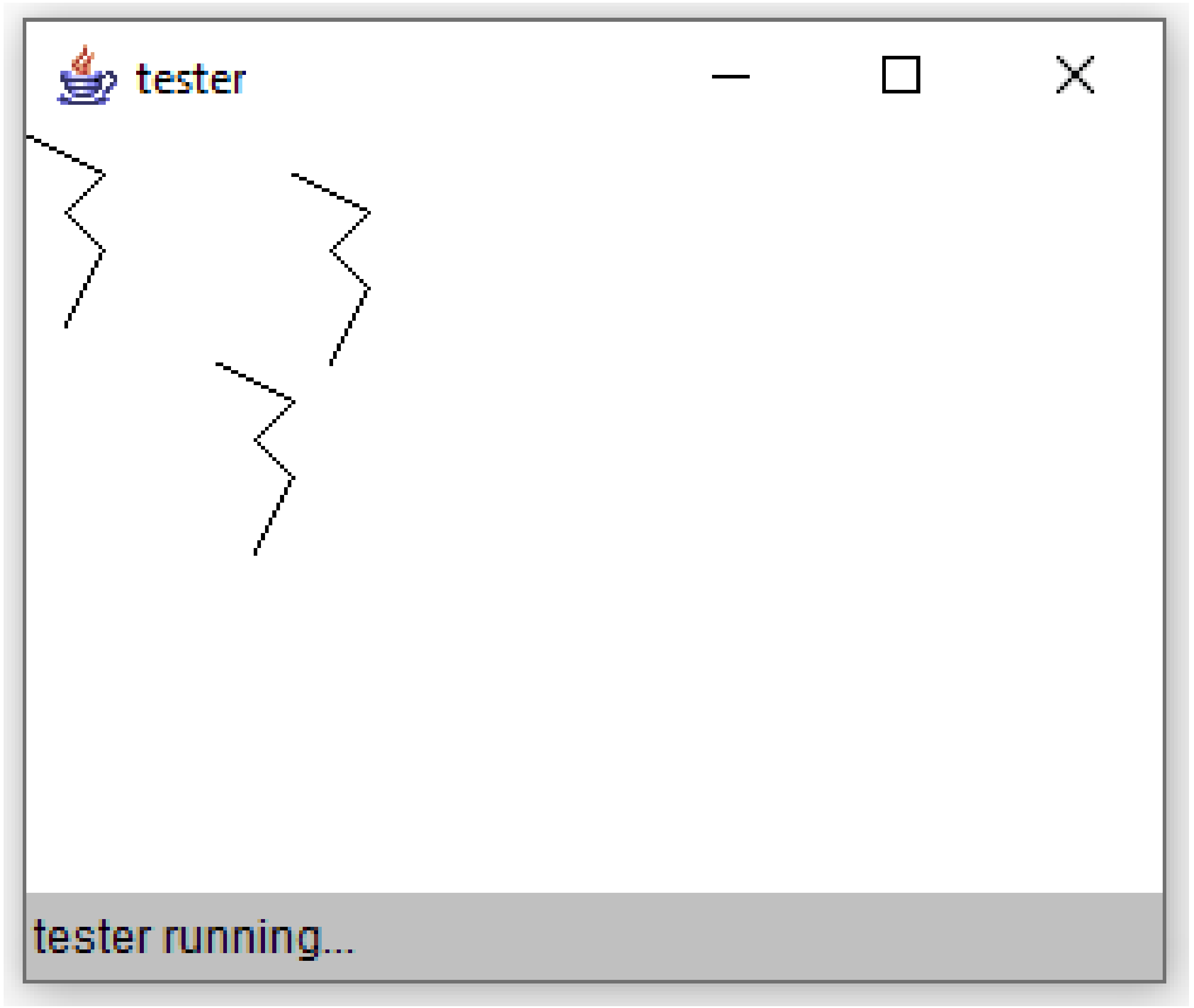
Graphics Methods

How to repeat things across the screen

Suppose that you want this really, really, really, great picture.

It is hard to imagine why you wouldn't want this picture.



tester running...

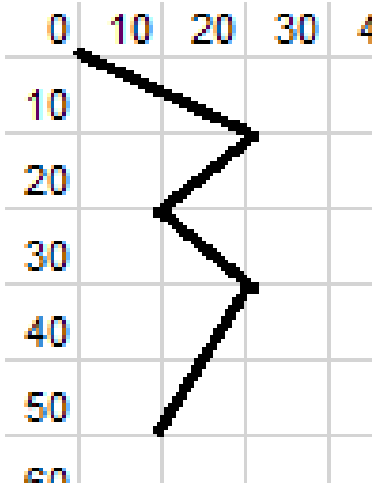
When you find yourself cutting and pasting code, STOP.

You need a method instead.

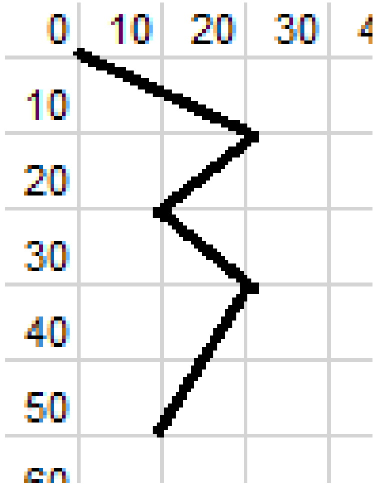
Why are methods useful?

Organization
Reusability
Abstraction
Testing
Extensibility

Organization	Breaks things up into smaller logical units. Think: Tasks in your RPG from last year.
Reusability	Instead of copy/pasting code, you call the method. Think: Less chance of copy/paste errors.
Abstraction	To use someone else’s code, you only need the method signature – then you can call it. You don’t need to understand the details. Think: IO
Testing	When we don’t repeat code (we call a method instead), it is easier to test. There are fewer lines for white box testing. Think: Testing the if 5 times!
Extensibility	By putting repeated code in one place, when we want to change it, we only need to change one place – not all of them.

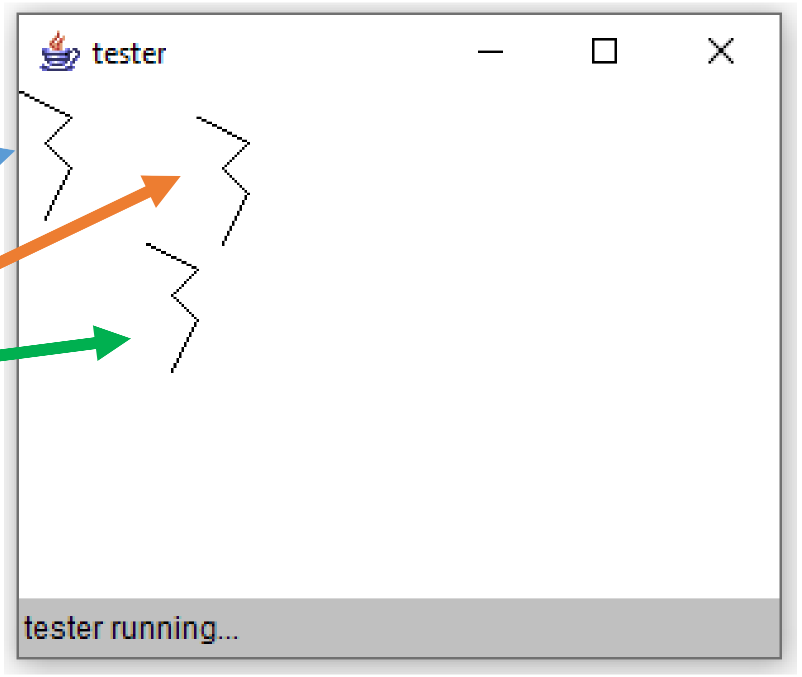


```
public void squiggle (int x, int y)
{
    Graphics g = getGraphics ();
    g.drawLine (  0,      0,   20,   10);
    g.drawLine (  20,   10,   10,   20);
    g.drawLine (  10,   20,   20,   30);
    g.drawLine (  20,   30,   10,   50);
}
```



```
public void squiggle (int x, int y)
{
    Graphics g = getGraphics ();
    g.drawLine (x+0,  y+0,  x+20, y+10);
    g.drawLine (x+20, y+10, x+10, y+20);
    g.drawLine (x+10, y+20, x+20, y+30);
    g.drawLine (x+20, y+30, x+10, y+50);
}
```

```
import java.applet.Applet;
import java.awt.*;
public class tester extends Applet{
    public void paint (Graphics g){
        squiggle (0, 0);
        squiggle (50, 60);
        squiggle (70, 10);
    }

    public void squiggle (int x, int y){
        Graphics g = getGraphics ();
        g.drawLine (x + 0, y + 0, x + 20, y + 10);
        g.drawLine (x + 20, y + 10, x + 10, y + 20);
        g.drawLine (x + 10, y + 20, x + 20, y + 30);
        g.drawLine (x + 20, y + 30, x + 10, y + 50);
    }
}
```



Method

- A subprogram.
- Used to break a larger program into smaller pieces.

```
import java.applet.Applet;
import java.awt.*;
public class tester extends Applet{
    public void paint (Graphics g){
        squiggle (0, 0);
        squiggle (50, 60);
        squiggle (70, 10);
    }

    public void squiggle (int x, int y){
        Graphics g = getGraphics ();
        g.drawLine (x + 0, y + 0, x + 20, y + 10);
        g.drawLine (x + 20, y + 10, x + 10, y + 20);
        g.drawLine (x + 10, y + 20, x + 20, y + 30);
        g.drawLine (x + 20, y + 30, x + 10, y + 50);
    }
}
```

Method Signature

- First line of the method
- Very important because it specifies all of the input and output of the method AND it’s name

```
import java.applet.Applet;
import java.awt.*;
public class tester extends Applet{
    public void paint (Graphics g){
        squiggle (0, 0);
        squiggle (50, 60);
        squiggle (70, 10);
    }

    public void squiggle (int x, int y){
        Graphics g = getGraphics ();
        g.drawLine (x + 0, y + 0, x + 20, y + 10);
        g.drawLine (x + 20, y + 10, x + 10, y + 20);
        g.drawLine (x + 10, y + 20, x + 20, y + 30);
        g.drawLine (x + 20, y + 30, x + 10, y + 50);
    }
}
```

Parameter

- A variable sent into a method
- It has a type(possibly a view type) and a name (follows Id’s naming rules
- INPUT of the method

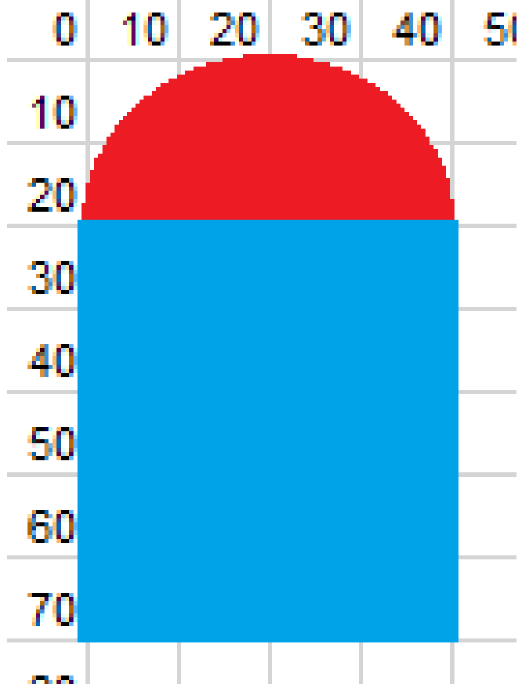
```java
import java.applet.Applet;
import java.awt.*;
public class tester extends Applet{
    public void paint (Graphics g){
        squiggle (0, 0);
        squiggle (50, 60);
        squiggle (70, 10);
    }

    public void squiggle (int x, int y){
        Graphics g = getGraphics ();
        g.drawLine (x + 0, y + 0, x + 20, y + 10);
        g.drawLine (x + 20, y + 10, x + 10, y + 20);
        g.drawLine (x + 10, y + 20, x + 20, y + 30);
        g.drawLine (x + 20, y + 30, x + 10, y + 50);
    }
}
```

Return Type

- The value sent OUT of the method.
- The value is sent back using the “return” line
- It must be the same type specified in the method signature.

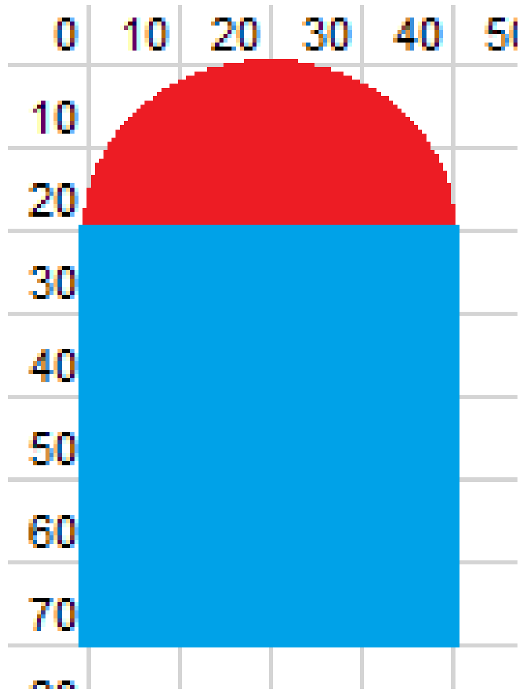
```
import java.applet.Applet;
import java.awt.*;
public class tester extends Applet{
    public void paint (Graphics g){
        squiggle (0, 0);
        squiggle (50, 60);
        squiggle (70, 10);
    }

    public void squiggle (int x, int y){
        Graphics g = getGraphics ();
        g.drawLine (x + 0, y + 0, x + 20, y + 10);
        g.drawLine (x + 20, y + 10, x + 10, y + 20);
        g.drawLine (x + 10, y + 20, x + 20, y + 30);
        g.drawLine (x + 20, y + 30, x + 10, y + 50);
    }
}
```

What's the method name?



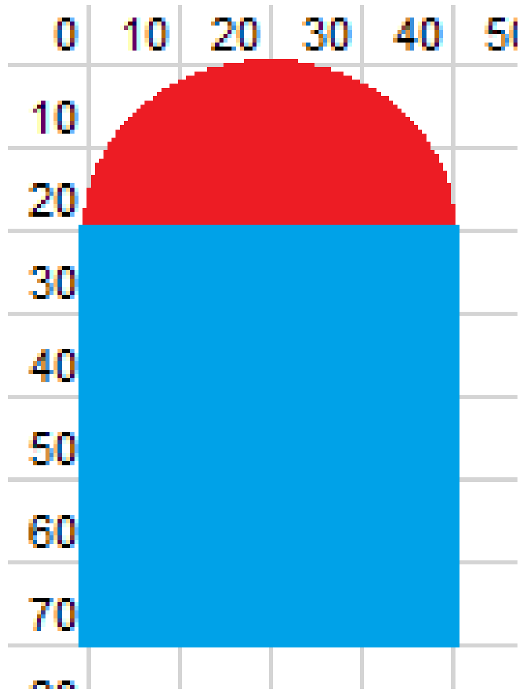
```
public void silo (int x, int y)
{
    Graphics g = getGraphics ();
    g.setColor (Color.red);
    g.fillOval (x + 0, y + 0, 40, 40);
    g.setColor (Color.blue);
    g.fillRect (x + 0, y + 20, 40, 50);
}
```

What is the parameter type?



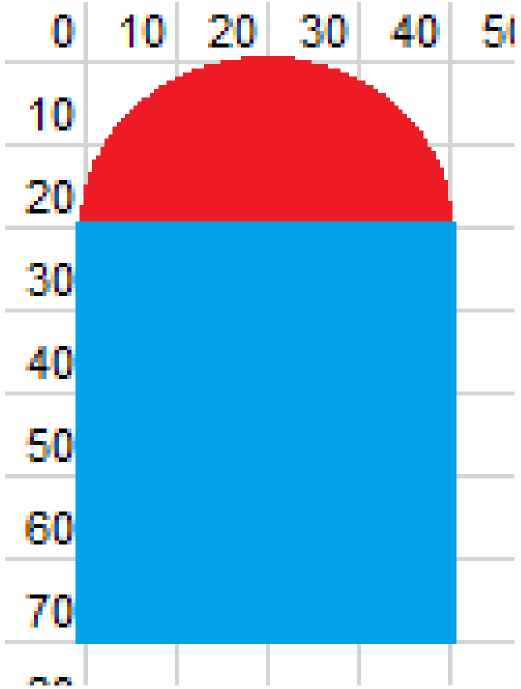
```
public void silo (int x, int y)
{
    Graphics g = getGraphics ();
    g.setColor (Color.red);
    g.fillOval (x + 0, y + 0, 40, 40);
    g.setColor (Color.blue);
    g.fillRect (x + 0, y + 20, 40, 50);
}
```

What are the parameter names?



```
public void silo (int x, int y)
{
    Graphics g = getGraphics ();
    g.setColor (Color.red);
    g.fillOval (x + 0, y + 0, 40, 40);
    g.setColor (Color.blue);
    g.fillRect (x + 0, y + 20, 40, 50);
}
```

What is wrong with the parts that are highlighted in green?



```
public void silo (int x, int y)
{
    Graphics g = getGraphics ();
    g.setColor (Color.red);
    g.fillOval (x + 0, y + 0, 40, 40);
    g.setColor (Color.blue);
    g.fillRect (x + 0, y + 20, 40, 50);
}
```