CSE 142 Summer 2001

Defining Methods

Introduction

- A Quick Review:
  - Objects, expressions, statements, types
  - Creating classes
- New today:
  - Defining methods – operational abstractions that "extend" Java
  - Method parameters

A Simple Program

- Let's look at a simple class that uses methods to draw a simple scene

```
/** create a window and draw a simple scene */
class Scene {
  GWindow theWindow;    // window shared by all methods in this class
  public void drawScene( ) {
    theWindow = new GWindow( );

    drawHouse( );
    drawLeftTree( );
    drawRightTree( );
  }
}
```

- It's really easy to see what this program does.
- It tells us *what* the program is doing (not how).

drawHouse( ) [review]

```
/**  Draw a picture of a house  */
public void drawHouse( ) {
  int frameX = 50;        // lower left x of the walls
  int frameY = 200;       // lower left y of the walls
  int wallWidth = 150;
  int wallHeight = 100;
  int roofWidth = 200;
  int roofHeight = 50;
  int overhang = roofWidth - wallWidth;
  int roofBaseY = frameY - wallHeight;

  // Make the basic frame of the house
  theWindow.add(new Rectangle(frameX, roofBaseY, wallWidth, wallHeight));

  // Now make the roof
  theWindow.add(new Triangle(frameX-overhang, roofBaseY,
                frameX + wallWidth + overhang, roofBaseY,
                frameX + (wallWidth/2), roofBaseY - roofHeight));
}
```


drawLeftTree( )

```
/**  Draw the garden's left tree. */
public void drawLeftTree() {
  int x = 270;
  int y = 150;
  int width = 20;
  int height = 100;
  int circleDiameter = 100;

  theWindow.add(new Rectangle(x, y, width, height, Color.blue, true));
  theWindow.add(new Oval(x - (circleDiameter-width)/2,
                         y - circleDiameter/2,
                         circleDiameter, circleDiameter,
                         Color.green, true));
}
```


drawRightTree( )

```
/**  Draw the garden's right tree. */
public void drawRightTree() {
  int x = 340;
  int y = 250;
  int width = 20;
  int height = 100;
  int circleDiameter = 100;

  theWindow.add(new Rectangle(x, y, width, height, Color.blue, true));
  theWindow.add(new Oval(x - (circleDiameter-width)/2,
                         y - circleDiameter/2,
                         circleDiameter, circleDiameter,
                         Color.green, true));
}
```

Discussion

- What are the similarities and differences between these two methods?
- We'd like to *abstract* out the notion of location.
- We can do that by *parameterizing* the method by the position of the tree.

drawTree(int x, int y)

```
/**
  Draw a tree at the given location.
  @param x the upper left X coordinate of the trunk
  @param y the upper left Y coordinate of the trunk
*/
public void drawTree(int x, int y) {
  int width = 20;
  int height = 100;
  int circleDiameter = 100;

  theWindow.add(new Rectangle(x, y, width, height, Color.blue, true));
  theWindow.add(new Oval(x - (circleDiameter-width)/2,
                         y - circleDiameter/2,
                         circleDiameter, circleDiameter,
                         Color.green, true));
}
```

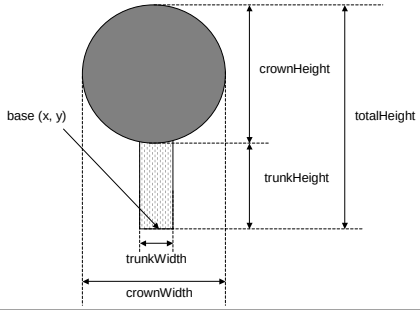

A new drawScene( )

- Now we can use our new method:

```
/**
  Draw the house and the two trees.
*/
public void drawScene( ) {
  drawHouse( );
  drawTree(270, 150);
  drawTree(340, 250);
}
```

- Other issues:
  - What if we want to make trees different sizes?
  - What's a little strange about the interface to drawTree?

How we really think about trees



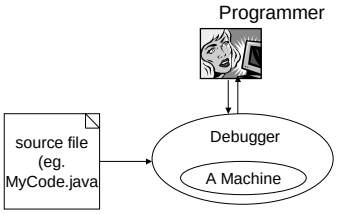
Let's rewrite drawTree

```
/**
  Draw a tree with the given dimensions.
  @param x the x coord of the center of the base of the trunk
  @param y the y coord of the center of the base of the trunk
  @param trunkWidth the width of the trunk   etc etc. . . */
public void drawTree(int x, int y,
                     int trunkWidth, int trunkHeight,
                     int crownWidth, int crownHeight) {

  int rectHeight = trunkHeight + (crownHeight/2);

  theWindow.add(new Rectangle(x - trunkWidth/2,
                              y - rectHeight,
                              trunkWidth, rectHeight, . . .));

  theWindow.add(new Oval(x - crownWidth/2,
                         y - trunkHeight - crownHeight,
                                  crownWidth, crownHeight, . . . ));

}
```


The Debugger

- We can watch this program execute inside of a debugger.
- Every debugger supports the following four fundamental concepts:
  - Setting a breakpoint
  - From a breakpoint, we *continue* execution
  - From a breakpoint, we can *step* our program: *stepping into* vs. *stepping over*.
  - From a breakpoint, we can *inspect* data

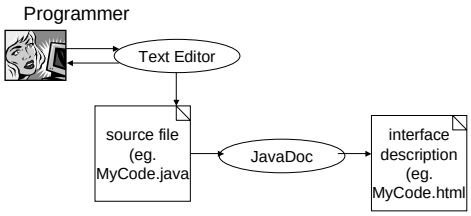
Tools In Pictures: The Debugger



Another Tool: JavaDoc

- Documentation is critical in software projects of any scale.
- Javadoc is a great tool that extracts documentation from Java source files.
- Given a source file, it generates an HTML file(s) containing:
  - Class descriptions
  - Method signatures
  - Method commentary
- To use it: simply write comments following the javadoc convention.

Tools In Pictures: JavaDoc



Example Java File: The Input

```
/**
  This class contains methods that draw a house onto a Gwindow.
  @author Ben Dugan
  @version 0.9
*/
public class Scene {

  /**
    Draw a tree at the given location.
    @param x the upper left X coordinate of the trunk
    @param y the upper left Y coordinate of the trunk
  */
  public void drawTree(int x, int y) { . . . }

  /** Draw the house at the given location.  etc. etc. */
  public void drawHouse(int x, int y) { . . . }

}
```


The Result

- We'd get something that looks like:

class SomeHouse

This class contains methods that draw a house onto a Gwindow.
Author: Ben Dugan
Version: 0.9

drawTree:
public static void drawTree(int x, int y)

Draw a tree at the given location.
parameters:
x the upper left X coordinate of the trunk
y the upper left Y coordinate of the trunk

drawHouse:
public static void drawHouse(int x, int y)

Draws the house at the given location. etc. etc

Javadoc Summary

- Javadoc doesn't eliminate the burden of writing good comments.
- If comments are written in a particular style, javadoc can automate the production good-looking, readable, accessible documentation.
- All you need to know:
  - Doc comments /** … */ as opposed to /* … */ or // …
  - `@author <your name>`
  - `@param <parameter name> <description>`
  - `@return <description>`