Conditionals, Iterators and Events

Alark Joshi
The if-else Statement

• An else clause can be added to an if statement to make an if-else statement

```java
if ( condition )
    statement1;
else
    statement2;
```

• If the condition is true, statement1 is executed; if the condition is false, statement2 is executed

• One or the other will be executed, but not both
Logic of an if-else statement

- Condition evaluated
  - True: statement1
  - False: statement2
Indentation Revisited

• Remember that indentation is for the human reader, and is ignored by the compiler

```java
if (depth >= UPPER_LIMIT)
    delta = 100;
else
    System.out.println("Resetting Delta");
    delta = 0;
```

• Despite what the indentation implies, `delta` will be set to 0 no matter what
Block Statements

• Several statements can be grouped together into a block statement delimited by braces

• A block statement can be used wherever a statement is called for in the Java syntax rules

```java
if (total > MAX)
{
    System.out.println ("Error!!");
    errorCount++;
}
```
Block Statements

• The if clause, or the else clause, or both, could govern block statements

```java
if (total > MAX)
{
    System.out.println ("Error!!");
    errorCount++;
}
else
{
    System.out.println ("Total: \" + total);
    current = total*2;
}
```

• See Guessing.java
import java.util.*;

public class Guessing
{
    // -------------------------------
    //  Plays a simple guessing game with the user.
    // -------------------------------
    public static void main (String[] args)
    {
        final int MAX = 10;
        int answer, guess;

        Scanner scan = new Scanner (System.in);
        Random generator = new Random();

        answer = generator.nextInt(MAX) + 1;
        continue
    }
}
System.out.print("I'm thinking of a number between 1 and "+MAX+". Guess what it is: ");

guess = scan.nextInt();

if (guess == answer)
    System.out.println("You got it! Good guessing!");
else
    {
        System.out.println("That is not correct, sorry.");
        System.out.println("The number was "+answer);
    }
}
Sample Run

I'm thinking of a number between 1 and 10. Guess what it is: 6
That is not correct, sorry.
The number was 9

```java
if (guess == answer)
    System.out.println ("You got it! Good guessing!");
else
{
    System.out.println ("That is not correct, sorry.");
    System.out.println ("The number was " + answer);
}
```
Nested if Statements

- The statement executed as a result of an if or else clause could be another if statement

- These are called nested if statements

- An else clause is matched to the last unmatched if (no matter what the indentation implies)

- Curly Braces can be used to specify the if statement to which an else clause belongs

- See MinOfThree.java
import java.util.Scanner;

public class MinOfThree
{
    // --------------------------------------------------------------------------------------------------
    //    Reads three integers from the user and determines the smallest
    //    value.
    // --------------------------------------------------------------------------------------------------
    public static void main (String[] args)
    {
        int num1, num2, num3, min = 0;

        Scanner scan = new Scanner (System.in);

        System.out.println ("Enter three integers: ");
        num1 = scan.nextInt();
        num2 = scan.nextInt();
        num3 = scan.nextInt();

        continue
    }
}
continue

    if (num1 < num2)
       if (num1 < num3)
           min = num1;
       else
           min = num3;
    else
       if (num2 < num3)
           min = num2;
       else
           min = num3;

    System.out.println("Minimum value: " + min);

}
```java
continue

if (num1 < num2)
    if (num1 < num3)
        min = num1;
    else
        min = num3;
else
    if (num2 < num3)
        min = num2;
    else
        min = num3;

System.out.println("Minimum value: " + min);
```

**Sample Run**

Enter three integers:
84 69 90
Minimum value: 69
Quick Check

How many times will the string "Here" be printed?

count1 = 1;
while (count1 <= 10)
{
    count2 = 1;
    while (count2 < 20)
    {
        System.out.println ("Here");
        count2++;
    }
    count1++;
}
Quick Check

How many times will the string "Here" be printed?

```java
count1 = 1;
while (count1 <= 10)
{
    count2 = 1;
    while (count2 < 20)
    {
        System.out.println("Here");
        count2++;
    }
    count1++;
}
```

$10 \times 19 = 190$
Comparing Data

• When comparing data using boolean expressions, it's important to understand the nuances of certain data types

• Let's examine some key situations:
  – Comparing floating point values for equality
  – Comparing strings (alphabetical order)
  – Comparing object vs. comparing object references
Comparing Float Values

• You should rarely use the equality operator (==) when comparing two floating point values (float or double)

• Two floating point values are equal only if their underlying binary representations match exactly

• Computations often result in slight differences that may be irrelevant

• In many situations, you might consider two floating point numbers to be "close enough" even if they aren't exactly equal
Comparing Float Values

• To determine the equality of two floats, use the following technique:

```java
if (Math.abs(f1 - f2) < TOLERANCE)
    System.out.println("Essentially equal");
```

• If the difference between the two floating point values is less than the tolerance, they are considered to be equal

• The tolerance could be set to any appropriate level, such as 0.000001
Comparing Strings

- Remember that in Java a character string is an object

- The `equals` method can be called with strings to determine if two strings contain exactly the same characters in the same order

- The `equals` method returns a boolean result

```java
if (name1.equals(name2))
    System.out.println("Same name");
```
Comparing Strings

- Cannot use the relational operators to compare strings
- The `String` class contains the `compareTo` method for determining if one string comes before another
- A call to `name1.compareTo(name2)`
  - returns zero if `name1` and `name2` are equal (contain the same characters)
  - returns a negative value if `name1` is less than `name2`
  - returns a positive value if `name1` is greater than `name2`
Comparing Strings

• Because comparing characters and strings is based on a character set, it is called a lexicographic ordering

```java
int result = name1.compareTo(name2);
if (result < 0)
    System.out.println (name1 + "comes first");
else
    if (result == 0)
        System.out.println ("Same name");
    else
        System.out.println (name2 + "comes first");
```
Lexicographic Ordering

- Lexicographic ordering is not strictly alphabetical when uppercase and lowercase characters are mixed.
- For example, the string "Great" comes before the string "fantastic" because all of the uppercase letters come before all of the lowercase letters in Unicode.
- Also, short strings come before longer strings with the same prefix (lexicographically).
- Therefore "book" comes before "bookcase".
Comparing Objects

• The == operator can be applied to objects – it returns true if the two references are aliases of each other

• The equals method is defined for all objects, but unless we redefine it when we write a class, it has the same semantics as the == operator

• It has been redefined in the String class to compare the characters in the two strings

• When you write a class, you can redefine the equals method to return true under whatever conditions are appropriate
Input Validation

• A loop can also be used for input validation, making a program more robust

• It's generally a good idea to verify that input is valid (in whatever sense) when possible

• See WinPercentage.java
import java.text.NumberFormat;
import java.util.Scanner;

public class WinPercentage
{
    // Computes the percentage of games won by a team.
    public static void main (String[] args)
    {
        final int NUM_GAMES = 12;
        double ratio;
        Scanner scan = new Scanner (System.in);

        System.out.print ("Enter the number of games won (0 to " + NUM_GAMES + "):");
        won = scan.nextInt();
    }
}

continue
continue

```java
while (won < 0 || won > NUM_GAMES)
{
    System.out.print("Invalid input. Please reenter: ");
    won = scan.nextInt();
}

ratio = (double)won / NUM_GAMES;

NumberFormat fmt = NumberFormat.getPercentInstance();

System.out.println();
System.out.println("Winning percentage: "+fmt.format(ratio));
}
```
continue

while (won < 0 || won > NUM_GAMES) {
    System.out.print ("Invalid input. Please reenter: ");
    won = scan.nextInt();
}

ratio = (double)won / NUM_GAMES;

NumberFormat fmt = NumberFormat.getPercentInstance();

System.out.println ("Winning percentage: ");
System.out.println ("" + fmt.format(ratio));
Nested Loops

• Similar to nested if statements, loops can be nested as well

• That is, the body of a loop can contain another loop

• For each iteration of the outer loop, the inner loop iterates completely

• See PalindromeTester.java
import java.util.Scanner;

public class PalindromeTester {
    public static void main (String[] args) {
        String str, another = "y";
        int left, right;

        Scanner scan = new Scanner (System.in);

        while (another.equalsIgnoreCase("y")) // allows y or Y
        {
            System.out.println ("Enter a potential palindrome:");
            str = scan.nextLine();

            left = 0;
            right = str.length() - 1;

            continue
        }
    }
}
continue

while (str.charAt(left) == str.charAt(right) && left < right)
{
    left++;
    right--;
}

System.out.println();

if (left < right)
    System.out.println ("That string is NOT a palindrome.");
else
    System.out.println ("That string IS a palindrome.");

System.out.println();
System.out.print ("Test another palindrome (y/n)? ");
another = scan.nextLine();
}
continue

```java
while (str.charAt(left) == str.charAt(right) && left < right)
{
    left++;
    right--;
}

System.out.println();
if (left < right)
    System.out.println("That string is NOT a palindrome.");
else
    System.out.println("That string IS a palindrome.");

System.out.println();
System.out.print("Test another palindrome (y/n)? ");
another = scan.nextLine();
```
In Class Activity

• From the code repo download `PalindromeTester.java`

• Modify it to test for the following palindromes
  – A man, a plan, a canal, Panama
  – Rise to vote, sir.
  – Dennis and Edna sinned.
Iterators

• An *iterator* is an object that allows you to process a collection of items one at a time

• It lets you step through each item in turn and process it as needed

• An iterator has a `hasNext` method that returns true if there is at least one more item to process

• The `next` method returns the next item
Iterators

• Several classes in the Java standard class library are iterators

• The **Scanner** class is an iterator
  
  – the `hasNext` method returns true if there is more data to be scanned

  – the `next` method returns the next scanned token as a string

• The **Scanner** class also has variations on the `hasNext` method for specific data types (such as `hasNextInt`)
Iterators

• The fact that a Scanner is an iterator is particularly helpful when reading input from a file

• Suppose we wanted to read and process a list of URLs stored in a file

• One scanner can be set up to read each line of the input until the end of the file is encountered

• Another scanner can be set up for each URL to process each part of the path

• See URLDissector.java
import java.util.Scanner;
import java.io.*;

public class URLDissector
{
    public static void main (String[] args) throws IOException
    {
        String url;
        Scanner fileScan, urlScan;

        fileScan = new Scanner (new File("urls.inp"));

        continue
    }
}
// Read and process each line of the file
while (fileScan.hasNext())
{
    url = fileScan.nextLine();
    System.out.println("URL: " + url);

    urlScan = new Scanner(url);
    urlScan.useDelimiter("/");

    // Print each part of the url
    while (urlScan.hasNext())
    {
        System.out.println("   " + urlScan.next());
    }
    System.out.println();
}
}
Sample Run

URL: www.google.com
   www.google.com

URL: www.linux.org/info/gnu.html
   www.linux.org
   info
   gnu.html

URL: thelyric.com/calendar/
   thelyric.com
   calendar

URL: www.cs.vt.edu/undergraduate/about
   www.cs.vt.edu
   undergraduate
   about

URL: youtube.com/watch?v=EHCRImWRGLs
   youtube.com
   watch?v=EHCRImWRGLs
The ArrayList Class

• An ArrayList object stores a list of objects, and is often processed using a loop

• The ArrayList class is part of the java.util package

• You can reference each object in the list using a numeric index

• An ArrayList object grows and shrinks as needed, adjusting its capacity as necessary
The ArrayList Class

• Index values of an ArrayList begin at 0 (not 1):
  
  0  "Bashful"
  1  "Sleepy"
  2  "Happy"
  3  "Dopey"
  4  "Doc"

• Elements can be inserted and removed

• The indexes of the elements adjust accordingly
ArrayList Methods

- Some ArrayList methods:

  boolean add (E obj)
  void add (int index, E obj)
  Object remove (int index)
  Object get (int index)
  boolean isEmpty()
  int size()
The ArrayList Class

• The type of object stored in the list is established when the ArrayList object is created:

```java
ArrayList<String> names = new ArrayList<String>();
ArrayList<Book> list = new ArrayList<Book>();
```

• This makes use of Java generics, which provide additional type checking at compile time

• An ArrayList object cannot store primitive types, but that's what wrapper classes are for

• See Beatles.java
import java.util.ArrayList;

public class Beatles {
    public static void main (String[] args) {
        ArrayList<String> band = new ArrayList<String>();

        band.add("Paul");
        band.add("Pete");
        band.add("John");
        band.add("George");

        continue
    }
}
System.out.println (band);
int location = band.indexOf ("Pete");
band.remove (location);

System.out.println (band);
System.out.println ("At index 1: " + band.get(1));
band.add (2, "Ringo");

System.out.println ("Size of the band: " + band.size());
int index = 0;
while (index < band.size())
{
    System.out.println (band.get(index));
    index++;
}
```java
continue

System.out.println(band);
int location = band.indexOf("Pete");
band.remove(location);
System.out.println(band);
System.out.println("At index 1: "+band.get(1));
band.add(2, "Ringo");
System.out.println("Size of the band: "+band.size());
int index = 0;
while (index < band.size()) {
    System.out.println(band.get(index));
    index++;
}
```

Output

```
[Paul, Pete, John, George]
[Paul, John, George]
At index 1: John
Size of the band: 4
Paul
John
Ringo
George
```
Determining Event Sources

• Recall that interactive GUIs require establishing a relationship between components and the listeners that respond to component events

• One listener object can be used to listen to two different components

• The source of the event can be determined by using the `getSource` method of the event passed to the listener

• See `LeftRight.java`
• See `LeftRightPanel.java`
import javax.swing.JFrame;

public class LeftRight
{
    //----------------------------------------------------------------------------------------
    //  Creates the main program frame.
    //----------------------------------------------------------------------------------------
    public static void main (String[] args)
    {
        JFrame frame = new JFrame ("Left Right");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);

        frame.getContentPane().add(new LeftRightPanel());

        frame.pack();
        frame.setVisible(true);
    }
}
import javax.swing.JFrame;

public class LeftRight {
    public static void main (String[] args) {
        JFrame frame = new JFrame("Left Right");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentTypePane().add(new LeftRightPanel());
        frame.pack();
        frame.setVisible(true);
    }
}
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class LeftRightPanel extends JPanel
{
    private JButton left, right;
    private JLabel label;
    private JPanel buttonPanel;

    continue
// Constructor: Sets up the GUI.

public LeftRightPanel ()
{
    left = new JButton ("Left");
    right = new JButton ("Right");

    ButtonListener listener = new ButtonListener();
    left.addActionListener (listener);
    right.addActionListener (listener);

    label = new JLabel ("Push a button");

    buttonPanel = new JPanel();
    buttonPanel.setPreferredSize (new Dimension(200, 40));
    buttonPanel.setBackground (Color.blue);
    buttonPanel.add (left);
    buttonPanel.add (right);

    setPreferredSize (new Dimension(200, 80));
    setBackground (Color.cyan);
    add (label);
    add (buttonPanel);
}
private class ButtonListener implements ActionListener
{
    public void actionPerformed (ActionEvent event)
    {
        if (event.getSource() == left)
            label.setText("Left");
        else
            label.setText("Right");
    }
}
Check Boxes

• A check box is a button that can be toggled on or off
• It is represented by the JCheckBox class
• Unlike a push button, which generates an action event, a check box generates an *item event* whenever it changes state
• The ItemListener interface is used to define item event listeners
• A check box calls the itemStateChanged method of the listener when it is toggled
Check Boxes

• Let's examine a program that uses check boxes to determine the style of a label's text string

• It uses the Font class, which embodies a character font's:
  – family name (such as Times or Courier)
  – style (bold, italic, or both)
  – font size

• See StyleOptions.java
• See StyleOptionsPanel.java
import javax.swing.JFrame;

public class StyleOptions
{
    public static void main (String[] args)
    {
        JFrame frame = new JFrame("Style Options");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        StyleOptionsPanel panel = new StyleOptionsPanel();
        frame.getContentPane().add(panel);

        frame.pack();
        frame.setVisible(true);
    }
}
import javax.swing.JFrame;

public class StyleOptions {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Style Options");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        StyleOptionsPanel panel = new StyleOptionsPanel();
        frame.getContentPane().add(panel);
        frame.pack();
        frame.setVisible(true);
    }
}
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class StyleOptionsPanel extends JPanel
{
    private JLabel saying;
    private JCheckBox bold, italic;

    continue
// Sets up a panel with a label and some check boxes that control the style of the label's font.

public StyleOptionsPanel()
{
    saying = new JLabel("Say it with style!");
saying.setFont(new Font("Helvetica", Font.PLAIN, 36));

    bold = new JCheckBox("Bold");
bold.setBackground(Color.cyan);
italic = new JCheckBox("Italic");
italic.setBackground(Color.cyan);

    StyleListener listener = new StyleListener();
    bold.addItemListener(listener);
    italic.addItemListener(listener);

    add(saying);
    add(bold);
    add(italic);

    setBackground(Color.cyan);
    setPreferredSize(new Dimension(300, 100));
}
private class StyleListener implements ItemListener {
    public void itemStateChanged (ItemEvent event) {
        int style = Font.PLAIN;

        if (bold.isSelected())
            style = Font.BOLD;

        if (italic.isSelected())
            style += Font.ITALIC;

        saying.setFont (new Font("Helvetica", style, 36));
    }
}
Radio Buttons

• A group of radio buttons represents a set of mutually exclusive options – only one can be selected at any given time

• When a radio button from a group is selected, the button that is currently "on" in the group is automatically toggled off

• To define the group of radio buttons that will work together, each radio button is added to a ButtonGroup object

• A radio button generates an action event
Radio Buttons

• Let's look at a program that uses radio buttons to determine which line of text to display

• See QuoteOptions.java

• See QuoteOptionsPanel.java
import javax.swing.JFrame;

public class QuoteOptions {
    public static void main (String[] args) {
        JFrame frame = new JFrame("Quote Options");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        QuoteOptionsPanel panel = new QuoteOptionsPanel();
        frame.getContentPane().add(panel);

        frame.pack();
        frame.setVisible(true);
    }
}
import javax.swing.JFrame;

public class QuoteOptions
{
    // -------------------------------
    // Creates and presents the program frame.
    // -------------------------------
    public static void main (String[] args)
    {
        JFrame frame = new JFrame ("Quote Options");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        QuoteOptionsPanel panel = new QuoteOptionsPanel();
        frame.getContentPane().add (panel);
        frame.pack();
        frame.setVisible(true);
    }
}
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class QuoteOptionsPanel extends JPanel {
    private JLabel quote;
    private JRadioButton comedy, philosophy, carpentry;
    private String comedyQuote, philosophyQuote, carpentryQuote;

    public QuoteOptionsPanel() {
        comedyQuote = "Take my wife, please.";
        philosophyQuote = "I think, therefore I am.";
        carpentryQuote = "Measure twice. Cut once."

        quote = new JLabel (comedyQuote);
        quote.setFont (new Font ("Helvetica", Font.BOLD, 24));
        continue
comedy = new JRadioButton ("Comedy", true);
comedy.setBackground (Color.green);
philosophy = new JRadioButton ("Philosophy");
philosophy.setBackground (Color.green);
carpentry = new JRadioButton ("Carpentry");
carpentry.setBackground (Color.green);

ButtonGroup group = new ButtonGroup();
group.add (comedy);
group.add (philosophy);
group.add (carpentry);

QuoteListener listener = new QuoteListener();
comedy.addActionListener (listener);
philosophy.addActionListener (listener);
carpentry.addActionListener (listener);

add (quote);
add (comedy);
add (philosophy);
add (carpentry);

setBackground (Color.green);
setPreferredSize (new Dimension(300, 100));
}
private class QuoteListener implements ActionListener {
    
    // Sets the text of the label depending on which radio button was pressed.
    public void actionPerformed (ActionEvent event) {
        Object source = event.getSource();

        if (source == comedy) {
            quote.setText (comedyQuote);
        } else {
            if (source == philosophy) {
                quote.setText (philosophyQuote);
            } else {
                quote.setText (carpentryQuote);
            }
        }
    }
}