Scope

CSE 142, Summer 2003

Computer Programming 1

http://www.cs.washington.edu/education/courses/142/03su/

Readings and References

- Reading
  - » Section 14.7, *Intro to Programming and Object-Oriented Design Using Java*, Niño and Hosch

Declarations

- Everything in a Java program is referenced using an identifier (name)
- Names must be *declared* in the source code
  - » Methods and instance variables in a class
  - » Parameters and local variables in constructors and methods of the class

Variables

- A *variable* is
  - » a portion of memory reserved to hold a single value
- Our program uses little chunks of memory to store the values that it is working with
  - » The program refers to each chunk by name, the name of the variable
  - » When we declare a variable, we give it a name and a type

Variable declarations

```
public class Road implements Prop {
    /** reference to the GWindow object we're displayed on */
    private GWindow gw;
    /** centerline of the road */
    private Shape centerLine;
    […snip…]
    /**
     * Construct the surface and the centerline of the road given the parameters.
     * @param x the x-coordinate of the upper left corner of the road
     * […snip…]
     */
    public Road(int x, int y, int width, int height, boolean east_west) {
        surface = new Rectangle(x, y, width, height, Color.black, true);
        // create the center line
        int centerLineX1;
        centerLineX1 = cornerX;
        […snip…]
    }
    /**
     * Add the elements of this display object to the graphics window.
     * @param g the graphics window to use
     */
    public void addTo(GWindow g) {
        gw = g;
        surface.addTo(gw);
        centerLine.addTo(gw);
    } […snip…]
```

instance variable

constructor parameter

local variable

method parameter

Lifetime

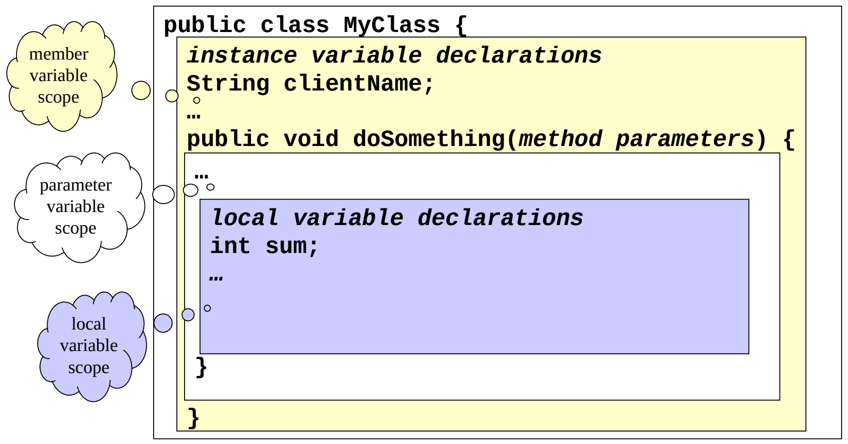
- We've talked about the lifetime of the variables
  - » *Parameter variables* can only be referenced within the body of the constructor or method and the value is lost when the constructor or method returns control to the caller
  - » *Local variables* can only be referenced within the body of the constructor or method and the value is lost when the constructor or method returns control to the caller
  - » *Instance variables* can be referenced using their simple (unqualified) name from within the class and retain their values as long as the object exists

Scope

- A variable's *scope* is the region of a program within which the variable can be referred to by its simple (unqualified) name
  - » Secondarily, scope also determines when the system creates and destroys memory for the variable. If you can't access it, you don't need it.
- Scope limits the range of a declaration
  - » Allows reuse of names (identifiers) in different parts of the code without conflict

What determines scope?

Location of the declaration within your program establishes the scope



```
public class MyClass {
  instance variable declarations
  String clientName;
  …
  public void doSomething(method parameters) {
    …
      local variable declarations
      int sum;
      …
  }
}
```

Members

- The members include fields (instance variables) and methods
- Declared within a class but outside of any method or constructor
- The scope of a class member is the entire declaration of the class.

Parameter variables

- Parameters are formal arguments to methods or constructors and are used to pass values into methods and constructors
- The scope of a parameter is the entire method or constructor for which it is a parameter

Local Variables

- Local variables are declared within a block of code
  - » for example, in the body of a method, in the statement block of a **`for`** loop
- The scope of a local variable extends from its declaration to the end of the code block in which it was declared.

```
/**
 * This class models a Tree using various Shapes.  There is a trunk
 * and a crown of leaves.
 */
public class Tree implements Prop {
  /** The tree trunk */
  private Shape trunk;
  /** The tree leaves */
  private Shape crown;
  /** The GWindow on which the Tree is to be drawn */
  private GWindow gw;

  /**
   * Construct a new Tree, including its component shapes.
   * @param x the x pixel location of the base of the trunk
   * @param y the y pixel location of the base of the trunk
   * @param h the height of the trunk.  Also used to determine
   * the size of the crown.
   */
  public Tree(int x, int y, int h) {
    int width = h/2;
    trunk = new Rectangle(x - width/2, y - h, width, h,
                          Color.orange, true);
    crown = new Oval(x - 3*width/4, y - 3*h/2, 3*width/2, h,
                     Color.blue, true);
  }
...
```

Qualified Names

- Member variables (instance variables, methods) can be referred to with a *qualified name*
  - » assuming that access is allowed (eg public)
- The qualifier is the object that contains the member

```
bob.createProps();
```

refers to the **`createProps()`** method in object **`bob`**, an instance of class **`Director`**

```
public class Producer {
  /**
   * Start the program running.
   * @param arg ignored
   */
  public static void main(String[] arg) {
    Director bob = new Director();
    bob.createProps();
    bob.action();
  }
}
```

```
public class Director {
  /**
   * Create a new Director
   */
  public Director() {
    GWindow frame = new GWindow("Tree Scene");
    frame.setExitOnClose();
    theStage = new Stage(frame);
  }
  /**
   * Add all the props to the stage.
   */
  public void createProps() {
    horizon = new Horizon(0, 200, 500, 200);
    theStage.addProp(horizon);
    sun = new Sun();
    theStage.addProp(sun);
    treeA = new Tree(200,200,30);
    theStage.addProp(treeA);
    treeB = new Tree(250,300,40);
    theStage.addProp(treeB);
  }
```

keyword **this**

- You may want to refer to the current object
  - » from hw4, Director.java

```
public void createProps() {
  Road currentRoad;
  currentRoad = new Road(0, 90, 500, 70, true); // east-west #1
  Car currentCar;
  currentCar = new Car(this,(Road)roadList.get(0),'W',40,30,4,Color.white);
```

- You may want to refer to members of the current object
  - » from hw4, Road.java

```
public Road(int x, int y, int width, int height, boolean east_west) {
  surface = new Rectangle(x, y, width, height, Color.black, true);
  cornerX = x;
  cornerY = y;
  this.width = width;
  this.height = height;
```

Variable Declaration with Initialization

- A variable declaration can specify an initial value

```
public double area(double diameter ) {
        double radius = diameter / 2.0;
        return Math.PI * radius * radius;
}
```

- Common for local variables in methods
  - » use it to create obvious intermediate quantities
- Not common for instance variables
  - » usually put initialization in the constructor instead

Type checking

- Java helps as much as it can to make sure you use variables the way you said you were going to when you declared them
- If you said that **`currentWeight`** is an **`int`**, then Java will make sure you don't unintentionally put a **`double`** value in it and lose the fractional part

```
int currentWeight;
currentWeight = 2;
currentWeight = currentWeight+0.5;
```

- What should the value of currentWeight be at this point?
  - » you said it was an integer, why are you adding 0.5 to it?
  - » the Java compiler decides that this must be a mistake
    - error: "possible loss of precision"

Type casting

- The compiler will tell you if it thinks there's a mistake

```
currentWeight = currentWeight + (currentWeight*rate);
```

  "possible loss of precision. found double, required int"

- If you are really sure that you know it's okay, you can tell the compiler not to worry about it
  - » "I know there's a possible loss of precision, don't fret about it."
- The mechanism for doing this is called casting
- The type you want the value converted to is placed in parentheses in front of the value or expression to convert

```
currentWeight = currentWeight+(int)(currentWeight*rate);
```

- The compiler will convert the value to **`int`** for you
  - » beware: loss of precision may be a real problem!

keyword **`void`**

- Must specify the type of object returned by a method

```
public String getName() {
    return theName;
}
```

- Sometimes we need to specify "nothing is here"

```
public void createProps() {
  horizon = new Horizon(0, 200, 500, 200);
  theStage.addProp(horizon);
  sun = new Sun();
  theStage.addProp(sun);
  treeA = new Tree(200,200,30);
  theStage.addProp(treeA);
  treeB = new Tree(250,300,40);
  theStage.addProp(treeB);
}
```