Tutorial Link https://course.testpad.chitkara.edu.in/tutorials/Java: Switch statement/5aa3ef255ead875d7faaa95d

TUTORIAL

Java: Switch statement

Topics

1.4 Statements to change control flow

Switch statement is alternative for multiple if-else statements. The if/else construct can be cumbersome when you have to deal with multiple selections with many alternatives. Java has a switch statement that is exactly like the switch statement in C and C++, warts and all.

For example, if you set up a menu system with four alternatives you could use code that looks like this:

Java
```
import java.util.Scanner;

class Main
{
  public static void main (String[] args)
  {
    int choice;
    System.out.print("Select an option (1, 2, 3, 4) ");
    choice = 2;
    switch (choice)
    {
      case 1:           // statements when choice =1
        System.out.println("This is for 1st Choice.");
        System.out.println("Second print message");
        break;
      case 2:           // statements when choice =2
        System.out.println("This is for 2nd Choice.");
        System.out.println("Second print message");
        break;
```

```java
      case 3:           // statements when choice =3
        System.out.println("This is for 3rd Choice.");
        System.out.println("Second print message");
        break;
      case 4:           // statements when choice =4
        System.out.println("This is for 4th Choice.");
        System.out.println("Second print message");
        break;
      default:          // bad input
        System.out.println("This is for bad Choice.");
        break;
    }
    System.out.println("This is print statement after Switch.");
  }
}
```

The case labels must be integers or enumerated constants. You cannot test strings. For example, the following is an error:

```java
String input = "Yes";
switch (input)
{            // ERROR
   case "Yes": // ERROR
      // Statements
      break;
   . . .
}
```

Statements to change control flow

Although the designers of Java kept the goto as a reserved word, they decided not to include it in the language. In general, goto statements are considered poor style. Java supports break statement to break out of a loop to terminate the loop. The same break statement that you use to exit a switch can also be used to break out of a loop. For example:

```java
while (n <= 100)
{
System.out.println(“Number is “ + n);
if (test_condition) break;
n++;
}
```

Now the loop is exited if either n > 100 occurs at the top of the loop or test_condition evaluates to true in if condition occurs in the middle of the loop.

Unlike C++, Java also offers a labeled break statement that lets you break out of multiple nested loops. Occasionally something weird happens inside a deeply nested loop. In that case, you may want to break completely out of all the nested loops. It is inconvenient to program that simply by adding extra conditions to the various loop tests. Here's an example that shows the break statement at work. Notice that the label must precede the outermost loop out of which you want to break. It also must be followed by a colon.

```
int n = 2;
break_label:
while (. . .)
{           // this loop statement is tagged with the label
        . . .
        for (. . .)
        {           // this inner loop is not labeled
                System.out.print("Enter a number >= 0: ");
                n = in.nextInt();
                if (n < 0) // should never happen—can't go on
                        break break_label;
// break out of break_label loop
                . . .
        }
}
// this statement is executed immediately after the labeled break
```

Finally, there is a continue statement that, like the break statement, breaks the regular flow of control. The continue statement transfers control to the header of the innermost enclosing loop. Here is an example:

```
while (i<10)
{
 if (i == 5) continue;
```

```
  sum += n; // not executed if n == 5
}
```

If n == 5, then the continue statement jumps immediately to the loop header, skipping the remainder of the current iteration. If the continue statement is used in a for loop, it jumps to the "update" part of the for loop. For example, consider this loop:

```
for (count = 1; count <= 100; count++)
{
        System.out.print("Enter a number, -1 to quit: ");
        n = in.nextInt();
        if (n < 0) continue;
        sum += n; // not executed if n < 0
}
```

If n < 0, then continue statement jumps to the count++ statement. There is also a labeled form of the continue statement that jumps to the header of the loop with the matching label.