# JFrame, JPanel and Drawing Basics:

In [1]:
```
import java.awt.*;
```

In [2]:
```
import javax.swing.*;
```

## Create JFrame

Create a new JFrame object and set title of window:

In [3]:
```
JFrame frame = new JFrame("My Frame");
```

Set your preferred size for this frame. Dimension is a java.awt object that stores a width and a height:

In [4]:
```
frame.setPreferredSize(new Dimension(500,500));
```

Keep user from resizing the window:

In [5]:
```
frame.setResizable(false);
```

`pack()` is a method to set frame and its contents to the preferred sizes and constraints

In [6]:
```
frame.pack();
```

Set frame to visible:

In [7]:
```
frame.setVisible(true);
```

## Add JPanel to JFrame

In [8]:
```
JPanel panel = new JPanel();
```

In [9]:
```
panel.setPreferredSize(new Dimension(500,500));
```

Add the JPanel to our JFrame

In [10]:
```
frame.add(panel);
```

Out[10]:
```
javax.swing.JPanel[,0,0,0x0,invalid,layout=java.awt.FlowLayout,ali
gnmentX=0.0,alignmentY=0.0,border=,flags=9,maximumSize=,minimumSiz
e=,preferredSize=java.awt.Dimension[width=500,height=500]]
```

Pack the JFrame again to adjust panel size and position to preferred size

In [11]:
```
frame.pack();
```

## Drawing to the frame or panel's graphics object

We could use javax.swing's built in methods for drawing to the panel and frame. You can look them up, they are a family of methods related to `paintComponent(Graphics g)`. They come with lots of additional functionality. For example they will automatically repaint the panel under certain window and operating system events (e.g. uncovering a window behind another). However all these extra functionality just makes our game slower and is unneccesary. We will stop the built in drawing functionality and use our own basic (yet effective) method.

Ignore auto repaint events from JFrame and JPanel:

In [12]:
```
frame.setIgnoreRepaint(true);
panel.setIgnoreRepaint(true);
```

Get the graphics object of the panel, so we can paint to it. This creates a buffer where we can add paint methods. As with all buffer objects remember to close the stream/dispose of them after use.

In [13]:
```
Graphics g = panel.getGraphics();
```

Cast the Graphics object to a Graphics2D object, which has more relevant methods we will use for 2D drawing.

In [14]:
```
Graphics g2 = (Graphics2D) g;
```

Now we can draw to this graphics object with various methods.

In [15]:
```
g2.drawString("Hello World!", 50, 50);
```

In [16]:
```
g2.drawString("How are you?", 100, 100);
```

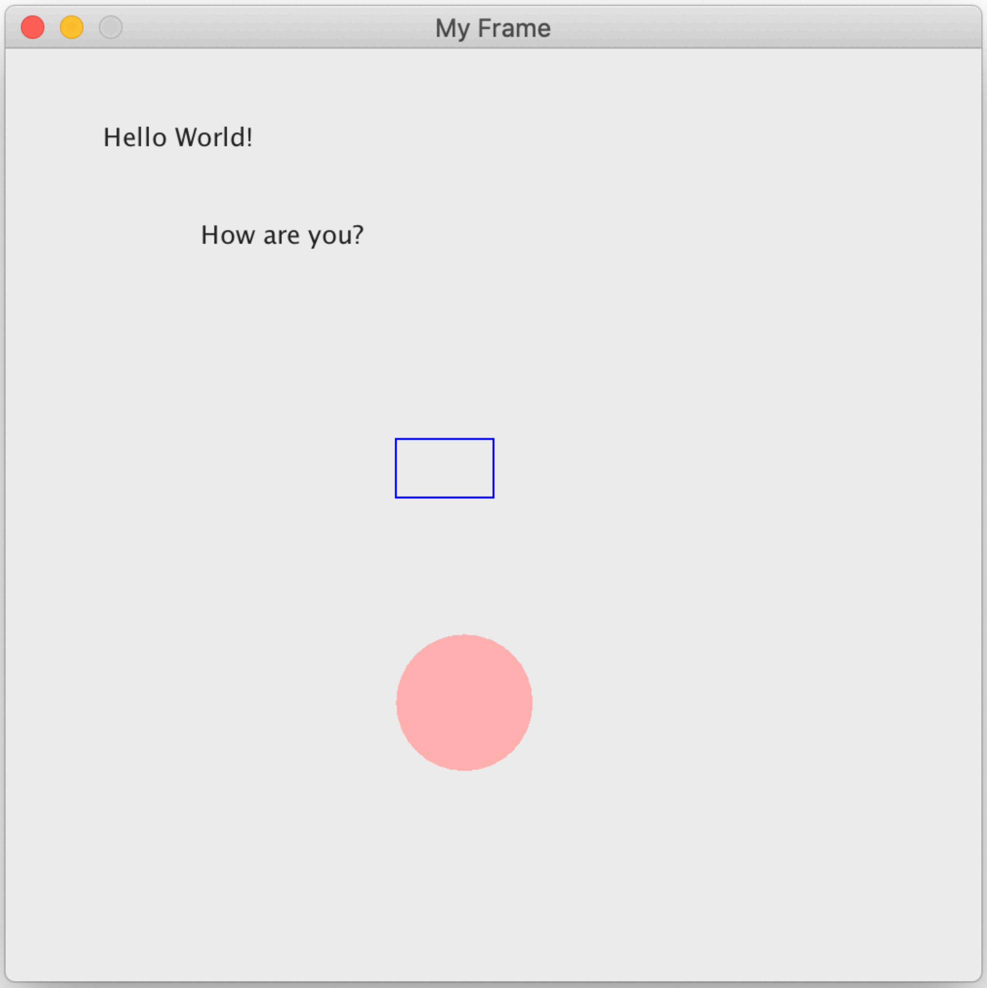
In [17]:
```
g2.setColor(Color.BLUE);
g2.drawRect(200,200,50,30);
```

In [18]:
```
g2.setColor(Color.PINK);
g2.fillOval(200,300,70,70);
```

Finally dispose graphics object to close the buffer stream:

In [19]:
```
g2.dispose()
```

This is what you should see:



Hello World!

How are you?

## Code:

```
import java.awt.*;

import javax.swing.*;

public class Tut1 {

    public static void main(String[] args) {

        JFrame frame = new JFrame("My Frame");
        frame.setPreferredSize(new Dimension(500, 500));
        frame.setResizable(false);
        frame.pack();
        frame.setVisible(true);

        JPanel panel = new JPanel();
        panel.setPreferredSize(new Dimension(500, 500));

        frame.add(panel);
        frame.pack();

        frame.setIgnoreRepaint(true);
        panel.setIgnoreRepaint(true);

        Graphics g = panel.getGraphics();
        Graphics g2 = (Graphics2D) g;

        g2.drawString("Hello World!", 50, 50);
        g2.drawString("How are you?", 100, 100);

        g2.setColor(Color.BLUE);
        g2.drawRect(200, 200, 50, 30);

        g2.setColor(Color.PINK);
        g2.fillOval(200, 300, 70, 70);

        g2.dispose();

    }

}
```