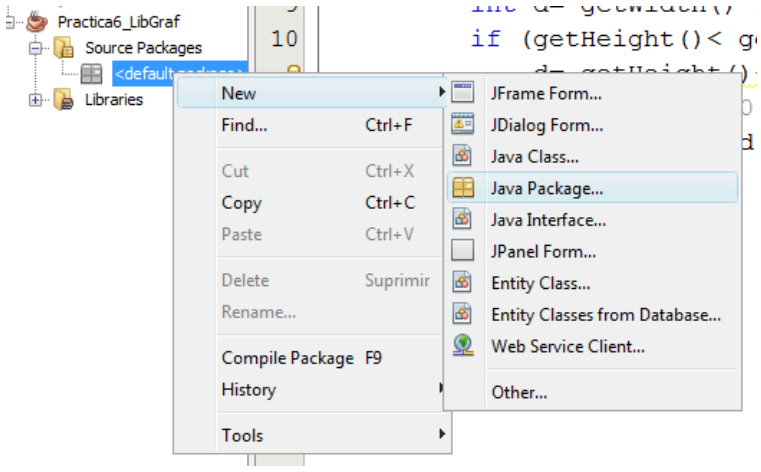
- Computadora o Laptop
- Software: Netbeans versión 7.2 en adelante.

## Metodología

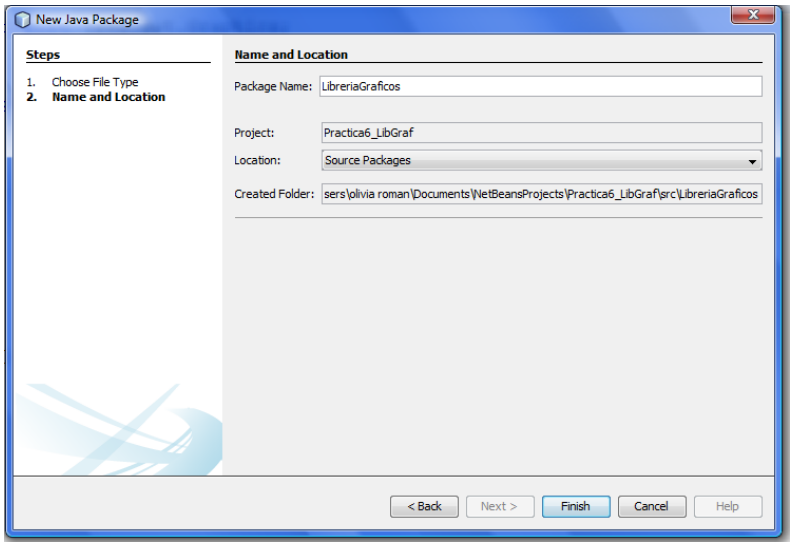
Crearemos una librería que nos de soporte para graficar varias figuras en 2D (dos dimensiones). Definiremos las clases Figura, Círculo, Rectángulo y Triángulo. Tendremos una jerarquía de clases en la que la clase padre o clase base es la clase Figura y ella heredará sus atributos y métodos a las otras clases que se derivan de ella, es decir que heredan de la clase Figura.

**Paso 1**. Vamos en primera instancia a crear nuestra librería. Creamos proyecto nuevo con nombre *Practica6_LibGraf*. Se debe desmarcar la casilla de verificación Create Main Class.

**Paso 2**. Llevamos el cursor a la ventana de exploración de proyectos y seleccionamos *Source Packages* o en *<default packages>*, con botón derecho activamos menú auxiliar y escogemos *new*. En la ventana que se abre seleccionamos *Java Packages*.



Se despliega una nueva ventana para dar nombre al paquete. Lo nombramos *LibreriaGraficos* y luego botón *finish*.




**Paso 3**. Llevamos el cursor a la ventana de proyectos y seleccionamos donde dice *LibreriaGraficos*. Hacemos luego clic con botón derecho para añadir una nueva clase *Java Class*. Escoja *new* y luego *Java Class*. Le damos de nombre *Figura*.

```
package LibreriaGraficos;

public class Figura {



}
```

La clase *Figura* es la clase superior que heredará a la clase *Círculo*, *Rectángulo* y *Triángulo*. Vamos a activar la posibilidad de que las figuras puedan ser dragadas, es decir seleccionadas y también desplazadas por la pantalla usando el mouse o el trackpad. Para ello se implementarán las interfaces *MouseListener* y *MouseMotionListener*. Se sobrescribirá el método *paint()* y además se definirá una nueva propiedad que indique si la figura está seleccionada o no lo está, que será usada para informar al método *paint()* si a la hora de repintar le cambia el color o no a la figura.

**Paso 4**. Modifique la clase Figura con el código siguiente:

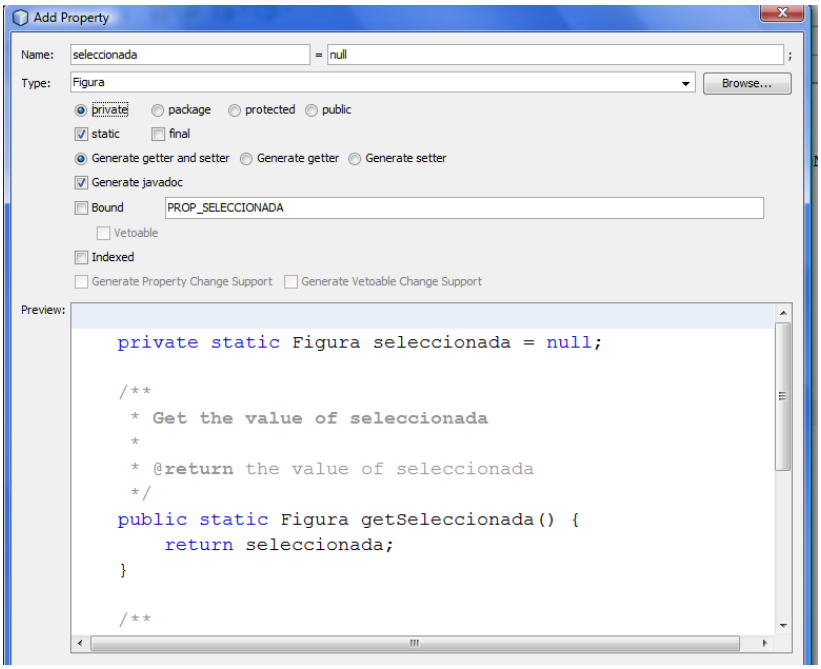
```
public class figura extends JComponent implements MouseListener,MouseMotionListener{
  private int xi,yi;
  public figura() {
    setSize(100,100);
    addMouseListener(this);
    addMouseMotionListener(this);
  }
```

Estamos haciendo que la clase *Figura* herede de la clase *JComponent* y que implemente las interfaces *MouseListener* y *MouseMotionListener* que son las que nos facultan para controlar eventos de mouse. Después de escribir este código, es posible que esté marcando errores. Éstos se pueden corregir primero activando botón derecho y seleccionando *Fix Imports*, luego llevando el cursor hacia el botón pequeño rojo en el margen izquierdo y hacer clic en él, escoger *implement all abstract methods*. Se añadirán las estructuras de TODOS los métodos de las dos interfaces. Habrá que eliminar las líneas que empiezan con throw… (una línea de código se elimina pulsando simultáneamente Ctrl + E).

Las variables xi y yi servirán para mantener las coordenadas en donde se realizó el clic del mouse y marcarán uno de los puntos para dibujar la figura. Además observe que es necesario añadir explícitamente los dos escuchadores en el constructor *addMouseListener(this)* y *addMouseMotionListener()*. Si no se añaden, su programa no funcionará.

**Paso 5**. Añadimos una nueva propiedad a la clase y le pondremos de nombre "seleccionada".

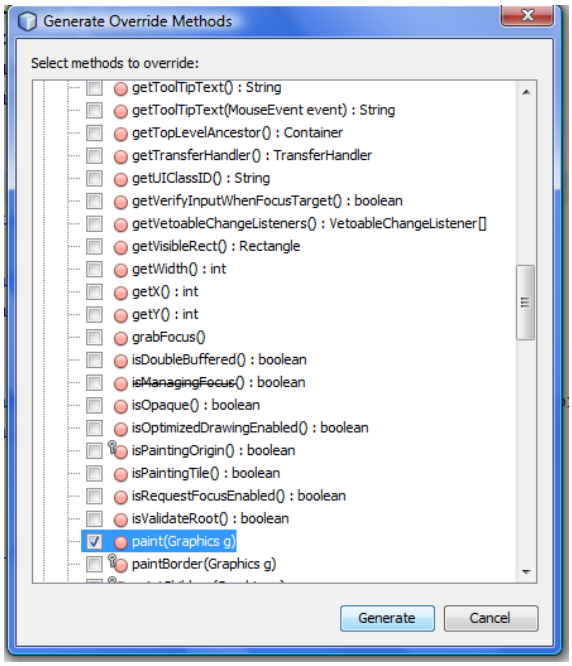
Debajo del constructor, pulsar el botón derecho del mouse, escoger *Insert Code* y en la ventana que se abre buscar *Add Property* y seleccionarla. Se abre una nueva ventana donde se capturan los datos de la nueva propiedad. Name: *seleccionada*; después del signo = se escribe *null*, en Type se escribe *Figura*, hacer clic en *private* y en *static*.




El código que habilita la nueva propiedad se añade automáticamente y se muestra enseguida:

```
private static Figura seleccionada = null;

public static Figura getSeleccionada() {
   return seleccionada;
}
public static void setSeleccionada(Figura seleccionada) {
   Figura.seleccionada = seleccionada;
}
```

**Paso 6**. Se sobrescribirá el método *paint()* del componente. Para hacer esto pulse el botón derecho del mouse estando el cursor abajo del constructor de la clase Figura. Seleccione *Insert Code*, y luego escoja *Override Method* (sobrescribir método), en la nueva ventana desplegada empiece a teclear la palabra *paint* y se mostrará una caja de texto que estaba oculta y además se irán mostrando los métodos que empiecen con la letra p y luego se reducen a los que tiene pa y así hasta que localice el método *paint()*. Marque la casilla de la izquierda de este método y luego clic en el botón *Generate*.




El código añadido automáticamente es el siguiente:

```
public void paint(Graphics g) {
    super.paint(g);
  }
```

Modificamos el método *paint()* con las instrucciones que permitan cambiar de color a una figura cuando ésta ha sido seleccionada.

```
public void paint(Graphics g) {
    super.paint(g);
    if(Figura.getSeleccionada()== this)
      g.setColor(Color.red);
    else
      g.setColor(Color.black);
  }
```

El método *paint()* se ejecuta para cada figura que tengamos en nuestro panel. La figura que se esté repintando está en ese momento estará identificada con el apuntador *this*, el cual es controlado por el software. Así, si coincide que al estar repintando una figura con *this* apuntándole y si *getSeleccionada()* devuelve true, entonces se debe pintar de color rojo. En caso contrario significa que no está seleccionada y se pinta de negro.

**Paso 7**. Se hace necesario adecuar el método *setSeleccionada()* para que primero respalde (guarde), la figura anteriormente seleccionada y se pueda repintar como no seleccionada y la nueva, identificada con el nombre de *aseleccionada*, se seleccione y se pinte con el color que le toca por estar seleccionada.

```
public static void setSeleccionada(Figura aseleccionada) {
    Figura auxi = seleccionada; // seleccionada actual se gurda en auxi
    seleccionada = aseleccionada; // la nueva figura se guarda en seleccionada
    if(auxi!= null)
      auxi.repaint(); // llama a repintar a auxi, le regresa su color
    seleccionada.repaint(); // llama a repintar a seleccionada, le cambia su color
    }
```

**Paso 8**. Procederemos a poner el código que controle el evento de clic con el mouse sobre una figura y por ende hacer que cambie de color. Además incluiremos las instrucciones para dragar las figuras por el área de dibujo controlando el evento *mouseDragged()*, repintando la figura dragada en la posición en donde se encuentra el mouse.

```
public void mousePressed(MouseEvent e) {
    xi = e.getX(); // guarda posición x donde se hizo click
    yi = e.getY();// guarda posición y
    setSeleccionada(this); // se manda marcar y por consecuencia repintar
  }

public void mouseDragged(MouseEvent e) {
    int xf, yf, increX,increY;
    xf = e.getX(); // posición x donde anda el mouse
    yf = e.getY(); // posición y donde anda el mouse
    increX = xf - xi; // cambio de valor en eje x
    increY = yf - yi; // cambio de valor en eje y
    setLocation(getX()+ increX,getY()+increY);
  }
```

**Paso 9**. Añadimos una nueva clase con nombre *Circulo* a nuestra *LibreriaGraficos*. A estas alturas Usted ya sabe cómo realizarlo. El código que aparece es:

```
package LibreriaGraficos;

public class Circulo {

}
```

Modificamos esta clase para que herede de *Figura* y además sobrescribimos su método *paint().*

```
package LibreriaGraficos;
import java.awt.Graphics;


public class Circulo extends Figura{
    @Override
   public void paint(Graphics g) {
      super.paint(g);
      int d= getWidth()-1;
      if (getHeight()< getWidth()) // pregunta la dimension menor
         d= getHeight()-1;
      //g.drawOval(0,  0,d, d);
      g.fillOval(0,  0,d, d);
   }
}
```

**Paso 10**. Añadimos ahora una nueva clase para *Rectángulo*. La cual también hacemos que herede de *Figura* y modificamos su método *paint()*.

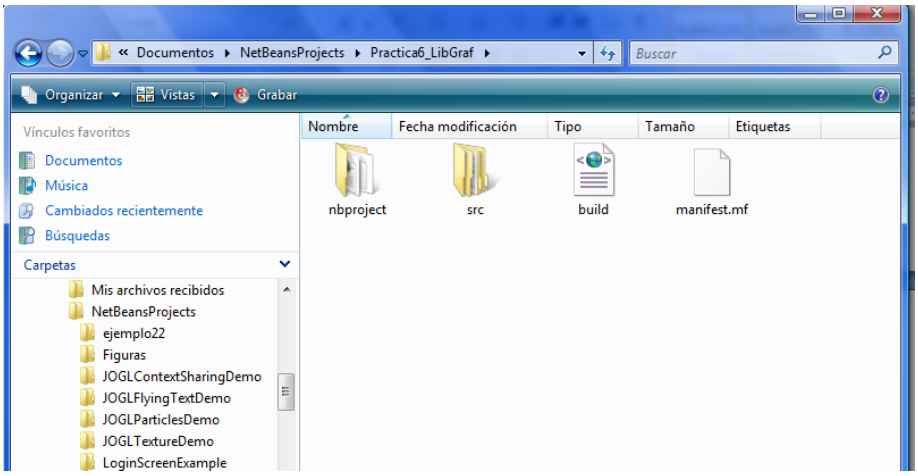
```
import java.awt.Color;
import java.awt.Graphics;
public class Rectangulo extends Figura {
   @Override
   public void paint(Graphics g) {
      super.paint(g);
      g.setColor(Color.GREEN);
      //g.drawRect(0, 0, getWidth()-1, getHeight()-1);
```

```
        g.fillRect(0, 0, getWidth()-1, getHeight()-1);
    }
}
```

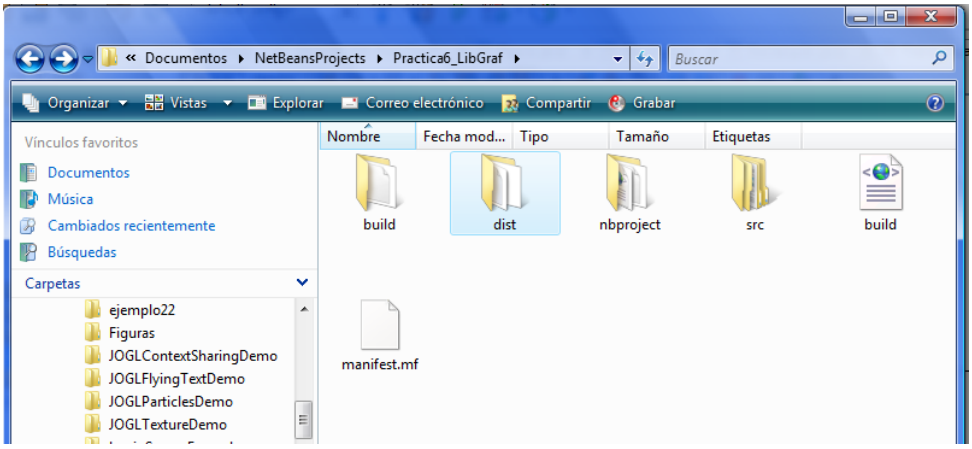
**Paso 11**. Una nueva clase mas a la librería con nombre *Triangulo*. Se modifica de manera semejante a las dos clases anteriores. Para dibujar el triángulo se identifican los tres puntos de sus vértices. Los puntos de los vértices se almacenan en dos arreglos, todas las coordenadas ***x*** en el arreglo *xPoints[]* y las ordenadas ***y*** en el arreglo *yPoints[]*. Un valor del arreglo *xPoints* junto con el valor en *yPoints*, en la misma posición (subíndice) definen uno de los vértices del triángulo. Los arreglos de valores x y y, se usan para pintar el triángulo como un polígono.

```
import java.awt.Graphics;
public class Triangulo extends Figura{
    @Override
    public void paint(Graphics g) {
        super.paint(g);
        // los vértices del triángulo son uno de xPoints con uno de yPoints
        int xPoints[] = {0,0,getWidth()-1};
        int yPoints[] = {0,getHeight()-1, getHeight()-1};
        //g.drawPolygon(xPoints, yPoints, 3);
       // se conectan las líneas formando un polígono y se rellena al usar el fill
        g.fillPolygon(xPoints, yPoints, 3);
    }
}
```

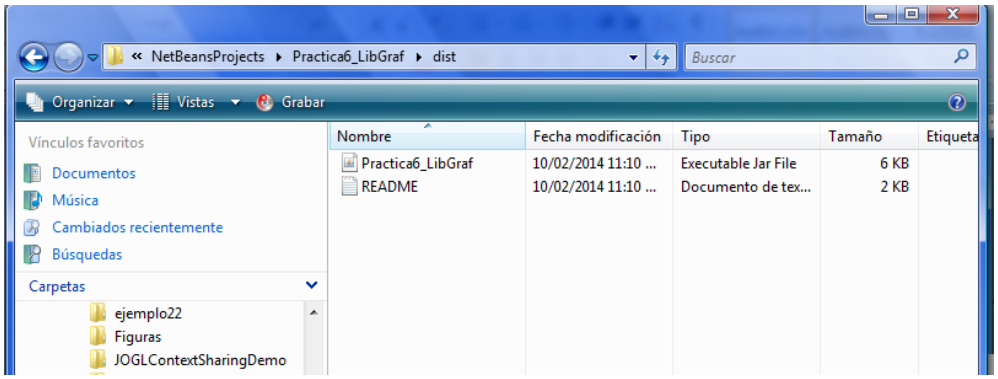
**Paso 12**. Vamos a explorar la carpeta del proyecto para constatar que NO tenemos aún una carpeta de nombre *dist* y por lo tanto aún seguimos sin ningún archivo con extención jar. Vamos al explorador de Windows, a Documentos, Netbeans Proyects, buscamos la carpeta con nombre *Practica6_LibGraf* y encontramos lo siguiente:




Regresemos a NetBeans y compilemos nuestro proyecto. Para hacerlo lleve el cursor hasta la ventana de proyectos y seleccione *Practica6_LibGraf*. Luego botón derecho y selecciona *Build*. Después de hacerlo regresamos a la ventana del proyecto y vemos qué estado tiene ahora:

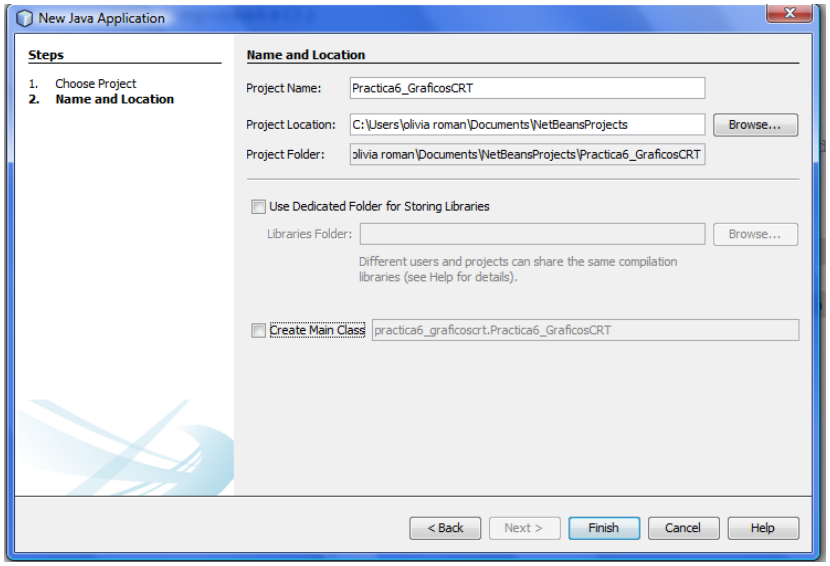



Podemos observar que ya tenemos una nueva carpeta producto de la compilación. Los archivos que tiene son:

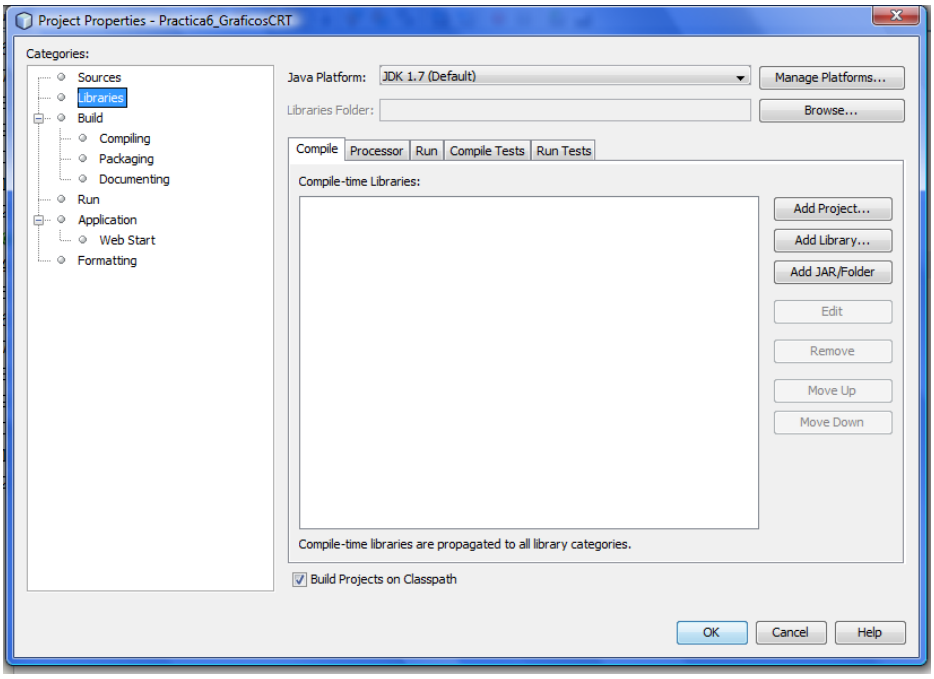



Ya tenemos el ejecutable Jar File de nombre *Practica6_LibGraf*. Con esto YA CREAMOS una nueva librería.

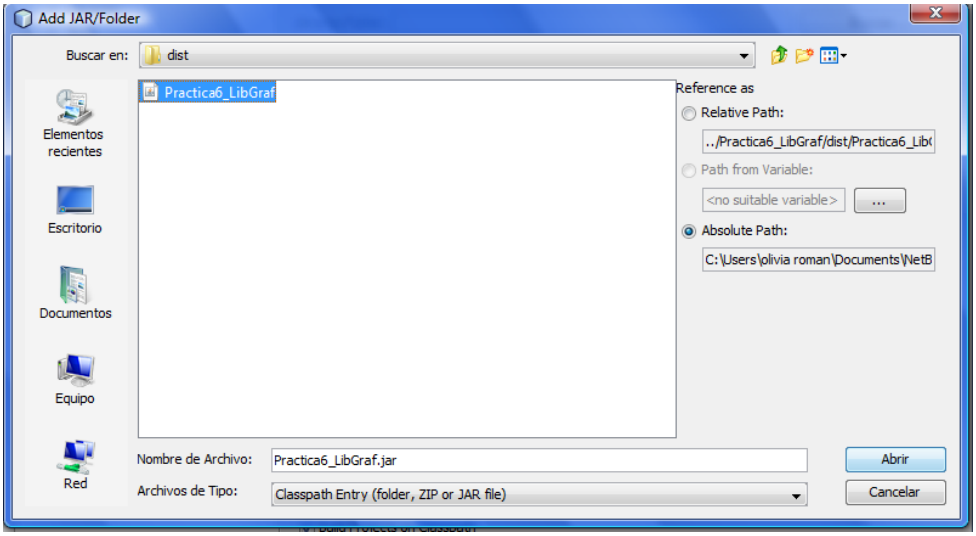
**Paso 13**. Vamos ahora a utilizar nuestra librería en un proyecto nuevo. Abra un nuevo proyecto con nombre *Practica6_GraficosCRT*, desmarcando la casilla del Create Main Class.




**Paso 14**. Vamos a la ventana de proyectos y seleccionamos *Practica6_GraficosCRT*, luego botón derecho y vamos hasta el fondo de la nueva ventana a *properties* y damos clic ahí. Se muestra una nueva ventana:




Para ver exactamente esta ventana debe hacer clic en la palabra *Libraries* de la ventana *Categories*. Enseguida dar clic en el botón de la derecha *Add JAR/Folder*. Recorremos la ruta hasta donde esta el archivo jar producto de la compilación de nuestras clases: Documentos/NetBeans Proyects/Practica6_LibGraf/dist y encontramos el archivo Jar y damos clic en botón *Abrir* y luego en *Ok*.



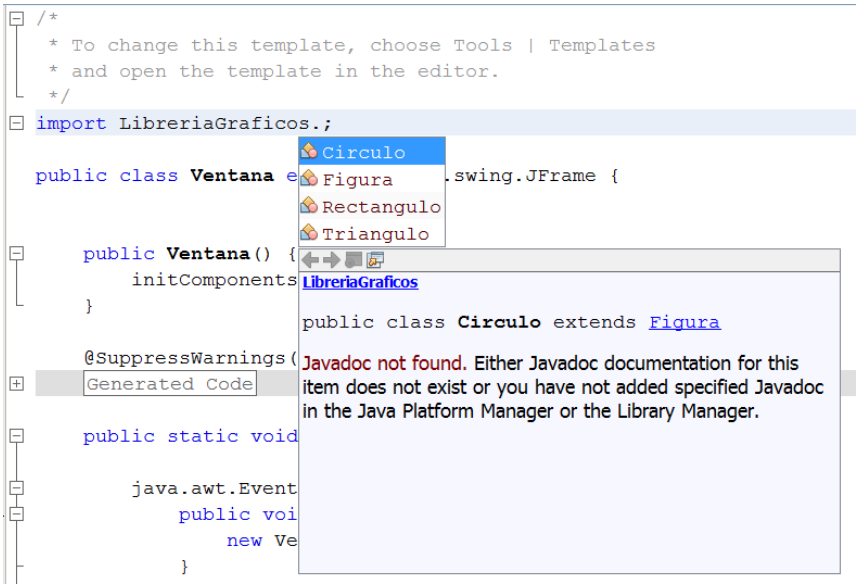
Después de esta acción podemos abrir la carpeta librería en la misma ventana de proyectos y veremos que ya se encuentra la librería *Practica6_LibGraf* incorporada al proyecto.

**Paso 15**. Añadimos un *JFrame Form* al proyecto. Damos el nombre de *Ventana*.

**Paso 16**. En la ventana de código, en la parte superior escribimos

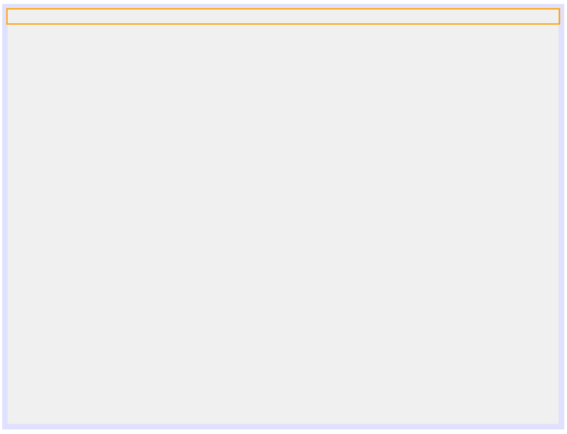
```
import LibreriaGraficos.*;
```

Con esta instrucción ponemos disponibles en el proyecto las clases *Figura*, *Circulo*, *Rectangulo* y *Triangulo* que tenemos en la librería creada. Puede verse que al momento de empezar a escribir LibreriaGraficos, si utiliza la combinación de teclas para autocompletado Ctrl + Barra espaciadora, se autocompleta y al momento de poner el punto se abre una ventana con las cuatro clases disponibles para poder escoger. Escribimos un asterisco después del punto para disponer de todas las clases definidas en la librería.




**Paso 17**. Trabajamos con el formulario *Ventana*.

- Cambie el *Layout* del formulario a *Border Layout*. Botón derecho dentro de la forma, luego *Set Layout* y enseguida *Border Layout*.
- Añada un componente panel que será utilizado para poner los botones de dibujo. Uno para cada figura. Cambie su *Layout* a *Flow Layout*. Se mostrará en la parte superior de la forma.



- Añada un segundo panel a la forma. Este panel será el área de dibujo de las figuras y donde se podrán dragar. El panel se acomoda en el espacio disponible.
- Cambie los nombres de los paneles a jpnBotones ya jpnDibujo.

**Paso 18**. Poner tres botones en el panel uno, el panel superior. Cambie el texto a *Rectángulo*, *Círculo* y *Triángulo*. Los nombres para los botones son: *jbtRectangulo*, *jbtCirculo* y *jbtTriangulo*.


**Paso 19**. Programaremos de forma que al hacer clic en alguno de los tres botones, se pinte la figura correspondiente en el panel dos, en un lugar escogido aleatoriamente.

```
public class Ventana extends javax.swing.JFrame {
  Random rand;
  public Ventana() {
    initComponents();
```

```
            rand = new Random();
        }

    private void jbtRectanguloActionPerformed(java.awt.event.ActionEvent evt) {
        Rectangulo rect = new Rectangulo();
        rect.setSize(50,60);
        rect.setLocation((int)(rand.nextFloat()*getWidth()),(int)(rand.nextFloat()*
getHeight()));
        jpnDibujo.add(rect);
        jpnDibujo.repaint();

    }
```

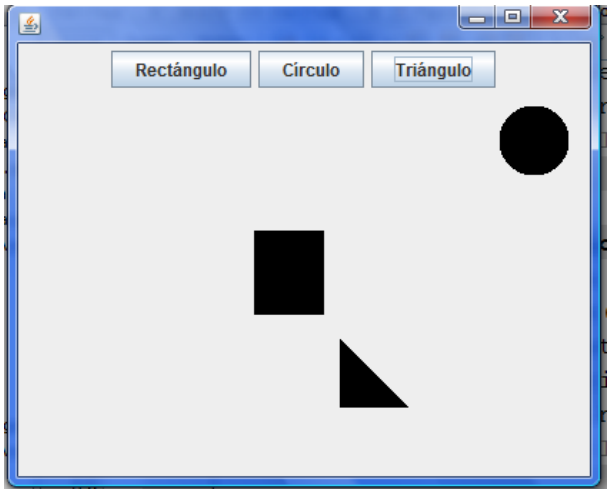
El código para el botón del círculo es el siguiente:

```
   private void jbtCirculoActionPerformed(java.awt.event.ActionEvent evt) {
    Circulo circu = new Circulo();
    circu.setSize(50,50);
                circu.setLocation((int)(rand.nextFloat()*getWidth()),(int)(rand.nextFloat()*
        getHeight()));
    jpnDibujo.add(circu);
    jpnDibujo.repaint();

  }
```

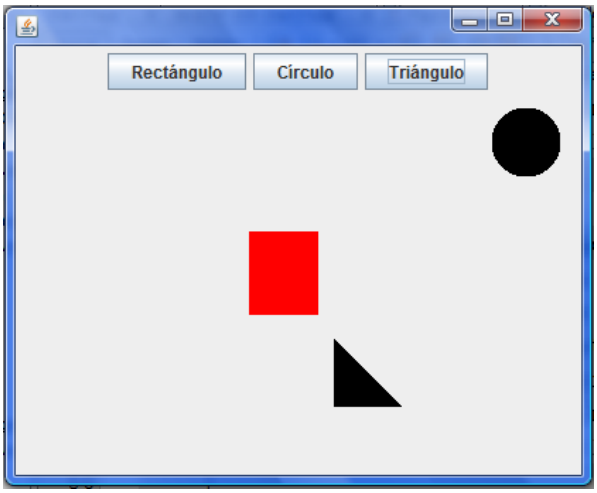
Y el código para el triángulo es:

```
  private void jbtTrianguloActionPerformed(java.awt.event.ActionEvent evt) {
        Triangulo trian = new Triangulo();
        trian.setSize(50,50);
                trian.setLocation((int)(rand.nextFloat()*getWidth()),(int)(rand.nextFloat()*
        getHeight()));
        jpnDibujo.add(trian);
        jpnDibujo.repaint();

}
```

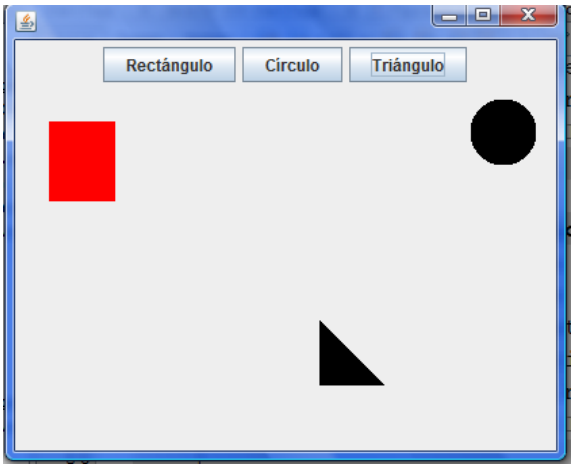
Al ejecutar el programa y hacer clic una vez en cada botón tenemos el resultado siguiente:




Si hace clic sobre alguna de las figuras, entonces se selecciona y cambia a color rojo:




Y si se draga se puede reubicar en el área de dibujo:




En esta imagen puede verse el rectángulo en una nueva posición.

**Sugerencias Didácticas**

Se recomienda el que los alumnos tengan los conceptos de herencia de manera que al hacer esta práctica puedan ver el uso de la clase figura y la ventaja que brinda esta clase superior. Es conveniente que se refresquen también los conceptos de interface y cómo este mecanismo permite la herencia múltiple. De ser posible se realicen ejercicios básicos de aplicación de ambos mecanismos de herencia.

**Reporte del alumno (resultados)**

El alumno debe mostrar evidencias de haber realizado la práctica completa, así como la de haber creado su propia librería.

Se sugiere pedir al alumno la creación de una nueva librería y su uso en una nueva aplicación.

Pedir al alumno que modifique el programa para que las figuras no se pinten en una posición aleatoriamente generada sino que se espere a dar un clic en el área de dibujo y ahí sea dibujada la figura escogida previamente al haber dado clic en el botón de la figura deseada.

Incluir en el reporte sus conclusiones y recomendaciones.

## Bibliografía preliminar


1. Java The Complete Reference Eighth Edition
   Herbert Schildt

   Oracle Press 2011

2. Java 2 Interfaces Gráficas y Aplicaciones para Internet
   Francisco Javier Ceballos
   Alfaomega- RAMA 2ª. Edición 2007

3. En la página oficial de Java:
   docs.oracle.com/javase/tutorial/java/