Control Statements: Part 1
4.2 Algorithms

• Algorithms
  – The actions to execute
  – The order in which these actions execute

• Program control
  – Specifies the order in which actions execute in a program
4.3 Pseudocode

• Pseudocode
  – An informal language similar to English
  – Helps programmers develop algorithms
  – Does no run on computers
  – Should contain input, output and calculation actions
  – Should not contain variable declarations
4.4 Control Structures

• **Sequential execution**
  
  – Statements are normally executed one after the other in the order in which they are written

• **Transfer of control**
  
  – Specifying the next statement to execute that is not necessarily the next one in order
  
  – Can be performed by the `goto` statement
    
    • Structured programming eliminated `goto` statements
4.4 Control Structures – No GOTOs

• Bohm and Jacopini’s research
  – Demonstrated that goto statements were unnecessary
  – Demonstrated that all programs could be written with three control structures
    • The sequence structure,
    • The selection structure and
    • The repetition structure
4.4 Control Structures (Cont.)

• **UML activity diagram** ([www.uml.org](http://www.uml.org))
  - Models the workflow (or activity) of a part of a software system
  - Action-state symbols (rectangles with their sides replaced with outward-curving arcs)
    • represent action expressions specifying actions to perform
  - Diamonds
    • Decision symbols (explained in section 4.5)
    • Merge symbols (explained in section 4.7)
4.4 Control Structures (Cont.)

– Small circles
  • Solid circle represents the activity’s initial state
  • Solid circle surrounded by a hollow circle represents the activity’s final state

– Transition arrows
  • Indicate the order in which actions are performed

– Notes (rectangles with the upper-right corners folded over)
  • Explain the purposes of symbols (like comments in Java)
  • Are connected to the symbols they describe by dotted lines
Fig. 4.1 | Sequence structure activity diagram.
4.4 Control Structures (Cont.)

• Selection Statements
  – if statement
    • Single-selection statement
  – if...else statement
    • Double-selection statement
  – switch statement
    • Multiple-selection statement
4.4 Control Structures (Cont.)

• Repetition statements
  – Also known as looping statements
  – Repeatedly performs an action while its loop-continuation condition remains true
  – \texttt{while} statement
    • Performs the actions in its body zero or more times
  – \texttt{do}...\texttt{while} statement
    • Performs the actions in its body one or more times
  – \texttt{for} statement
    • Performs the actions in its body zero or more times
4.4 **Control Structures (Cont.)**

• **Java has three kinds of control structures**
  – Sequence statement,
  – Selection statements (three types) and
  – Repetition statements (three types)
  – All programs are composed of these control statements
    • Control-statement stacking
      – All control statements are single-entry/single-exit
    • Control-statement nesting
4.5 if Single-Selection Statement

• **if** statements
  – Execute an action if the specified condition is *true*
  – Can be represented by a decision symbol (diamond) in a UML activity diagram
    • Transition arrows out of a decision symbol have guard conditions
      – Workflow follows the transition arrow whose guard condition is true
Fig. 4.2 | if single-selection statement UML activity diagram.
4.6 *if...else* Double-Selection Statement

- **if...else statement**
  - Executes one action if the specified condition is *true* or a different action if the specified condition is *false*

- **Conditional Operator ( ? : )**
  - Java’s only ternary operator (takes three operands)
  - `? :` and its three operands form a conditional expression
    - Entire conditional expression evaluates to the second operand if the first operand is *true*
    - Entire conditional expression evaluates to the third operand if the first operand is *false*
Fig. 4.3 | if else double-selection statement UML activity diagram.
4.7 while Repetition Statement

• **while statement**
  – Repeats an action while its loop-continuation condition remains true
  – Uses a merge symbol in its UML activity diagram
    • Merges two or more workflows
    • Represented by a diamond (like decision symbols) but has:
      – Multiple incoming transition arrows,
      – Only one outgoing transition arrow and
      – No guard conditions on any transition arrows
Fig. 4.4 | while repetition statement UML activity diagram.
4.8 Formulating Algorithms: Counter-Controlled Repetition

• Counter-controlled repetition
  – Use a counter variable to count the number of times a loop is iterated
4.9 Formulating Algorithms: Sentinel-Controlled Repetition

- **Sentinel-controlled repetition**
  - Also known as indefinite repetition
  - Use a sentinel value (also known as a signal, dummy or flag value)
    - A sentinel value cannot also be a valid input value
• Top-down, stepwise refinement
  – Top step: a single statement that conveys the overall function of the program
  – First refinement: multiple statements using only the sequence structure
  – Second refinement: commit to specific variables, use specific control structures
4.10 Formulating Algorithms: Nested Control Statements

• Control statements can be nested within one another
  – Place one control statement inside the body of the other
1. Initialize passes to zero
2. Initialize failures to zero
3. Initialize student counter to one
4. While student counter is less than or equal to 10
5.   Prompt the user to enter the next exam result
6.   Input the next exam result
7. If the student passed
8.   Add one to passes
9. Else
10.   Add one to failures
11. Add one to student counter
12. Print the number of passes
13. Print the number of failures
14. If more than eight students passed
15.   Print “Raise tuition”
4.11 Compound Assignment Operators

• Compound assignment operators
  – An assignment statement of the form:
    \( \text{variable} = \text{variable operator expression} ; \)
    where \( \text{operator} \) is +, −, *, / or % can be written as:
    \( \text{variable operator=} \text{expression} ; \)
  – Example: \( c = c + 3 ; \) can be written as \( c += 3 ; \)
    • This statement adds 3 to the value in variable \( c \) and stores
      the result in variable \( c \)
4.12 Increment and Decrement Operators

• Unary increment and decrement operators
  – Unary increment operator (++) adds one to its operand
  – Unary decrement operator (--) subtracts one from its operand
  – Prefix increment (and decrement) operator
    • Changes the value of its operand, then uses the new value of the operand in the expression in which the operation appears
  – Postfix increment (and decrement) operator
    • Uses the current value of its operand in the expression in which the operation appears, then changes the value of the operand
### Fig. 4.15 | Increment and decrement operators.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Called</th>
<th>Sample expression</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>prefix increment</td>
<td>++a</td>
<td>Increment $a$ by 1, then use the new value of $a$ in the expression in which $a$ resides.</td>
</tr>
<tr>
<td>++</td>
<td>postfix increment</td>
<td>a++</td>
<td>Use the current value of $a$ in the expression in which $a$ resides, then increment $a$ by 1.</td>
</tr>
<tr>
<td>--</td>
<td>prefix decrement</td>
<td>--b</td>
<td>Decrement $b$ by 1, then use the new value of $b$ in the expression in which $b$ resides.</td>
</tr>
<tr>
<td>--</td>
<td>postfix decrement</td>
<td>b--</td>
<td>Use the current value of $b$ in the expression in which $b$ resides, then decrement $b$ by 1.</td>
</tr>
</tbody>
</table>
4.13 Primitive Types

• Java is a strongly typed language
  – All variables have a type
• Primitive types in Java are portable across all platforms that support Java
4.14 GUI and Graphics Case Study: Creating Simple Drawings

- **Java’s coordinate system**
  - Defined by x-coordinates and y-coordinates
    - Also known as horizontal and vertical coordinates
    - Are measured along the x-axis and y-axis
  - Coordinate units are measured in pixels
- **Graphics class from the java.awt package**
  - Provides methods for drawing text and shapes
- **JPanel class from the javax.swing package**
  - Provides an area on which to draw
Fig. 4.18 | Java coordinate system. Units are measured in pixels.
4.14 GUI and Graphics Case Study: Creating Simple Drawings (Cont.)

• **Inheriting**
  – `extends` keyword
  – The subclass inherits from the superclass
    • The subclass has the data and methods that the superclass has as well as any it defines for itself
// Fig. 4.19: DrawPanel.java
// Draws two crossing lines on a panel.
import java.awt.Graphics;
import javax.swing.JPanel;

public class DrawPanel extends JPanel
{
    // draws an X from the corners of the panel
    public void paintComponent(Graphics g)
    {
        // call paintComponent to ensure the panel displays correctly
        super.paintComponent(g);

        int width = getWidth(); // total width
        int height = getHeight(); // total height

        // draw a line from the upper-left to the lower-right
        g.drawLine(0, 0, width, height);

        // draw a line from the lower-left to the upper-right
        g.drawLine(0, height, width, 0);
    }
}

DrawPanel.java

• DrawPanel.java

Import the java.awt.Graphics and the javax.swing.JPanel classes

The DrawPanel class extends the JPanel class

Declare the paintComponent method

Retrieve the JPanel’s width and height

Draw the two lines
4.14 GUI and Graphics Case Study: Creating Simple Drawings (Cont.)

• The JPanel class
  – Every JPanel has a paintComponent method
    • paintComponent is called whenever the system needs to display the JPanel
  – getWidth and getHeight methods
    • Return the width and height of the JPanel, respectively
  – drawLine method
    • Draws a line from the coordinates defined by its first two arguments to the coordinates defined by its second two arguments
4.14 GUI and Graphics Case Study: Creating Simple Drawings (Cont.)

- **JFrame class from the javax.swing package**
  - Allows the programmer to create a window
  - `setDefaultCloseOperation` method
    - Pass `JFrame.EXIT_ON_CLOSE` as its argument to set the application to terminate when the user closes the window
  - `add` method
    - Attaches a JPanel to the JFrame
  - `setSize` method
    - Sets the width (first argument) and height (second argument) of the JFrame
// Fig. 4.20: DrawPanelTest.java
// Application to display a DrawPanel.
import javax.swing.JFrame;

public class DrawPanelTest
{
    public static void main( String args[] )
    {
        // create a panel that contains our drawing
        DrawPanel panel = new DrawPanel();

        // create a new frame to hold the panel
        JFrame application = new JFrame();

        // set the frame to exit when it is closed
        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

        application.add( panel ); // add the panel to the frame
        application.setSize( 250, 250 ); // set the size of the frame
        application.setVisible( true ); // make the frame visible
    } // end main
} // end class DrawPanelTest