Scope: Local Variables

• We have discussed the scope of local variables (ones defined in methods) already
  —A variable’s scope extends from the line it is declared until the end of the block it is contained in
  —A formal parameter’s scope is the entire method
  —A variable declared in the for loop control has a scope that extends to the end of the for loop
• This means that variables can’t be used before they are declared, or after the end of their scope

Scope: Instance and Class Variables

• The scope rules for instance and class variables are different
• The scope of instance and class variables is the entire class
  —regardless of where they are declared
• They can be referred to from any method in the class
• This means that you can refer to an instance or class variable before you declare it

Example: Clock class

```java
class Clock {
    private int hour;
    private int minute;
    private int second;
    public Clock( int h, int m, int s ) {
        hour = h % HOURSPERDAY;
        minute = m % MINUTESPERHOUR;
        second = s % SECONDSPERMINUTE;
    }
    public void add( Clock c ) { ... }
    public void tick() { ... }
    public String toString() { ... }
    public static final int HOURSPERDAY = 24;
    public static final int MINUTESPERHOUR = 60;
    public static final int SECONDSPERMINUTE = 60;
}
```

Hidden Variables

• Local variables hide or eclipse instance and class variables with the same name
• References to that name will refer to the local variable
Hidden Class Variables

- Access hidden class variables using the class name

```java
// Scope2.java: class versus local scope
class Scope2 {
    public static void main(String[] args) {
        int x = 2;
        System.out.println("local x = " + x);
        System.out.println("class variable x = " + Scope2.x);
    }
    static int x = 1;
}
```

Hidden Instance Variables

- Last time one of the sample question asked us to write a class Point

```java
class Point {
    private int x;
    private int y;
    public Point(int x1, int y1) {
        x = x1;
        y = y1;
    }
    public String toString() {
        return ("x = ", " + y");
    }
}
```

Hidden Instance Variables

- Notice that we named the Constructor's parameters x1 and y1 instead of the more intuitive x and y

```java
class Point {
    private int x;
    private int y;
    public Point(int x, int y) {
        x = x;
        // which x is which?
        y = y;
    }
    ...
}
```

The Keyword this

- Java provides the keyword this to allow us to refer to an object's hidden instance variables
  - this is a reference to the implicit object

```java
class Point {
    private int x;
    private int y;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
    ...
}
```
Arrays of Objects

• Just as we can have arrays of primitive types, we can also have arrays of objects
• Recall that
  – When we declare an array we have to use new to create the storage for the array
  – When we create an object we have to use new to create the storage for the object
• So, when we create an array of objects we have to use new twice; once for the array and once for the objects.

```java
Point[] pArray = new Point[5];
```

A Deck of Cards

• The book has an example where a deck of cards is represented as a 52 element array.
• Each element is an object of type Card

```
//Deck.java - a deck of playing cards
class Deck {
    Card[] deck;
    Deck() {
        deck = new Card[52];
        for (int i = 0; i < deck.length; i++)
            deck[i] = new Card(new Suit(i / 13 + 1),
                               new Pips(i % 13 + 1));
    }
```
Shuffle the Deck

```java
void shuffle() {
    for (int i = 0; i < deck.length; i++) {
        int k = (int) (Math.random() * 52);
        Card t = deck[i];
        deck[i] = deck[k];
        deck[k] = t;
    }
}
```

• The book has an error
• The highlighted line is a correction

Deck.toString()

```java
public String toString() {
    String t = "";
    for (int i = 0; i < 52; i++) {
        if ((i + 1) % 5 == 0) {
            t = t + deck[i] + "\n";
        } else {
            t = t + deck[i];
        }
    }
    return t;
}
```

• The book has an error
• The highlighted line is a correction

Testing the Deck

```java
//CardTest.java testing the shuffling method.
public class CardTest {
    public static void main(String argv[]) {
        Deck deck = new Deck();
        System.out.println("New Shuffle\n" + deck);
        deck.shuffle();
        System.out.println("New Shuffle\n" + deck);
        deck.shuffle();
        System.out.println("New Shuffle\n" + deck);
    }
}
```

Using classes with HW9

• You might choose to represent the yard as a two dimensional array of Cells
  - Cell contains information about whether that yard location is exposed, contains scat, etc.

```java
Cell[][] yard = new Cell[SIZE][SIZE];
for (int i=0; i < yard.length; i++) {
    for (int j=0; j < yard[i].length; j++) {
        yard[i][j] = new Cell();
    }
}
```
Applets

- A Java applet is executed by a web browser
- There is a special HTML tag that tells the browser to load and run the applet

```html
<applet code="FirstApplet.class" width=500 height=200>
The text here is displayed if the browser doesn't support applets.
</applet>
```

Applet Methods

- Applets have some basic methods
  - `init()` - gets called when applet is loaded
  - `start()` - gets called when applet is started
  - `stop()` - gets called when the reader leaves the page
  - `paint()` - gets called to display the applets UI
- You can define any of these methods
  - If you don't, Java provides default behavior
- There is no `main()` method

A First Applet

```java
import java.awt.*; // required for programs that draw
import javax.swing.*; // required for Swing applets

public class FirstApplet extends JApplet {

public void paint(Graphics g) {
    // draw a line from the upper left corner to
    // 100 pixels below the top center of the Applet
    g.drawLine(0,0,250,100);

    // draw a line from the end of the previous line
    // up to the top center of the Applet
    g.drawLine(250,100,250,0);

    // draw an oval inscribed in an invisible
    // rectangle with its upper left corner at the
    // intersection of the two lines drawn above
    g.drawOval(250,100,200,100);
}
```

Loading the Applet

```html
<html>
<head>
<title>Applet Test</title>
</head>
<body>
<p>Test a simple applet</p>
<applet code="FirstApplet.class" width=500 height=200>
</applet>
</body>
</html>
```
Input and Output

- Java has packages for graphical user interfaces
  - java.awt.*
  - javax.swing.*
- Provide components such as windows, buttons, text input fields, labels, checkboxes
- These packages can be used in Java programs or applets

A GUI Applet

```java
public class AppletSum extends JApplet
    implements ActionListener {

    // instance variables: text boxes for input
    JTextField inputOne = new JTextField(20);
    JTextField inputTwo = new JTextField(20);
    JTextField output = new JTextField(20);

    public void init() {
        Container pane = getContentPane();
        pane.setLayout(new FlowLayout());
        pane.add(new JLabel("Enter one number.");
        pane.add(inputOne);
        pane.add(new JLabel("Enter a number and hit return.");
        pane.add(inputTwo);
        pane.add(new JLabel("Their sum is:");
        pane.add(output);

        // Call actionPerformed when user presses return
        inputTwo.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        double first, second, sum;
        first = Double.parseDouble(inputOne.getText());
        second = Double.parseDouble(inputTwo.getText());
        sum = first + second;
        output.setText(String.valueOf(sum));
    }
}
```