Control Statements: Selection Control Structure

By Lim Pei Geok

Lecturer of Department of Computer Science,

Southern University College

Objectives

- To understand the flow of control in selection statements
- To use Boolean expressions to control selection statements
- To implement selection control using

1. **if and nested if statements**
2. **Switch** statements

Objectives (cont)

- To write expression using the conditional operator
- To implement program control with

  break.
- To display formatted output using the System.out.printf method, String.format method and DecimalFormat class

Selection Statements

Java has several types of selection statements:

1. simple **if** statements
2. **if…else** statements
3. **Nested if** statements
4. **switch** statements
5. Conditional expressions

Simple if Statements

A simple if statement executes an action only if the condition is true
Syntax:

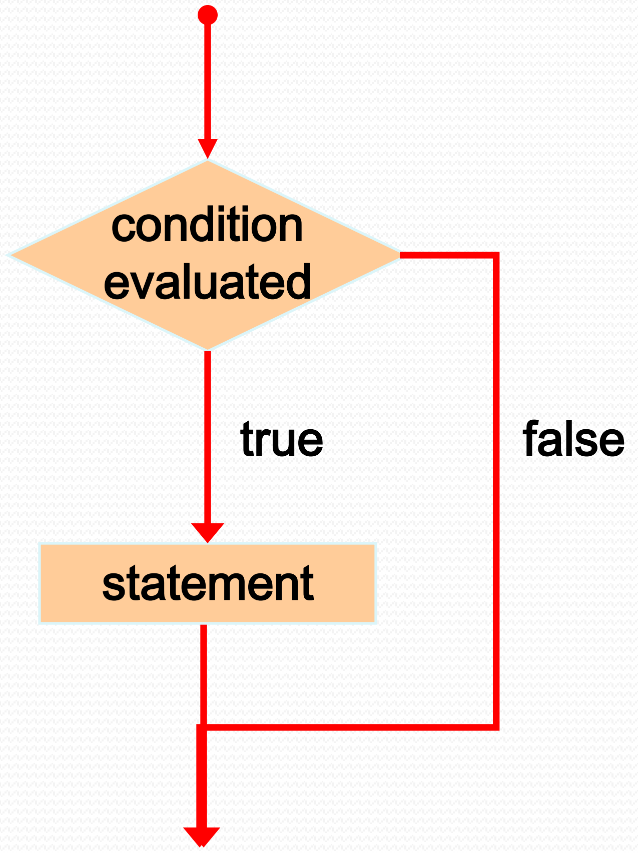
The *condition* must be a boolean expression.
It must evaluate to either true or false.

if is a Java **reserved word**

```
if ( condition )
   statement;
```

If the *condition* is true, the *statement* is executed.
If it is false, the *statement* is skipped.

Logic of if statements



An if statement executes statements if the condition evaluated as true

Example: Simple if Statements

```
public class simpleIf{
   public static void main(String[] args){
          int i=3;      Outer parentheses required

          if ((i>0) && (i<10)) {
             System.out.println("i is an integer between
             0 and 10");
          }
   }

}
```

Braces can be omitted if the block contains a single statement

Caution

Adding a semicolon at the end of an if clause is a common **mistake**.

```
if (radius >= 0);        Logic error
{
  area = radius*radius*PI;
  System.out.println( "The area for the circle of radius " +
  radius + " is " + area);
}
```

This mistake is hard to find, because it is not a compilation error or a runtime error, it is a logic error.

This error often occurs when you use the next-line block style.

Question (if statement)

Write a java program to prompt the user for the student’s mark. If the mark is greater than or equal to 50, then display “Passed”.

The `if...else` Statement

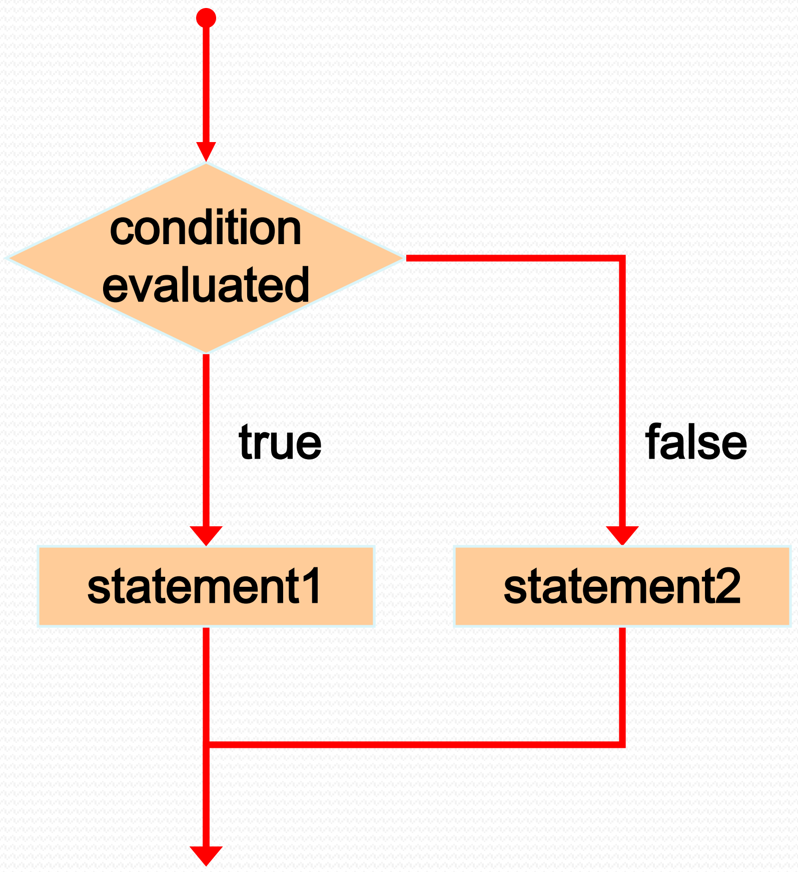
An *else clause* can be added to an if statement to make an ***if-else statement***

```
if (condition) {
   statement1;
}
else {
   statement2;
}
```

If the *condition* is true, *statement1* is executed;

If the condition is false, *statement2* is executed

Logic of an if…else statements



An if…else statement executes statements for the true case if the condition evaluated as true; otherwise statements for the false case are executed

Example: `if...else` statement

```
public class ComputeArea{
    public static void main(String[] args){
    final double PI = 3.142;
    double radius = -3;
    double area;

    if (radius >= 0) {
            area = radius*radius*PI;
            System.out.println("The area for the circle of radius " +
            radius + " is " + area);
    }
    else {
            System.out.println("Negative input");
    }

    }
}
```

Question (if..else statement)

Write a java program to prompt the user for the student’s mark. If the mark is greater than or equal to 50, then display “Passed”, otherwise display “Failed”.

Question

Compute a salesperson’s commission as follows:

- 10% if sales are greater than or equal to RM5000
- 20% if sales are greater than or equal to RM10000

Nested if statements

- The statement executed as a result of an **if** statement 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)
- Braces can be used to specify the **if** statement to which an **else** clause belongs

Example: nested if

```
if ( the time is after 7 pm){
        if ( you have a book)
                read the book;
        else
                watch TV;
}else{
        go for a walk;
}
```

Example: Nested if statement

```
int i = 1; int j = 2; int k = 3;
if(i>k){
   if(j>k)
          System.out.println("i and j are greater than k");
}
else{
   System.out.println("i is less than or equal to k");
}
```

The if (j>k) statement is nested inside the if (i>k) statement

Nested if and if statement uses logical operator

```
int i =1;
if( i > 0){
    if( i < 3){
        System.out.println("i  > 0 and < 3");
    }
}
```

Nested if

```
int i = 1;
 if( i > 0 && i < 3 ){
    System.out.println("i  > 0 and < 3");
 }
```

If statement uses logical AND operator

Question

Write a java program to prompt the user for the student’s mark. The mark entered must in the range of 0-100. If the mark is greater than or equal to 50, then display “Passed”, otherwise display “Failed”. If the mark entered is not in the range of 0-100, the program will display “Invalid mark”.

Multiple Alternative if Statements

```
if (score >= 90)
  grade = 'A';
else
  if (score >= 80)
    grade = 'B';
  else
    if (score >= 70)
      grade = 'C';
    else
      if (score >= 60)
        grade = 'D';
      else
        grade = 'F';
```

```
if (score >= 90)
  grade = 'A';
else if (score >= 80)
  grade = 'B';
else if (score >= 70)
  grade = 'C';
else if (score >= 60)
  grade = 'D';
else
  grade = 'F';
```

This is better

Question

Write a java program to ask user to enter an exam score and display the grade of the entered exam score as follows:

Score	Grade
80-100	A
70-79	B
60-69	C
50-59	D
0-49	F

Note

The <u>else</u> clause matches the most recent <u>if</u> clause in the same block. For example, the following statement

```
int i = 1; int j = 2; int k = 3;
if (i > j)
  if (i > k)
    System.out.println("A");
else
  System.out.println("B");
```

is equivalent to

```
int i = 1; int j = 2; int k = 3;
if (i > j)
  if (i > k)
    System.out.println("A");
  else
    System.out.println("B");
```

This is better with correct indentation

Note

Nothing is printed from the preceding statement. To force the else clause to match the first if clause, you must add a pair of braces:

```
int i = 1;
int j = 2;
int k = 3;
  if (i > j) {
    if (i > k)
      System.out.println("A");
  }
  else
    System.out.println("B");
```

This statement prints B.

`switch` Statements

- The ***switch statement*** provides another means to decide which statement to execute next
- The *switch* statement evaluates an expression, then attempts to match the result to one of several possible ***cases***
- Each case contains a value and a list of statements
- The flow of control transfers to statement associated with the first value that matches

`switch` Statements

The general syntax of a `switch` statement is:

switch and **case** are reserved words

```
switch ( expression )
{
   case value1 :
      statement-list1
   case value2 :
      statement-list2
   case value3 :
      statement-list3
   case ...

}
```

If *expression* matches *value2*, control jumps to here

`switch` Statements

- Often a *break statement* is used as the last statement in each case's statement list
- A break statement causes control to transfer to the end of the switch statement
- **If a break statement is not used, the flow of control will continue into the next case**
- Sometimes this can be appropriate, but usually we want to execute only the statements associated with one case

`switch` Statements

- A `switch` statement can have an optional ***default case***
- The default case has no associated value and simply uses the reserved word `default`
- If the default case is present, control will transfer to it if no other case value matches
- Though the default case can be positioned anywhere in the switch, usually it is placed at the end
- **If there is no default case, and no other value matches, control falls through to the statement after the switch**

Example: switch statements

```
public class SwitchDemo {
    public static void main(String[] args) {
        int month = 8;
        switch (month) {
            case 1:  System.out.println("January"); break;
            case 2:  System.out.println("February"); break;
            case 3:  System.out.println("March"); break;
            case 4:  System.out.println("April"); break;
            case 5:  System.out.println("May"); break;
            case 6:  System.out.println("June"); break;
            case 7:  System.out.println("July"); break;
            case 8:  System.out.println("August"); break;
            case 9:  System.out.println("September"); break;
            case 10: System.out.println("October"); break;
            case 11: System.out.println("November"); break;
            case 12: System.out.println("December"); break;
        }
    }
}
```

Question(Switch)

Use Switch statement to write a program to prompt user for the grade and display the message as follows:

A print “First Class”

B print “Second Upper Class”

C print “Second Lower Class”

D print “Pass”

E print “Fail”

Others print “Error”

switch Statement Rules

The switch-expression must yield a value of char, byte, short, or int type and must always be enclosed in parentheses.

The value1, ..., and valueN must have the same data type as the value of the switch-expression. The resulting statements in the case statement are executed when the value in the case statement matches the value of the switch-expression. (The case statements are executed in sequential order.)

switch Statement Rules

1. The keyword break is optional, but it should be used at the end of each case in order to terminate the remainder of the switch statement. ***If the break statement is not present, the next case statement will be executed.***

2. The default case, which is optional, can be used to perform actions when none of the specified cases is true.

3. The order of the cases (including the default case) does not matter. However, it is a good programming style to follow the logical sequence of the cases and place the default case at the end.

Caution

Do not forget to use a break statement when one is needed. For example, the following code always displays "Wrong number of years" regardless of what numOfYears is. Suppose the numOfYears is 15. The statement annualInterestRate = 8.50 is executed, then the statement annualInterestRate = 9.0, and finally the statement System.out.println("Wrong number of years").

```
switch (numOfYears) {
   case 7:  annualInterestRate = 7.25;
   case 15: annualInterestRate = 8.50;
   case 30: annualInterestRate = 9.0;
   default: System.out.println("Wrong number of years");
}
```

Conditional Operator

- Java has a *conditional operator* that evaluates a boolean condition that determines which of two other expressions is evaluated
- The result of the chosen expression is the result of the entire conditional operator
- Its syntax is:

    condition ? expression1 : expression2

- If the *condition* is true, *expression1* is evaluated; if it is false, *expression2* is evaluated

Conditional Operator (cont)

- The conditional operator is similar to an if-else statement, except that it forms an expression that returns a value
- For example:

```
        larger = ((num1 > num2) ? num1 : num2);
```

- If num1 is greater that num2, then num1 is assigned to larger; otherwise, num2 is assigned to larger
- The conditional operator is *ternary* because it requires three operands

Conditional Operator

```
if (x > 0)
  y = 1;
else
  y = -1;

is equivalent to

y = (x > 0) ? 1 : -1;
```

Example: Conditional Operator

```
public class TestConditional{
        public static void main(String[] args){
                int num=20;
                if (num % 2 == 0)
                        System.out.println(num + " is even");
                else
                        System.out.println(num + " is odd");

                System.out.println((num % 2 == 0)? num + " is
                even": num + "is odd");


        }
}
```

Question (conditional operator)

Write a java program to prompt the user for the student's mark. If the mark is greater than or equal to 50, then display "Passed", otherwise display "Failed".

Use conditional operator to solve the problem

Formatting Console Output

Format the output using ***printf*** method.

Syntax:

System.out.printf(format, item1, item2,...,itemk);

String that consist of substrings and format specifiers

Can be a numeric value, a character, a Boolean value or a string

Format specifier

Specific how an item should be displayed

Specifier	Output	Example
%b	A boolean value	True or false
%c	A character	'a'
%d	A decimal integer	200
%f	A floating-point number	45.460000
%e	A number in standard scientific notation	4.556000e+01
%s	A String	"Java is cool"

Table: Frequently used specifiers

Example

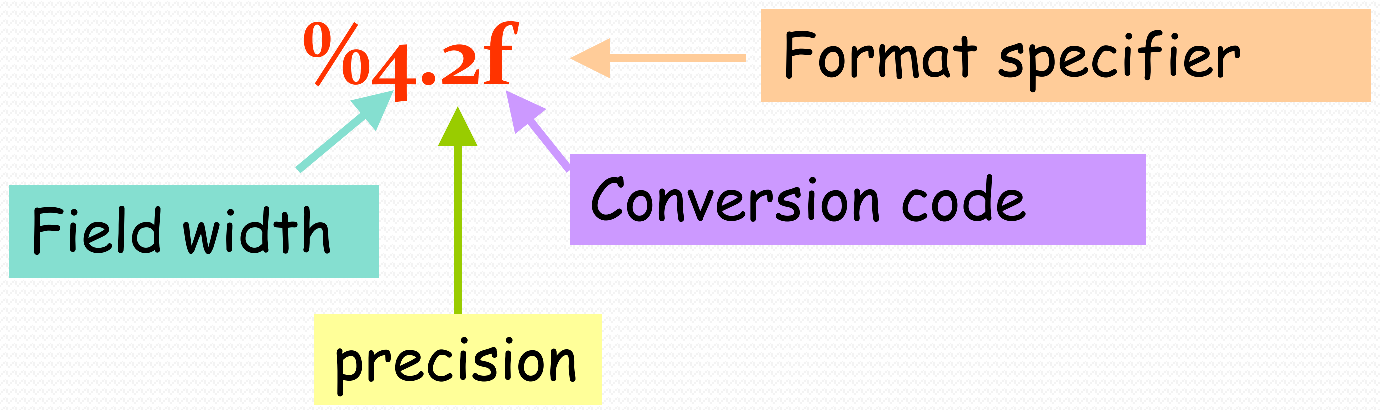
```
public class FormatOutput{
   public static void main(String[] args){
         int count = 5;
         double amount = 45.56;
                                                                  items
         System.out.printf(“count is %d and amount is %f”, count, amount);
   }
}
```

Output in console:

```
Count is 5 and amount is 45.560000
```

Format Specifier

Specify the width and precision in a specified



Example:

double x = 2.0/3;

System.out.printf(“x is %4.2f”, x);

It will display x is 0.67

Examples of specifying width and precision

Example	Output
%5c	Output the character and add four spaces before the character item
%6b	Output the Boolean value and add one space before false value and two spaces before a true value
%5d	Output the integer item with width at least five. If the number of digits in the item is <5, add space before the number. If the number of digits in the item is >5, the width is automatically increase
%10.2f	Output the floating-point item with width at least 10 including a decimal point and two digits after the point. Thus there are 7 digits allocated before the decimal point. If the number of digits before the decimal point is <7, add space before the number. If the number of digits before the decimal point is >7, the width is automatically increase
%12s	Output the string with width at least 12 characters. If the string item has <12, add space before the string. If the string item has >12, the width is automatically increase

Example

```
public class FormatOutput{
  public static void main(String[] args){
      System.out.printf(“%8d%8s%8.1f\n”,1234, “Java”, 5.6);
      System.out.printf(“%-8d%-8s%-8.1f\n”,1234, “Java”, 5.6);
  }
}
```

Output in console:

```
    1234    Java     5.6
1234    Java    5.6
```

By default, the output is right justified. You can put minus sign (-) in the specifier to specified the item is left justified in the output.

Using String.format method

- JDK1.5 simplifies the operation of formatting a String based on parameters.
- The String class now provides a new method called format().
- Use the format specifier to format the output

Example

```
import javax.swing.*;
public class StringFormat {
    public static void main(String[] agr) {

            double value = 1234.56789;

            String format = "%10.2f"; //width = 10 and 2 digits after the decimal point
            JOptionPane.showMessageDialog(null, String.format(format, value),
            "String format", JOptionPane.PLAIN_MESSAGE);

            String formattedVal = String.format("The value is %10.2f", value);
            System.out.println(formattedVal);

            System.out.format(String.format(format, value));
    }
}
```