Lecture 4.3

Case Study
Using Non-void Methods

Non-void Methods
Parameters are largely one-way communication.

- calling code
- parameter passage
- method

Sometimes the method needs to transmit information back to the caller.
  - Shared instances variables is one way to accomplish this.
  - Another alternative is a non-void method. These are sometimes called functions.

Characteristics of a Non-void Method
1) A non-void method is declared with a type instead of “void”.

2) A non-void method can be called anywhere an expression of the same type is permitted.

3) A non-void method returns a value. Therefore, the last instruction it executes must be a return instruction.
Writing Your Own Non-void Methods

Syntax
The form of the method is the same as for void methods except replacing “void” with the name of a class (or primitive type).

Each non-void method must include a return instruction with the following form.

\[ \text{return expr;} \]

where expr denotes a valid expression with the same type as the method.

Executing a return

1) The method completes execution and returns to the caller.

2) The value of expr is returned as the value of the method and used in the expression from which the method was called.

Think Halloween!

Program Requirements: Draw three jack-o-lanterns as follows

What patterns do you see repeated?
What does such repetition suggest for the use of methods?

Our first prototype does not include the teeth or stems.
import java.awt.*;
import javax.swing.JFrame;
/** First prototype of jack-o-lantern program
 * Author: David D. Riley
 * Date: Sep, 2015
 */
public class Driver  {
    private JFrame win;
    public Driver() {
        Rectangle leftJack, midJack, rightJack;
        win = new JFrame("the window");
        win.setBounds(10, 10, 380, 150);
        win.setLayout(null);
        win.getContentPane().setBackground(Color.black);
        win.setVisible(true);
        leftJack = jackOLantern(20, 5);
        win.add(leftJack, 0);
        midJack = jackOLantern(140, 5);
        win.add(midJack, 0);
        rightJack = jackOLantern(260, 5);
        win.add(rightJack, 0);
        win.repaint();
    }
    // the jackOLantern on next slide
}

// post: @return == a jack-o-lantern with two eyes and a stem
// at location (x, y).
private Rectangle jackOLantern(int x, int y) {
    Rectangle jackCanvas;
    Oval body, leftEye, rightEye;
    jackCanvas = new Rectangle(x, y, 100, 100);
    body = new Oval(0, 20, 100, 80);
    body.setBackground(Color.orange);
    jackCanvas.add(body, 0);
    leftEye = new Oval(20, 25, 15, 10);
    body.add(leftEye, 0);
    rightEye = new Oval(65, 25, 15, 10);
    body.add(rightEye, 0);
    // need to make and add stem here
    // need to make and add teeth here
    return jackCanvas;
}
...
Add the stems to the jacks.

Make two changes:
1) alter the `jackOLantern` method as shown below.
2) write a `getStem` method (see next slide).

```java
// need to add stem here
is replaced by...

```rectangle```
stem = getStem();
jackCanvas.add(stem, 0);
```
OR
```
jackCanvas.add(getStem(), 0);
```

```java
/** post: @return == a green pumpkin stem image
 *  AND the stem is 20 by 20 pixels
 *  AND the stem is located at (40, 0)
 */
private Rectangle getStem() {
   Oval stem, stemCover;
   Rectangle stemBoundary;
   stemBoundary = new Rectangle(40, 0, 20, 20);
   stem = new Oval(6, 6, 28, 28);
   stem.setBackground(Color.green);
   stemCover = new Oval(7, 7, 14, 14);
   stem.add(stemCover, 0);
   stemBoundary.add(stem, 0);
   return stemBoundary;
}
```
Jack-o-lantern -- Prototype 3

Add the teeth to the jacks.

Let’s use a non-void method called **getTooth**.

Make two changes:
1) alter the `jackOLantern` method as shown below.
2) write a `getTooth` method (see next slide).

```java
// need to make and add teeth here

tooth1 = getTooth(20, 52);
body.add(tooth1, 0);
tooth2 = getTooth(35, 50);
body.add(tooth2, 0);
tooth3 = getTooth(50, 50);
body.add(tooth3, 0);
tooth4 = getTooth(65, 52);
body.add(tooth4, 0);
```

---

Jack-o-lantern -- Prototype 3 (cont’d)

```java
/**
 * post: result == is 15 by 15 pixels
 * AND positioned at (x,y)
 * AND displays a black tooth-like image on orange
 */
private Rectangle getTooth(int x, int y) {
    Rectangle background, toothTop, toothBottom;

    background = new Rectangle(x, y, 15, 15);
    background.setBackground(Color.orange);
    toothTop = new Rectangle(0, 0, 15, 10);
    background.add(toothTop, 0);
    toothBottom = new Rectangle(4, 9, 7, 7);
    background.add(toothBottom, 0);
    return background;
}
```
Using Anonymous Objects

```java
    tooth1 = getTooth(20, 52);
    body.add(tooth1, 0);
    tooth2 = getTooth(35, 50);
    body.add(tooth2, 0);
    tooth3 = getTooth(50, 50);
    body.add(tooth3, 0);
    tooth4 = getTooth(65, 52);
    body.add(tooth4, 0);
```

is replaced by...

```java
    body.add( getTooth(20, 52), 0 );
    body.add( getTooth(35, 50), 0 );
    body.add( getTooth(50, 50), 0 );
    body.add( getTooth(65, 52), 0 );
```

No need to declare tooth1, tooth2, tooth3 and tooth4.

Jack-o-lantern – a final improvement

Suppose you’d like to create overlapping jack-o-lanterns.

Problem: Rectangles are not ideal for backgrounds.

Solution: java.awt.Container
java.awt.Container Class Diagram

java.awt.Container
- int x
- int y
- int width
- int height

«constructor»
+ Container()

«update»
+ void add(java.awt.Component, int)
+ void remove(java.awt.Component)
+ void repaint()
+ void setBounds(int, int, int)
+ void setLocation(int, int)
+ void setSize(int, int)

«query»
+ int getX()
+ int getY()
+ int getWidth()
+ int getHeight()
...

Note the non-void (query) methods.