THE SWITCH STATEMENT IN JAVA

THE SECOND BRANCHING STATEMENT in Java is the `switch` statement, which is introduced in this section. The `switch` statement is used far less often than the `if` statement, but it is sometimes useful for expressing a certain type of multi-way branch.

The Basic switch Statement

A switch statement allows you to test the value of an expression and, depending on that value, to jump directly to some location within the switch statement. Only expressions of certain types can be used. The value of the expression can be one of the primitive integer types int, short, or byte. It can be the primitive char type. Or, as we will see later in this section, it can be an enumerated type. In Java 7, *Strings* are also allowed. In particular, the expression **cannot** be a real number, and prior to Java 7, it **cannot** be a *String*. The positions that you can jump to are marked with case labels that take the form: "case **constant**:". This marks the position the computer jumps to when the expression evaluates to the given **constant**. As the final case in a switch statement you can, optionally, use the label "default:", which provides a default jump point that is used when the value of the expression is not listed in any case label.

A `switch` statement, as it is most often used, has the form:

```
switch (expression) {
   case constant-1:
      statements-1
      break;
   case constant-2:
      statements-2
      break;
      .
      .   // (more cases)
      .
   case constant-N:
      statements-N
      break;
   default:  // optional default case
      statements-(N+1)
} // end of switch statement
```

The `break` statements are technically optional. The effect of a `break` is to make the computer jump to the end of the switch statement. If you leave out the break statement, the computer will just forge ahead after completing one case and will execute the statements associated with the next case label. This is rarely what you want, but it is legal. (I will note here -- although you won't understand it until you get to the next chapter -- that inside a subroutine, the `break` statement is sometimes replaced by a `return` statement.)

Note that you can leave out one of the groups of statements entirely (including the `break`). You then have two case labels in a row, containing two different constants. This just means that the computer will jump to the same place and perform the same action for each of the two constants.

Here is an example of a switch statement. This is not a useful example, but it should be easy for you to follow. Note, by the way, that the constants in the case labels don't have to be in any particular order, as long as they are all different:

```
switch ( N ) {   // (Assume N is an integer variable.)
   case 1:
      System.out.println("The number is 1.");
      break;
   case 2:
   case 4:
   case 8:
      System.out.println("The number is 2, 4, or 8.");
      System.out.println("(That's a power of 2!)");
      break;
   case 3:
   case 6:
   case 9:
      System.out.println("The number is 3, 6, or 9.");
      System.out.println("(That's a multiple of 3!)");
      break;
   case 5:
      System.out.println("The number is 5.");
      break;
   default:
      System.out.println("The  number  is  7  or  is  outside  the
range 1 to 9.");
}
```

The switch statement is pretty primitive as control structures go, and it's easy to make mistakes when you use it. Java takes all its control structures directly from the older programming languages C and C++. The switch statement is certainly one place where the designers of Java should have introduced some improvements.

3.6.2 Menus and switch Statements

One application of `switch` statements is in processing menus. A menu is a list of options. The user selects one of the options. The computer has to respond to each possible choice in a different way. If the options are numbered 1, 2, ..., then the number of the chosen option can be used in a `switch` statement to select the proper response.

In a *TextIO*-based program, the menu can be presented as a numbered list of options, and the user can choose an option by typing in its number. Here is an example that could be used in a variation of the`LengthConverter` example from the previous section:

```
int optionNumber;    // Option number from menu, selected by
user.
double measurement; // A numerical measurement, input by the
user.
                    //     The unit of measurement depends on
which
                    //    option the user has selected.
double inches;        // The same measurement, converted into
inches.

/* Display menu and get user's selected option number. */

TextIO.putln("What unit of measurement does your input use?");
TextIO.putln();
TextIO.putln("         1.  inches");
TextIO.putln("         2.  feet");
TextIO.putln("         3.  yards");
TextIO.putln("         4.  miles");
TextIO.putln();
TextIO.putln("Enter the number of your choice: ");
optionNumber = TextIO.getlnInt();

/* Read user's measurement and convert to inches. */

switch ( optionNumber ) {
   case 1:
       TextIO.putln("Enter the number of inches: ");
       measurement = TextIO.getlnDouble();
       inches = measurement;
       break;
   case 2:
       TextIO.putln("Enter the number of feet: ");
       measurement = TextIO.getlnDouble();
       inches = measurement * 12;
       break;
   case 3:
       TextIO.putln("Enter the number of yards: ");
       measurement = TextIO.getlnDouble();
       inches = measurement * 36;
```

```
            break;
        case 4:
            TextIO.putln("Enter the number of miles: ");
            measurement = TextIO.getlnDouble();
            inches = measurement * 12 * 5280;
            break;
        default:
            TextIO.putln("Error!  Illegal option number!  I quit!");
            System.exit(1);
    } // end switch

    /* Now go on to convert inches to feet, yards, and miles... */
```

In Java 7, this example might be rewritten using a *String* in the `switch` statement:

```
String units;       // Unit of measurement, entered by user.
double measurement; // A numerical measurement, input by the
user.
double inches;        // The same measurement, converted into
inches.

/* Read the user's unit of measurement. */

TextIO.putln("What unit of measurement does your input use?");
TextIO.put("inches, feet, yards, or miles ?");
units = TextIO.getln().toLowerCase();

/* Read user's measurement and convert to inches. */

TextIO.put("Enter the number of " + units + ":  ");
measurement = TextIO.getlnDouble();

switch ( units ) {  // Requires Java 7 or higher!
    case "inches":
        inches = measurement;
        break;
    case "feet":
        inches = measurement * 12;
        break;
    case "yards":
        inches = measurement * 36;
        break;
    case "miles":
        inches = measurement * 12 * 5280;
        break;
    default:
        TextIO.putln("Wait a minute!   Illegal unit of measure!
I quit!");
        System.exit(1);
} // end switch
```

3.6.3 Enums in switch Statements

The type of the expression in a `switch` can be an enumerated type. In that case, the constants in the `case` labels must be values from the enumerated type. For example, if the type of the expression is the enumerated type *Season* defined by

```
enum Season { SPRING, SUMMER, FALL, WINTER }
```

then the constants in the case label must be chosen from among the values `Season.SPRING`, `Season.SUMMER`, `Season.FALL`, or `Season.WINTER`. However, there is another quirk in the syntax: when an enum constant is used in a `case` label, only the simple name, such as "`SPRING`" can be used, not the full name "`Season.SPRING`". Of course, the computer already knows that the value in the `case` label must belong to the enumerated type, since it can tell that from the type of expression used, so there is really no need to specify the type name in the constant. As an example, suppose that `currentSeason` is a variable of type*Season*. Then we could have the `switch` statement:

```
switch ( currentSeason ) {
   case WINTER:    // ( NOT Season.WINTER ! )
      System.out.println("December, January, February");
      break;
   case SPRING:
      System.out.println("March, April, May");
      break;
   case SUMMER:
      System.out.println("June, July, August");
      break;
   case FALL:
      System.out.println("September, October, November");
      break;
}
```

3.6.4 Definite Assignment

As a somewhat more realistic example, the following `switch` statement makes a random choice among three possible alternatives. Recall that the value of the expression `(int)(3*Math.random())` is one of the integers 0, 1, or 2, selected at random with equal probability, so the `switch` statement below will assign one of the values "`Rock`", "`Scissors`", "`Paper`" to `computerMove`, with probability 1/3 for each case. Although the switch statement in this example is correct, this code segment as a whole illustrates a subtle syntax error that sometimes comes up:

```
String computerMove;
switch ( (int)(3*Math.random()) ) {
   case 0:
      computerMove = "Rock";
```

```
            break;
         case 1:
            computerMove = "Scissors";
            break;
         case 2:
            computerMove = "Paper";
            break;
      }
      System.out.println("Computer's move is " + computerMove);    //
      ERROR!
```

You probably haven't spotted the error, since it's not an error from a human point of view. The computer reports the last line to be an error, because the variable `computerMove` might not have been assigned a value. In Java, it is only legal to use the value of a variable if a value has already been definitely assigned to that variable. This means that the computer must be able to prove, just from looking at the code when the program is compiled, that the variable must have been assigned a value. Unfortunately, the computer only has a few simple rules that it can apply to make the determination. In this case, it sees a `switch`statement in which the type of expression is int and in which the cases that are covered are 0, 1, and 2. For other values of the expression, `computerMove` is never assigned a value. So, the computer thinks`computerMove` might still be undefined after the `switch` statement. Now, in fact, this isn't true: 0, 1, and 2 are actually the only possible values of the expression `(int)(3*Math.random())`, but the computer isn't smart enough to figure that out. The easiest way to fix the problem is to replace the case label `case 2` with `default`. The computer can then see that a value is assigned to `computerMove` in all cases.

More generally, we say that a value has been definitely assigned to a variable at a given point in a program if every execution path leading from the declaration of the variable to that point in the code includes an assignment to the variable. This rule takes into account loops and `if` statements as well as `switch` statements. For example, the following two `if` statements both do the same thing as the `switch` statement given above, but only the one on the right definitely assigns a value to `computerMove`:

```
String computerMove;
int rand;
rand  =  (int)(3*Math.random());
(int)(3*Math.random());
if ( rand == 0 )
   computerMove = "Rock";
else if ( rand == 1 )
   computerMove = "Scissors";
```

```
String computerMove;
int rand;
                rand  =
if ( rand == 0 )
         computerMove =
"Rock";
else if ( rand == 1 )
         computerMove =
"Scissors";
```

```
else if ( rand == 2 )                    else
   computerMove = "Paper";                       computerMove =
"Paper";
```

In the code on the left, the test "`if ( rand == 2 )`" in the final `else` clause is unnecessary because if `rand` is not 0 or 1, the only remaining possibility is that `rand == 2`. The computer, however, can't figure that out.

Source : http://math.hws.edu/javanotes/c3/s6.html