SELECTION

(Chapter 2)

Selection

- Very often you will want your programs to make *choices* among different groups of instructions
  - For example, a program processing requests for airline tickets could have the following choices to make:
    - display the price of the seats requested;
    - display a list of alternative flights;
    - display a message saying that no flights are available to that destination.
- **Selection** is a method of program control in which a choice can be made among which instructions to execute

Implementing selection in Java

- In Java there are three forms of selection you can use:
  - an **if** statement
  - an **if…else** statement
  - a **switch** statement

"if" statement

- Sometimes one or more instructions need to be guarded so that they are executed only when appropriate

- This particular form of selection is implemented by making use of Java's **if** statement
  - The general form of an **if** statement is given as follows:

```
if ( /* a test goes here */)
{
    // instruction(s) to be guarded go here
}
```

Tests

- A **test** is any expression that gives a **boolean** result of **true** or **false**. For example,
  - the password is valid;
  - there is an empty seat on the plane;
  - the temperature in the laboratory is too high.

- The instructions inside the braces of the **if** statement are executed
  - when the test gives a **boolean** result of **true**

- The instructions inside the **if** braces are skipped and not executed
  - if the test gives a **boolean** result of **false**

"if" statement: an example

- Consider a temperature value that has been entered by the user
- Let us now check if the temperature is below freezing point (0 degrees) and display an appropriate message if it is
- Regardless of whether or not the temperature was freezing, we will then ask the user to enter another temperature

```
if (temperature < 0) // check if temperature is below freezing
{
   // this line executed only when the test is true
     System.out.println("Temperature is below freezing");
}
// this line is outside the 'if' statement so is always executed
System.out.print("Enter another temperature value ");
```

When there is only a single instruction associated with an if statement, the braces can be omitted if so desired.

"if" statement: another example

- Consider once again the program that calculates the cost of a product
- Now assume that a special promotion is in place for those products with an initial price over 100. For such products the company pays half the tax so, for the customer, the tax is effectively halved
- Write a program that applies this promotion, as well as informs the user that a tax discount has been applied

Program 2.1

```
import java.util.*;
public class FindCostWithDiscount
{
 public static void main(String[] args )
 {
  double price, tax;
  Scanner sc = new Scanner(System.in);
  System.out.println("*** Product Price Check ***");
  System.out.print("Enter initial price: ");
  price = sc.nextDouble();
  System.out.print("Enter tax rate: ");
  tax = sc.nextDouble();
  // the following 'if' statement allows a selection to take place
```

```
   if (price > 100) // test the price to see if a discount applies
   {
       // these two instructions executed only when test is true
       System.out.println
            ("Special Promotion: Your tax will be halved!");
       tax = tax * 0.5;
   }
   // the remaining instructions are always executed
   price = price * (1 + tax/100);
   System.out.println("Cost after tax = " + price);
 }
}
```

Sample runs

```
*** Product Price Check ***
Enter initial price: 50
Enter tax rate: 10
Cost after tax = 55.0

*** Product Price Check ***
Enter initial price: 1000
Enter tax rate: 10
Special Promotion: Your tax will be halved!
Cost after tax = 1050
```

Comparison operators

Table 2.1 The comparison operators of Java	
Operator	**Meaning**
==	equal to
!=	not equal to
<	less than
>	greater than
>=	greater than or equal to
<=	less than or equal to

These comparison operators are used with primitive values only (such as **`int`** and **`double`**) and not with strings.

Checking for equality

- Double equals (==) is used to check for equality in Java and not the single equals (=), which is used for assignment
- To use the single equals is a very common error!
- The following checks if an angle is a right angle:

```
if (angle == 90)// note the use of the double equals
{
   System.out.println("This is a right angle");
}
```

"if … else" statement

- **if** statement allows us to build the idea of a *single* choice into our programs

- **if…else** statement allows us to build the idea of choose *two* alternative courses of action

```
if ( /* test goes here */ )
{
   // instruction(s) if test is true go here
}
else
{
   // instruction(s) if test is false go here
}
```

"if … else" statement: an example

- Write a program that checks a student's exam mark and tells the student whether or not s/he has passed, before displaying a good luck message on the screen

Program 2.2

```
import java.util.*;


public class DisplayResult
{
   public static void main(String[] args)
   {
     int mark;
     Scanner sc = new Scanner (System.in);
     System.out.println("What exam mark did you get?");
     mark = sc.nextInt();
     if (mark >= 40)
     {
```

```
        // executed when test is true
        System.out.println("Congratulations, you passed");
      }
      else
      {
        // executed when test is false
        System.out.println("I'm sorry, but you failed");
      }
      System.out.println("Good luck with your other exams");
    }
}
```

Combining tests

- Often it is necessary to join two or more tests together to a create a single more complicated test

- For example, assume that, for an experiment in the laboratory to be successful, the temperature must remain between 5 and 12 degrees celsius.
  - This involves combining two tests together

```
if (temperature  >= 5 && temperature <= 12)
```

Logical operators

- Symbols that join tests together to form longer tests are known as **logical operators**

Table 2.2 The logical operators of Java	
Logical operator	**Java counterpart**
AND	&&
OR	||
NOT	!

Nested "if … else" statement

- Instructions within **if** and **if…else** statements can themselves be *any* legal Java commands
- In particular, they could contain other **if** or **if… else** statements
- This form of control is referred to as **nesting**
- Nesting allows *multiple* choices to be processed

Nested "if … else" statement: an example

- Write a program that asks a student to enter his/her tutorial group (A, B, or C) and then displays the time of the software lab on the screen

Program 2.4

```
import java.util.*;
public class Timetable
{
   public static void main(String[] args)
  {
      char group; // to store the tutorial group
      Scanner sc = new Scanner (System.in);
      System.out.println("***Lab Times***"); // display header
      System.out.println("Enter your group (A,B,C)");
      group = sc.next().charAt(0);
      // check tutorial group and display appropriate time
      if (group == 'A')
      {
        System.out.print("10.00 a.m"); // lab time for group A
      }
```

```
        else
        {
            if (group == 'B')
            {
                System.out.print("1.00 p.m"); // lab time for group B
            }
            else
            {
                if (group == 'C')
                {
                  System.out.print("11.00 a.m"); //lab time for group C
                }
                else
                {
                   System.out.print("No such group"); // invalid group
                }
            }
        }
    }
}
```

"switch" statement

- A **switch** can sometimes be used instead of a series of nested **if...else** statements

```
switch(someVariable)
{
    case value1:  // instructions(s) to be executed
                  break;
    case value2:  // instructions(s) to be executed
                  break;
    // more  values to be tested can be added
    default: // instruction(s) for default case
}
```

"switch" statement

- **`someVariable`** is the name of the variable being tested. This variable is usually of type of **`int`** or **`char`**, but may also be of type **`long`**, **`byte`**, and **`short`**
- **`value1`**, **`value2`**, etc. are the possible values of that variable
- **`break`** is an optional command that forces the program to skip the rest of the **`switch`** statement
  - If it is not added, not only will the instructions associated with the matching case will be executed, but also, all the instructions associated with all the cases below it
- **`default`** is an optional (last) case that can be considered as an "otherwise" statement
  - Associated statements are executed if none of the cases is true

When to use a "switch" statement

- The **switch** statement works in exactly the same way as a set of nested **if** statements, but is more compact and readable

- A **switch** statement may be used when
  - only one variable is being checked in each condition (in this case every condition involves checking the variable group)
  - the check involves specific values of that variable (e.g., 'A', 'B') and not ranges (e.g., >= 40)

"switch" statement: an example

- Rewrite the following program using a "switch" statement

- Write a program that asks a student to enter his/her tutorial group (A, B, or C) and then displays the time of the software lab on the screen

Program 2.5

```
import java.util.*;

public class TimetableWithSwitch
{
  public static void main(String[] args)
  {
      char group;
      Scanner sc = new Scanner(System.in);
      System.out.println("***Lab Times***");
      System.out.println("Enter your group (A,B,C)");
      group = sc.next().charAt(0);
```

```
        switch(group) // beginning of switch
        {
          case 'A': System.out.print("10.00 a.m ");
                     break;
          case 'B': System.out.print("1.00 p.m ");
             break;
          case 'C': System.out.print("11.00 a.m ");
             break;
          default:  System.out.print("No such group");
        } // end of switch
    }
}
```

Combining options

- Let's assume that both groups A and C have a lab at 10.00a.m
- The following **switch** statement could process this by grouping case 'A' and 'C' together

```
// groups A and C have been processed together
switch(group)
{
        case 'A': case 'C': System.out.print("10.00 a.m ");
                  break;
        case 'B': System.out.print("1.00 p.m ");
                  break;
        default:  System.out.print("No such group");
}
```