Prototyping with Methods

Recall that the process of prototyping is one of creating a sequence of prototypes for the same problem. Each prototype is expected to move closer to the final product.

Example:

Think of a name for this image.

```java
import java.awt.Color;
import javax.swing.JFrame;

public class Driver {

    private JFrame win;
    private Oval poleTop, poleBottom;
    private Rectangle pole;

    public Driver() {
        win = new JFrame("Ring Stack");
        win.setBounds(10, 10, 500, 400);
        win.setLayout(null);
        win.setVisible(true);
        win.getContentPane().setBackground(Color.black);
        pole = new Rectangle(240, 80, 20, 200);
        pole.setBackground(Color.yellow);
        win.add(pole, 0);
        poleTop = new Oval(240, 77, 20, 6);
        poleTop.setBackground(Color.orange);
        win.add(poleTop, 0);
        poleBottom = new Oval(240, 277, 20, 6);
        poleBottom.setBackground(Color.yellow);
        win.add(poleBottom, 0);
        /* construct and place three rings */
        win.repaint();
    }
}
```

A First Prototype
Private Methods

When programs get to be too lengthy, it is a good idea to create subprograms.

In Java subprograms take the form of methods. A method has its own code (body) and sometimes has its own (local) variables.

A private method is declared within a class and can be called from any other method within the same class.

```java
import java.awt.Color;
import javax.swing.JFrame;
public class Driver {
    private JFrame win;
    
    public Driver() {
        win = new JFrame( "Ring Stack " );
        
    }
    
    private void someLocalMethod() {
        
    }
}
```

Private Method (parameterless & void)

A private method is __________ by an instruction that consists of the form

```
MethodName()
```

When a method is called, the body of the method (InstructionSequence) is executed, then the method returns to the location immediately after the call instruction.
public class Driver {
    private JFrame win;
    private Oval poleTop, poleBottom, redRing, ringCenter;
    private Rectangle pole;

    public Driver() {
        win = new JFrame("Ring Stack");
        win.setBounds(10, 10, 500, 400);
        win.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        win.getContentPane().setBackground(Color.black);
        pole = new Rectangle(240, 80, 20, 200);
        pole.setBackground(Color.yellow);
        win.add(pole, 0);
        poleTop = new Oval(240, 77, 20, 6);
        poleTop.setBackground(Color.orange);
        win.add(poleTop, 0);
        poleBottom = new Oval(240, 277, 20, 6);
        poleBottom.setBackground(Color.yellow);
        win.add(poleBottom, 0);
        makeRedRing();
        /* construct and place two more rings */
        win.repaint();
    }

    private void makeRedRing() {
        redRing = new Oval(200, 150, 100, 30);
        redRing.setBackground(Color.red);
        win.add(redRing, 0);
        ringCenter = new Oval(10, 10, 80, 10);
        ringCenter.setBackground(Color.black);
        redRing.add(ringCenter, 0);
    }
}

Local Variables

Any method can have its own local variables.

Local variables are declared within the body of the method. (Note: it is best to include local variable declarations before other instructions of the body.)

The syntax of a local variable declaration is the same as for an instance variable excepting that the "private" prefix must be omitted.

The ______(region where they are known) of an instance variable is the class in which it is declared. The scope of a local variable is its method, enclosed by { … }.

Local variables are preferable programming style to instance variables.
private void makeRedRing() {
    Oval redRing, ringCenter;
    Rectangle ringCover;

    redRing = new Oval(200, 150, 100, 30);
    redRing.setBackground(Color.red);
    win.add(redRing, 0);
    ringCenter = new Oval(10, 10, 80, 10);
    ringCenter.setBackground(Color.black);
    redRing.add(ringCenter, 0);
    ringCover = new Rectangle(40, 0, 20, 22);
    ringCover.setBackground(Color.yellow);
    redRing.add(ringCover, 0);
}

Private Method with Parameters

private void MethodName(Parms) { InstructionSequence }

Parms

Type ParameterName

Type denotes any valid data type (such as int or the name of some class).

Parameters (also called __________ parameters) are assigned arguments at the time of call. Otherwise, they behave like local variables.
public Driver() {
    win = new JFrame(" Ring Stack");
    ...
    makeRing(130, Color.green);
    makeRing(150, Color.red);
    makeRing(170, Color.blue);
    win.repaint();
}

private void makeRing(int y, Color c) {
    Oval theRing, ringCenter;
    Rectangle ringCover;
    theRing = new Oval(200, y, 100, 30);
    theRing.setBackground(c);
    win.add(theRing, 0);
    ringCenter = new Oval(10, 10, 80, 10);
    ringCenter.setBackground(Color.black);
    theRing.add(ringCenter, 0);
    ringCover = new Rectangle(40, 0, 20, 22);
    ringCover.setBackground(Color.yellow);
    theRing.add(ringCover, 0);
    ...
}

To fix the code . . .

public Driver() {
    win = new JFrame("Ring Stack");
    win.setBounds(10, 10, 500, 400);
    win.setLayout(null);
    win.setVisible(true);
    win.getContentPane().setBackground(Color.black);
    pole = new Rectangle(240, 80, 20, 200);
    pole.setBackground(Color.yellow);
    win.add(pole, 0);
    poleTop = new Oval(240, 77, 20, 6);
    poleTop.setBackground(Color.orange);
    win.add(poleTop, 0);
    poleBottom = new Oval(240, 277, 20, 6);
    poleBottom.setBackground(Color.yellow);
    win.add(poleBottom, 0);
    makeRing(170, Color.blue);
    makeRing(150, Color.red);
    makeRing(130, Color.green);
    win.repaint();
}