Lecture 3
Chapter 3 Selections

COMP217
Java Programming
Spring 2021

Objectives

- In this chapter you will:
  - Use Java to compare numbers using relational operators
  - Use `'compareTo'` and `'equals'` methods to compare strings
  - 'if – else' statements
  - 'switch – case' statements

Comparing Numbers in Java

- For primitive data types
  - **`byte`, `short`, `int`, `long`, `float`, `double`, `char`, `boolean`**
- Comparing numbers in Java is to compare numeric values
  - Less than, greater than, equal to
  - Result: **"`true`" or "`false`"**
- **Relational operator**
  - is a symbol in a logical expression to compare two values
  - <u>is used compare two numeric primitives</u>

The boolean Type and Operators

- Often in a program you need to compare two values, such as whether a variable `i` is greater than the other variable `j`.
- Java provides six comparison operators (also known as relational operators) that can be used to compare two values. The result of the comparison is a Boolean value : `true` or `false.`

```
boolean b = (1 > 2);
```

Relational Operators

Relational Operator	Example Let x=5, y=20	Value of Example
`==`	`x == y`	`false`
`!=`	`x != y`	`true`
`<`	`x < y`	`true`
`>`	`x > y`	`false`
`<=`	`x <= y`	`true`
`>=`	`x >= y`	`false`

Comparing Different Data Types

- Can compare literals and variables
  - `"a == 7", "b >= 6", "a == b"`
- Cannot compare a `String` with a number
  - `"7" == 7` does not compile
- Can compare any number with any other number, regardless of numeric data type
  - `"7 >= 6.9"`
- Can compare a number and a `char`
  - Recall `char` evaluates to Unicode integer
  - `'B' > 'A'` is `true`

Apply the Concept

- Declare and initialize numeric variables
- Compare numeric variables, and print the results
- A common mistake is to use = instead of ==

```
c = (a = b);    // assign the value of b
                // to a, and to c
c = (a == b);   // assign true to c
                // if a and b are the same
                // false otherwise
```

ComparingNumbers.java

```java
public class ComparingNumbers {
  public static void main ( String[] args ) {
    // declare and initialize variables
    byte   aByte  = 5;
    short  aShort = -9025;
    int    anInt  = 50000;
    //long   aLong  = 80923097239874992342L;
    long   aLong  = 809230972398749L;
    float  aFloat = 5.0F;
    double aDouble = 3.1415926535897;
    char   char1 = 'A', char2 = 'B', char3 = 'a';

    // form logical expressions
    boolean longFloatComparison = (aLong == aFloat),
            byteIntComparison   = (aByte <= anInt),
            doubleShortComparison = (aDouble != aShort),
            charComparison1 = (char1 == char3),
            charComparison2 = (char3 < char2);

    boolean expr = 15 % 4 * 7 + 15 >= 1
      || 7 < 12 || !(-8 != 7 && 7 <= 10 && 5 > 7);

    // print results
    System.out.println("Compare long   & float: " + longFloatComparison);
    System.out.println("Compare byte   & int  : " + byteIntComparison);
    System.out.println("Compare double & short: " + doubleShortComparison);
    System.out.println("Compare char1  & char3: " + charComparison1);
    System.out.println("Compare char3  & char2: " + charComparison2);
    System.out.println("Value of long expression: " + expr);
  }     // end of main
}       // end of class definition
```

28,37 All

```
$ java ComparingNumbers
Compare long   & float: false
Compare byte   & int  : true
Compare double & short: true
Compare char1  & char3: false
Compare char3  & char2: false
Value of long expression: true
```

Useful Function: SquareRoot.java

The square root of a real number can be computed by Math.sqrt()

```
import java.util.Scanner;
public class SqaureRoot {
  public static void main ( String[] args ) {
    double value;
    Scanner input = new Scanner(System.in);
    System.out.print("Enter a real number: ");
    value = input.nextDouble();
    System.out.println("Square root of "
      + value + " = " + Math.sqrt(value));
  }
}
```

Comparing Strings in Java

- Relational operators compare literals and variables of primitive numeric types
- In Java, strings are not primitives
  - Cannot be compared using relational operators

```
String str1 = "Hello";
String str2 = "Hi";
boolean result;

result = (str1==str2);  // ok on some machines
                        // but not always
result = (str1 > str2); // compilation error
```

The `String` Class

- In Java a string is an object
- In a `String` object:
  - The data are the individual characters
  - The operations are methods for comparing and manipulating the strings
- The `String` class is used to define `String` objects
- The `String` class contains the methods `compareTo` and `equals` which compare strings

The Method `compareTo`

- Used to determine if one string is less than, equal to, or greater than another string
  - Each character in the string has a Unicode (numeric) value
  - Strings are compared by comparing the Unicode values of each individual character
  - Strings are compared left to right
- At the first different character, the (numerical) difference between the Unicode of the second string and the first string is returned

CompareTo Example

```
public class Ex32_compareTo {
  public static void main ( String[] args ) {
    String string1 = "aardvarks";
    int comp1, comp2, comp3, comp4;

    comp1 = string1.compareTo( "boa constrictors" );
    // -1: Unicode for 'a' and 'b' are 97 and 98, string1 is smaller
    // 'a' - 'b' = 97 - 98 = -1
    comp2 = string1.compareTo( "aardvarks" );
    // 0: exactly the same
    comp3 = string1.compareTo( "Aardvarks" );
    // 32: Unicode for 'a' and 'A' are 97 and 65, string1 is larger
    // 'a' - 'A' = 97 - 65 = 32
    comp4 = string1.compareTo( "aardvarks are cooler" );
    // -11: the first 9 characters are the same
    // but the second has 11 more characters

    System.out.println(comp1 + " " + comp2 + " " + comp3 + " " + comp4);
    // -1 0 32 11
  }
}
                                                              21,2
```

The Method `equals`

- `compareTo` returns the difference in Unicode values of the first different character
- `equals` returns `true` if the strings are identical and `false` otherwise
- `equals` is case sensitive
  - Uppercase and lowercase letters have different Unicode values
- `equalsIgnoreCase` is case insensitive

equals Example

```
public class Ex32_equals {
  public static void main ( String[] args ) {
    String string1 = "aardvarks";
    boolean comp1, comp2, comp3, comp3b, comp4; // equals result is a boolean

    comp1 = string1.equals( "boa constrictors" );
    // false: Unicode for 'a' and 'b' are 97 and 98
    comp2 = string1.equals( "aardvarks" );
    // true: exactly the same
    comp3 = string1.equals( "Aardvarks" );
    // false: case sensitive
    comp3b = string1.equalsIgnoreCase( "Aardvarks" );
    // true: ignore case differences
    comp4 = string1.equals( "aardvarks are cooler" );
    // false: numbers of characters mismatch

    System.out.println(
        comp1 + " " + comp2 + " " + comp3 + " " + comp3b + " " + comp4);
    // false true false true false
  }
}
"Ex32_equals.java" 21L, 756C written                          21,2          All
```

Making Decisions

All selections have a comparison between two numbers or strings

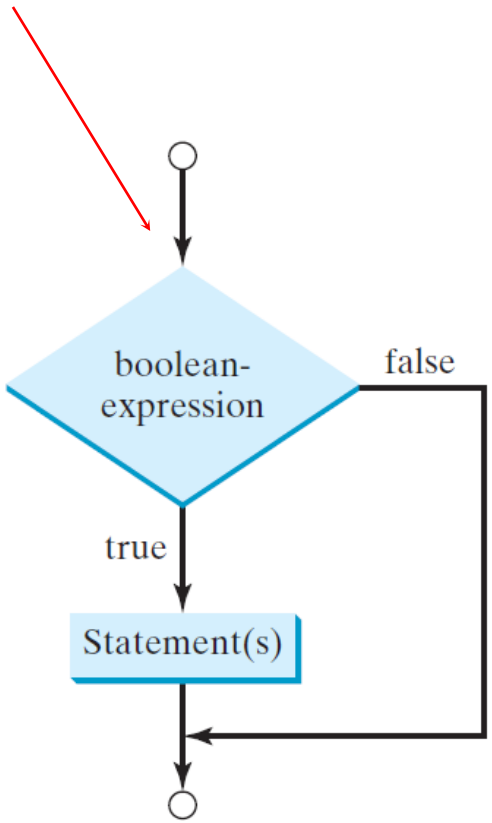
‘if’ / ‘if, else’ statements

Nested if

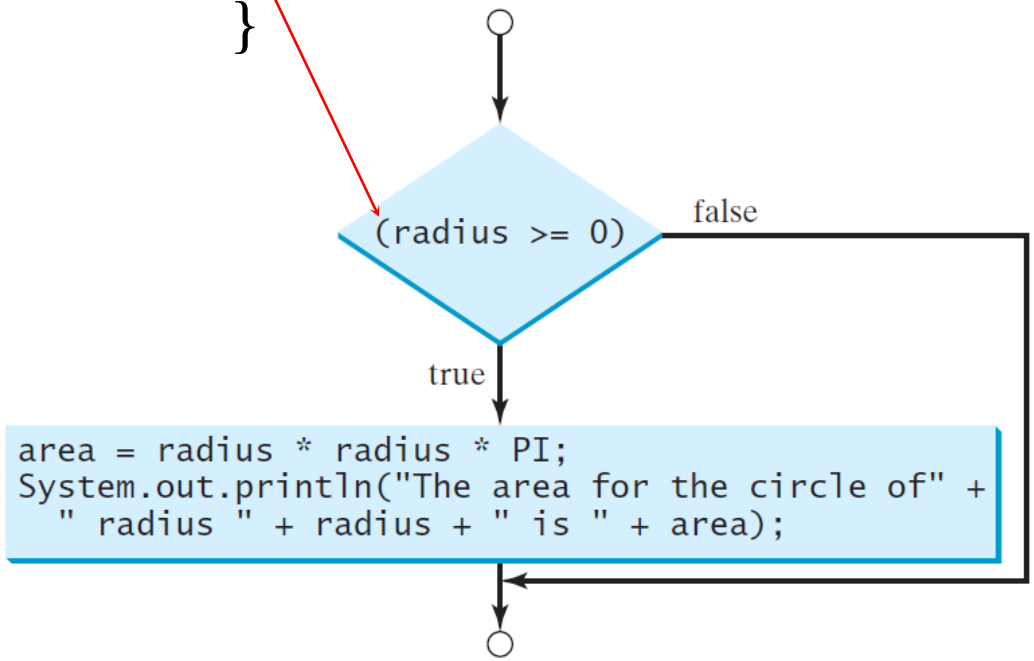
‘switch – case’ / ‘break’

One-way `if` Statements

```
if (boolean-expression) {
  statement(s);
}
```




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



Note

```
if i > 0 {
  System.out.println("i is positive");
}
```

(a) Wrong

```
if (i > 0) {
  System.out.println("i is positive");
}
```

(b) Correct

```
if (i > 0) {
  System.out.println("i is positive");
}
```

(a)

Equivalent

```
if (i > 0)
  System.out.println("i is positive");
```

(b)

Two-way: if-else statements

```
if ( condition )
   action1;   // true
else
   action2;   // false
```

```
if ( grade >= 60 )
  System.out.println("Passed");
else
  System.out.println("Failed");
```

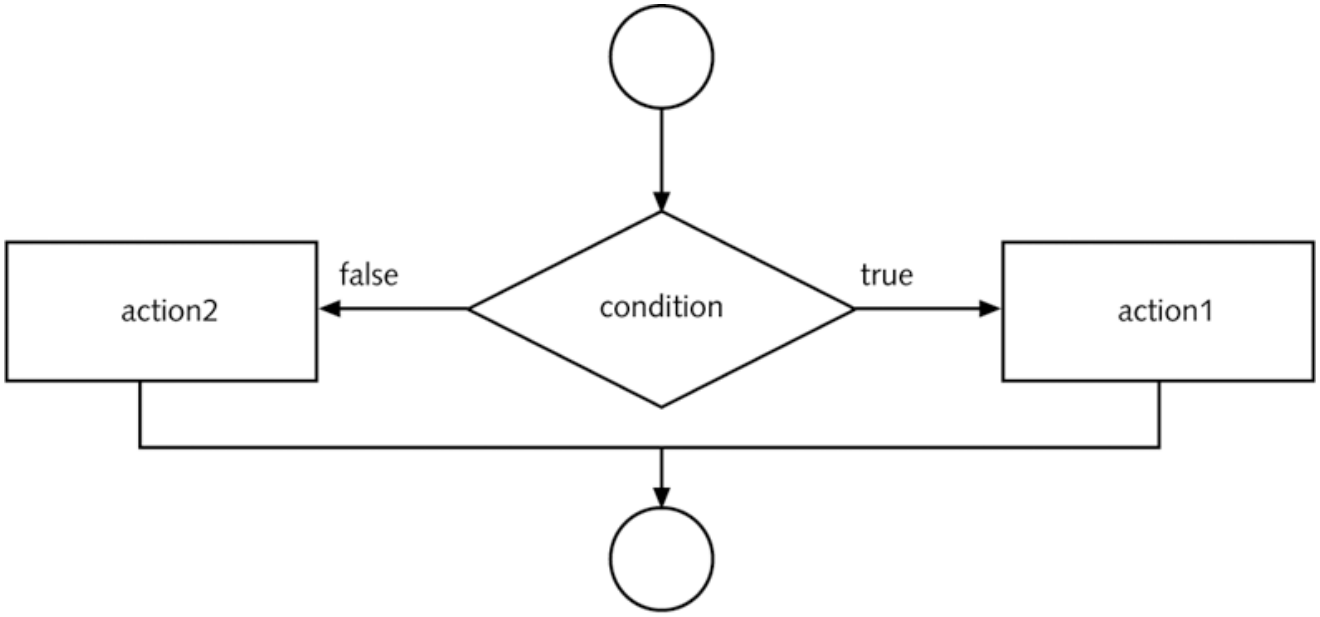


Figure 3.15 Flowchart segment for a two-way selection structure

Compound Statements

- Grouping single statements with braces (‘{ }’)
  - Also called a ‘block’
  - Replaceable with a single statement

```
if ( grade >= 60 )
{
  System.out.println("Passed");
  System.out.println("Scholarship");
}
else
{
   System.out.println("Failed");
   System.out.println("No Scholarship");
}
```

```
[Q] What if no braces?

if ( grade >= 60 )
  System.out.println("Passed");
  System.out.println("Scholarship");
else
  System.out.println("Failed");
  System.out.println(
      "No Scholarship");

➔ Error!!
```

Example: Pay.java

```java
import java.util.Scanner;

public class Pay {
  public static void main ( String[] args ) {
    final int RATE = 5000;
    int pay, hours;
    Scanner input = new Scanner(System.in);

    System.out.print("Enter time: ");
    hours = input.nextInt();

    if ( hours > 8 )
      pay = RATE * 8 + (int) (1.5 * RATE * (hours-8));
    else
      pay = RATE * hours;
    System.out.printf("Salary is %d.\n",pay);
  }
}

/*
$ javac Pay.java
$ java Pay
Enter time: 8
Salary is 40000.
*/
```

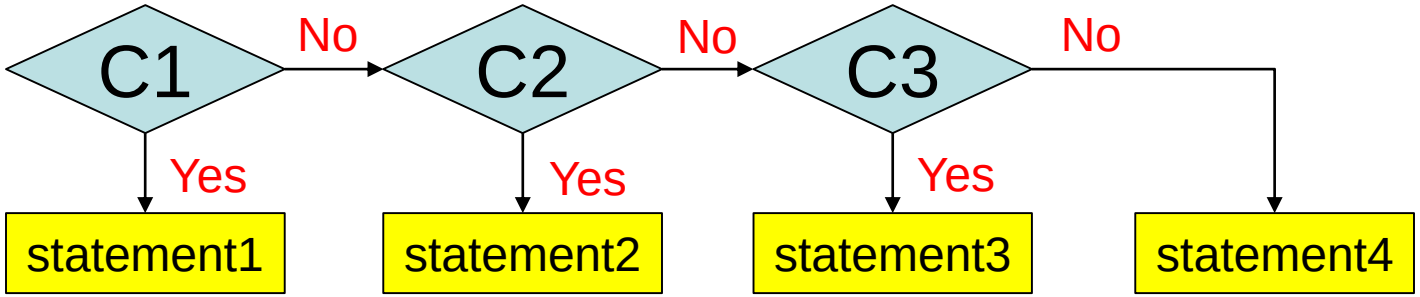
Nested if

```
// 1. single statement if
if ( C1 )
  statement;

// 2. nested if
if ( C1 )
  if ( C2 )
    statement2;     // when both C1 and C2 are true

// 2-b. equivalent to
if ( C1 && C2 )
  statement2;       // when both C1 and C2 are true
```

Nested if

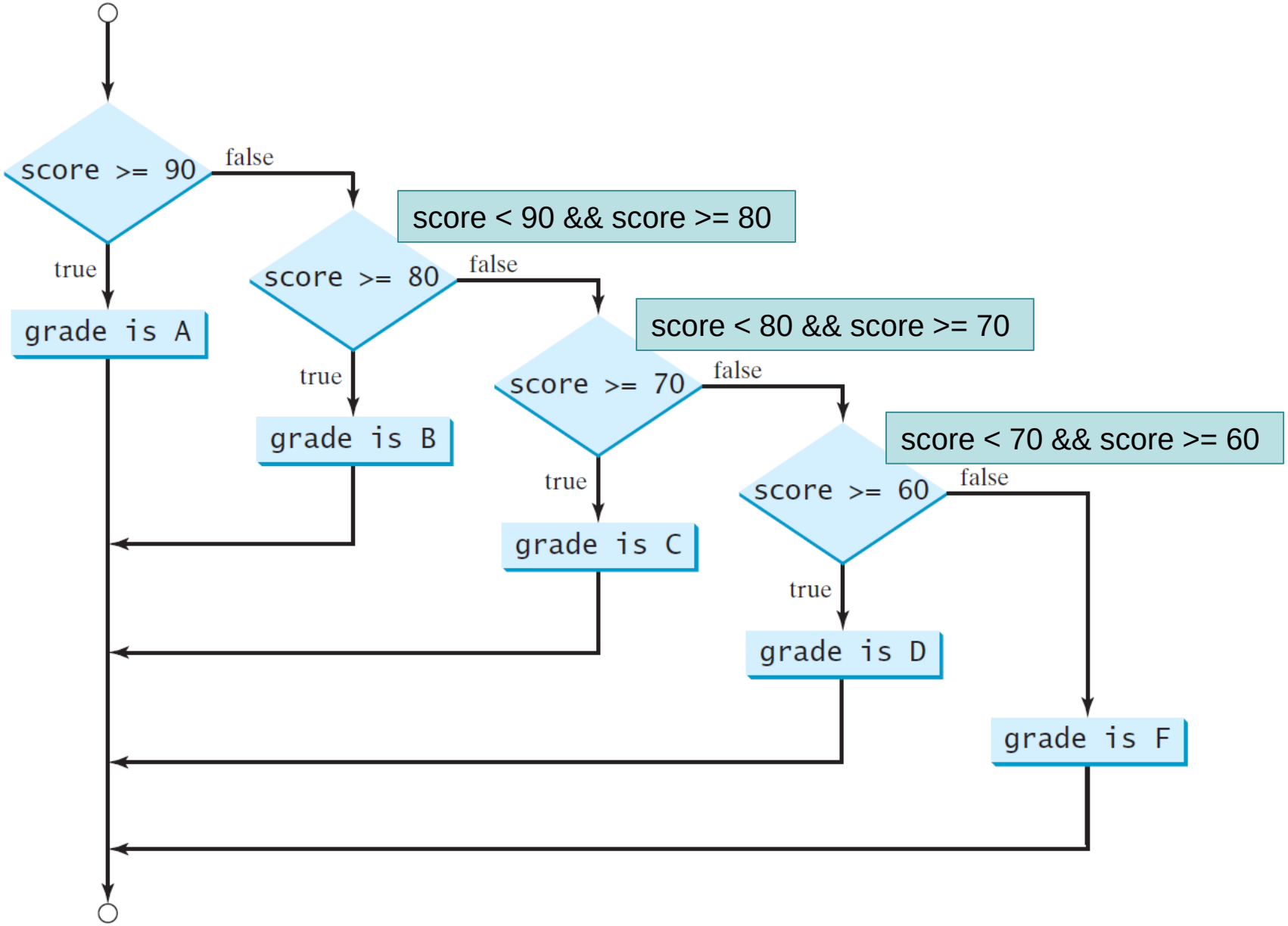
```
// 3. nested if-else
if ( C1 )  // if-1
  if ( C2 )  // if-2
    statement2;      // C1 && C2
  else       // matched to if-2
    statement3;      // C1 && !C2

// 4. nested if-else with braces
if ( C1 )  // if-1
{
  if ( C2 )  // if-2
    statement2;      // C1 && C2
}
else         // matched to if-1
  statement3;        // !C1
```

Nested if: cascaded




```
// 5. cascaded if
if ( C1 )
  statement1;
else if ( C2 )
  statement2;          // !C1 && C2
else if ( C3 )
  statement3;          // !C1 && !C2 && C3
else
  statement4;          // !C1 && !C2 && !C3
```

Cascaded if-else Statements



Cascaded if-else Statements

```java
if (score >= 90.0)
  System.out.print("A");
else
  if (score >= 80.0)
    System.out.print("B");
  else
    if (score >= 70.0)
      System.out.print("C");
    else
      if (score >= 60.0)
        System.out.print("D");
      else
        System.out.print("F");
```

(a)

Equivalent

This is better

```java
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```

(b)

animation

Trace if-else statement

(1) Suppose score is 70.0

(2) The condition is false

```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```

(3) The condition is false

(4) The condition is **true**

(5) Grade is C

(6) Exit the if statement

Tax.java

```java
import java.util.Scanner;

public class Tax {
  public static void main ( String[] args ) {
    int income, tax;
    Scanner input = new Scanner(System.in);

    System.out.print("Enter your income: ");
    income = input.nextInt();

    if ( income <= 1000 )
      tax = (int) (0.09 * income);
    else if ( income <= 4000 )  // income > 1000 && income <= 4000
      tax = (int) (0.18 * income);
    else if ( income < 8000 )   // income > 4000 && income < 8000
      tax = (int) (0.27 * income);
    else        // income >= 8000
      tax = (int) (0.36 * income);

    System.out.printf("Total tax is %d.\n",tax);
  }
}

/*
$ javac Tax.java
$ java Tax
Enter your income: 3000
Total tax is 540.
*/
```

Checkpoints

1. Write a if-else statements that prints “large” if a variable ***n*** is greater than or equal to 100, or prints “small” if ***n*** is less than 100.

2. What is the output of the following code for the values of ***k*** being 3, 0, -1, respectively.

```
if( k == 0 )
    System.out.println("A");
else if( k > 3 )
    System.out.println("B");
else
    System.out.println("C");
```

Note

The else clause matches the most recent if clause in the same block.

```
int i = 1, j = 2, k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
else
    System.out.println("B");
```

(a)

Equivalent

This is better with correct indentation

```
int i = 1, j = 2, k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
  else
    System.out.println("B");
```

(b)

Note, cont.

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.

Common Errors

Adding a semicolon at the end of an <u>if</u> clause is a common mistake.

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

Wrong

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

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

TIP

```
if (number % 2 == 0)
  even = true;
else
  even = false;
```

(a)

Equivalent

```
boolean even
  = number % 2 == 0;
```

(b)

```
if (even == true)
  System.out.println(
    "It is even.");
```

(a)

This is better

Equivalent

```
if (even)
  System.out.println(
    "It is even.");
```

(b)

Logical Operators

Operator	Name	Description
`!`	`not`	logical negation
`&&`	`and`	logical conjunction
`||`	`or`	logical disjunction
`^`	`exclusive or`	logical exclusion

Truth Table for Operator !

p	!p
true	false
false	true

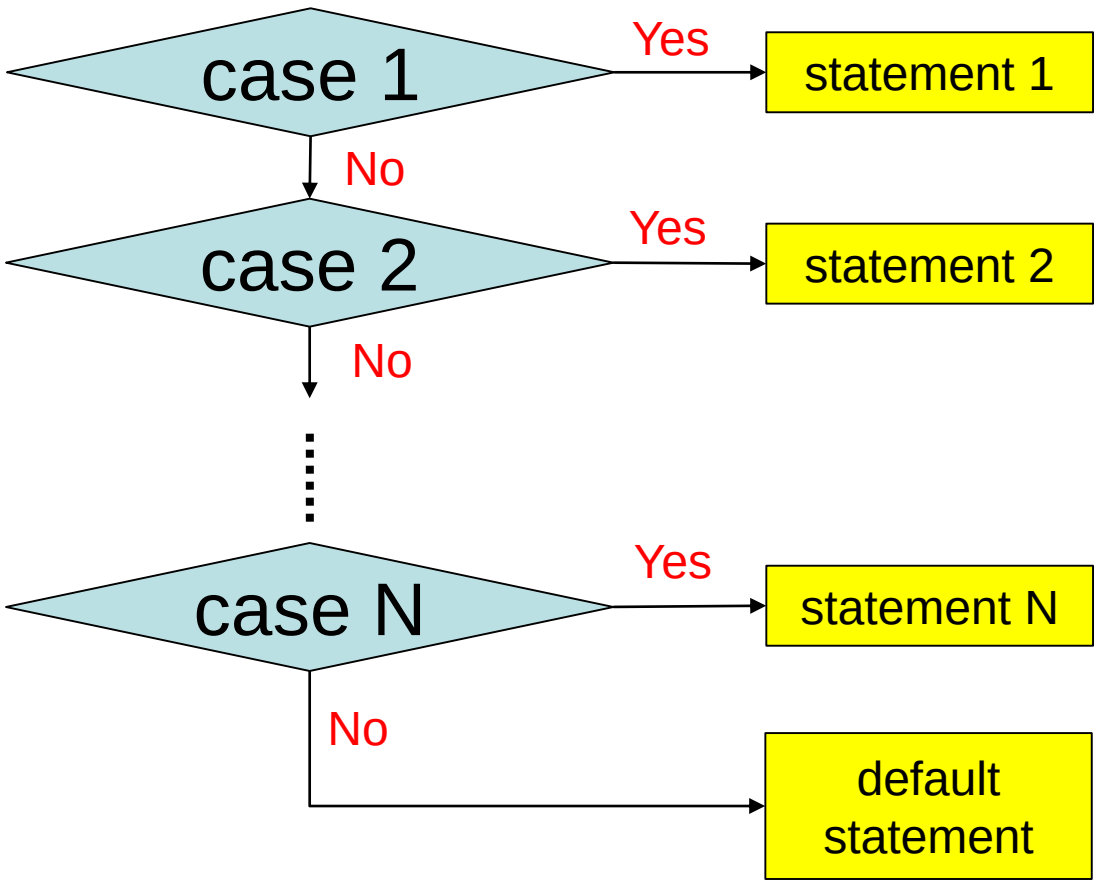
p_1	p_2	p_1 && p_2	p_1 || p_2	p_1 ^ p_2
false	false	false	false	false
false	true	false	true	true
true	false	false	true	true
true	true	true	true	false

Examples

Here is a program that checks whether a number is divisible by 2 and 3, whether a number is divisible by 2 or 3, and whether a number is divisible by 2 or 3 but not both:

```
import java.util.Scanner;

public class TestBooleanOperators {
  public static void main(String[] args) {
    // Create a Scanner
    Scanner input = new Scanner(System.in);

    // Receive an input
    System.out.print("Enter an integer: ");
    int number = input.nextInt();

    if (number % 2 == 0 && number % 3 == 0)
      System.out.println(number + " is divisible by 2 and 3.");

    if (number % 2 == 0 || number % 3 == 0)
      System.out.println(number + " is divisible by 2 or 3.");

    if (number % 2 == 0 ^ number % 3 == 0)
      System.out.println(number +
        " divisible by 2 or 3, but not both.");
  }
}
```

switch-case statement replaces if-else




```
// Java
switch ( a ) {
  case value_1:
     statement_1;
     break;
 case value_2:
     statement_2;
     break;

  …..
 case value_N:
     statement_N;
     break;
  default:
     default_stmt;
}
```

Switch Example – Number

```
import java.util.Scanner;

public class SwitchExample {
  public static void main ( String[] args ) {
    int number;

    Scanner scan = new Scanner(System.in);
    System.out.print("Enter your a number: ");
    number = scan.nextInt();

    switch ( number ) {
      case 0:
        System.out.println("Zero");
        break;
      case 1:
        System.out.println("One");
        break;
      case 2:
        System.out.println("Two");
        break;
      default:
        System.out.println("Many");
        break;  // actually, no need to add this
    }
  }
}
```

Switch Example – String

```java
public class StringSwitch {
  public static void main ( String[] args ) {
    String month = "february";
    int monthNumber;

    // from JDK version 7, string can be used in switch-case statements
    switch ( month ) {
      case "january":
      case "January":   // if break is omitted, logical "or"
        monthNumber = 1;
        break;
      case "february":
      case "February":  // if break is omitted, logical "or"
        monthNumber = 2;
        break;
      case "march":
      case "March":     // if break is omitted, logical "or"
        monthNumber = 3;
        break;
      default:
        monthNumber = 0;
    }
    System.out.println(monthNumber);
  }
}
```

```java
import java.util.Scanner;

public class DaysInMonth {
  public static void main ( String[] args ) {
    int month;
    int year = 2009;
    int days = 0;
    Scanner scan = new Scanner(System.in);

    System.out.print("Enter your a number: ");
    month = scan.nextInt();

    switch ( month ) {
      case 1:  case 3:   case 5:  case 7:
      case 8:  case 10:  case 12:       // "or"
        days = 31;
        break;
      case 4:  case 6:   case 9:  case 11:
        days = 30;
        break;
      case 2:
        if ( ((year%4 == 0) && (year%100 != 0)) || (year%400 == 0) )
          // leap year
          days = 29;
        else
          days = 28;
        break;
      default:
        days = 0;
        System.out.println("Wrong month number");
    }
    System.out.println("Number of days in month " + month + " is " + days);
  }
}
```