Control Statements: Part 2
5.2 Essentials of Counter-Controlled Repetition

- Counter-controlled repetition requires:
  - Control variable (loop counter)
  - Initial value of the control variable
  - Increment/decrement of control variable through each loop
  - Loop-continuation condition that tests for the final value of the control variable
for ( initialization; loopContinuationCondition; increment )
statement;

can usually be rewritten as:

initialization;
while ( loopContinuationCondition )
{
    statement;
    increment;
}
5.4 Examples Using the \texttt{for} Statement

- Varying control variable in \texttt{for} statement
  - Vary control variable from 1 to 100 in increments of 1
    \begin{verbatim}
    for ( int i = 1; i <= 100; i++ )
    \end{verbatim}
  - Vary control variable from 100 to 1 in increments of −1
    \begin{verbatim}
    for ( int i = 100; i >= 1; i-- )
    \end{verbatim}
  - Vary control variable from 7 to 77 in increments of 7
    \begin{verbatim}
    for ( int i = 7; i <= 77; i += 7 )
    \end{verbatim}
  - Vary control variable from 20 to 2 in decrements of 2
    \begin{verbatim}
    for ( int i = 20; i >= 2; i -= 2 )
    \end{verbatim}
  - Vary control variable over the sequence: 2, 5, 8, 11, 14, 17, 20
    \begin{verbatim}
    for ( int i = 2; i <= 20; i += 3 )
    \end{verbatim}
  - Vary control variable over the sequence: 99, 88, 77, 66, 55, 44, 33, 22, 11, 0
    \begin{verbatim}
    for ( int i = 99; i >= 0; i -= 11 )
    \end{verbatim}
5.5 do...while Repetition Statement

- **do..while** structure
  - Similar to **while** structure
  - Tests loop-continuation after performing body of loop
    - i.e., loop body always executes at least once
Fig. 5.8 | do...while repetition statement UML activity diagram.
5.6 switch Multiple-Selection Statement

- **switch statement**
  - Used for multiple selections
  - I recommend not using it…
5.6 *switch* Multiple-Selection Statement (Cont.)

- **Expression in each case**
  - Constant integral expression
    - Combination of integer constants that evaluates to a constant integer value
  - Character constant
    - E.g., ‘A’, ‘7’ or ‘$’
  - Constant variable
    - Declared with keyword *final*
5.7 break and continue Statements

- **break/continue**
  - Alter flow of control

- **break statement**
  - Causes immediate exit from control structure
    - Used in while, for, do...while or switch statements

- **continue statement**
  - Skips remaining statements in loop body
  - Proceeds to next iteration
    - Used in while, for or do...while statements
Some programmers feel that `break` and `continue` violate structured programming. Since the same effects are achievable with structured programming techniques, these programmers do not use `break` or `continue`.
5.8 Logical Operators

• Logical operators
  – Allows for forming more complex conditions
  – Combines simple conditions

• Java logical operators
  – && (conditional AND)
  – || (conditional OR)
  – & (boolean logical AND)
  – | (boolean logical inclusive OR)
  – ^ (boolean logical exclusive OR)
  – ! (logical NOT)
5.8 Logical Operators (Cont.)

• Boolean Logical AND (\&\&) Operator
  – Works identically to \&\&
  – Except \&\& always evaluate both operands

• Boolean Logical OR (\|\|) Operator
  – Works identically to \|\|
  – Except \| always evaluate both operands
5.9 Structured Programming Summary

• **Sequence structure**
  – “built-in” to Java

• **Selection structure**
  – if, if...else and switch

• **Repetition structure**
  – while, do...while and for
Fig. 5.20 | Java’s single-entry/single-exit sequence, selection and repetition statements.
Rules for Forming Structured Programs

1. Begin with the simplest activity diagram (Fig. 5.22).
2. Any action state can be replaced by two action states in sequence.
3. Any action state can be replaced by any control statement (sequence of action states, if, if else, switch, while, do while or for).
4. Rules 2 and 3 can be applied as often as you like and in any order.

Fig. 5.21 | Rules for forming structured programs.
5.10 (Optional) GUI and Graphics Case Study: Drawing Rectangles and Ovals

• Draw rectangles
  – Method `drawRect` of `Graphics`

• Draw ovals
  – Method `drawOval` of `Graphics`
// Fig. 5.26: Shapes.java
// Demonstrates drawing different shapes.
import java.awt.Graphics;
import javax.swing.JPanel;

public class Shapes extends JPanel
{
    private int choice; // user's choice of which shape to draw

    // constructor sets the user's choice
    public Shapes( int userChoice )
    {
        choice = userChoice;
    } // end Shapes constructor

    // draws a cascade of shapes starting from the top left corner
    public void paintComponent( Graphics g )
    {
        super.paintComponent( g );
    } // end paintComponent
} // end class Shapes

© 2005 Pearson Education, Inc. All rights reserved.
for ( int i = 0; i < 10; i++ )
{
    // pick the shape based on the user's choice
    switch ( choice )
    {
        case 1: // draw rectangles
            g.drawRect( 10 + i * 10, 10 + i * 10,
                        50 + i * 10, 50 + i * 10 );
            break;
        case 2: // draw ovals
            g.drawOval( 10 + i * 10, 10 + i * 10,
                         50 + i * 10, 50 + i * 10 );
            break;
    } // end switch
} // end for
} // end method paintComponent
} // end class Shapes

Outline

Shapes.java
(2 of 2)

Lines 27-28

Lines 31-32

Draw rectangles

Draw ovals
// Fig. 5.27: ShapesTest.java
// Test application that displays class Shapes.
import javax.swing.JFrame;
import javax.swing.JOptionPane;

public class ShapesTest {
    public static void main( String args[] ) {
        // obtain user's choice
        String input = JOptionPane.showInputDialog(
            "Enter 1 to draw rectangles
            "Enter 2 to draw ovals" );

        int choice = Integer.parseInt( input ); // convert input to int

        // create the panel with the user's input
        Shapes panel = new Shapes( choice );

        JFrame application = new JFrame(); // creates a new JFrame

        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        application.add( panel ); // add the panel to the frame
        application.setSize( 300, 300 ); // set the desired size
        application.setVisible( true ); // show the frame
    } // end main
} // end class ShapesTest
ShapesTest.java

Program output
5.11 (Optional) Software Engineering Case Study: Identifying Object’s State and Activities

- **State Machine Diagrams**
  - Commonly called state diagram
  - Model several states of an object
  - Show under what circumstances the object changes state
  - Focus on system behavior
  - UML representation
    - **State**
      - Rounded rectangle
    - **Initial state**
      - Solid circle
    - **Transitions**
      - Arrows with stick arrowheads
**Fig. 5.29** | State diagram for the ATM object.
Software Engineering Observation 5.5

Software designers do not generally create state diagrams showing every possible state and state transition for all attributes—there are simply too many of them. State diagrams typically show only key states and state transitions.
5.11 (Optional) Software Engineering Case Study (Cont.)

• Activity Diagrams
  – Focus on system behavior
  – Model an object’s workflow during program execution
  – Model the actions the object will perform and in what order
  – UML representation
    • Action state (rectangle with its left and right sides replaced by arcs curving outwards)
    • Action order (arrow with a stick arrowhead)
    • Initial state (solid circle)
    • Final state (solid circle enclosed in an open circle)