Switch-case construct

The **switch** statement is Java's multi-way branch statement. It provides an easy way to dispatch execution to different parts of your code based on the value of an expression.

Objectives

Upon completion of this topic, we will be able to:

- Learn the syntax of switch statement
- Understand when to apply switch-case blocks

Switch Statement

As such, it often provides a better alternative than a large series of **if-else-if** statements.

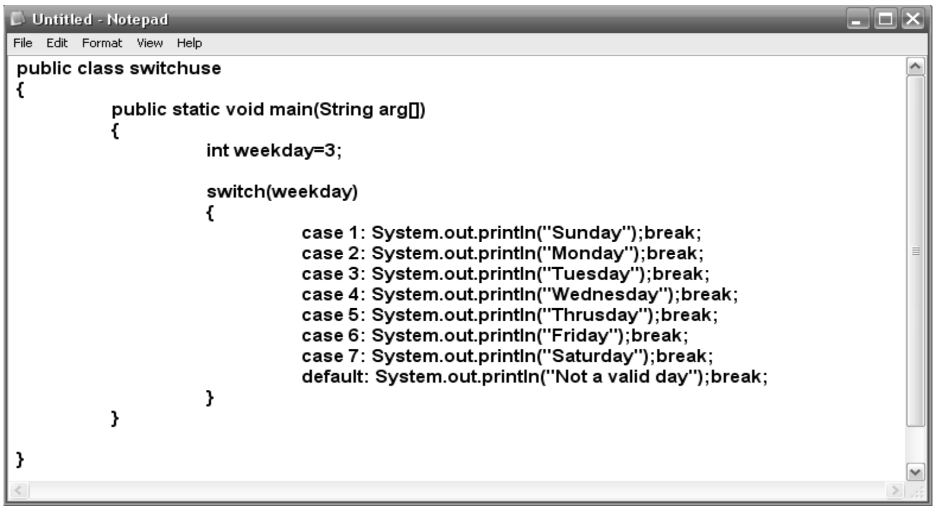
Here is the general form of a **switch** statement:

```
switch (expression) {

case value1:

        // statement sequence

        break;

case value2:

        // statement sequence

        break;

.

.

.

case valueN:

        // statement sequence

        break;

default:
        // default statement sequence

    }
```

The *expression* must be of type **byte**, **short**, **int**, or **char**; each of the *values* specified in the **case** statements must be of a type compatible with the expression. Each **case** value must be a unique literal (it must be a constant, not a variable). Duplicate **case** values are not allowed.

The **switch** statement works as follows: The value of the expression is compared with each of the literal values in the **case** statements. If a match is found, the code sequence following that **case** statement is executed. If none of the constants matches the value of the expression, then the **default** statement is executed. However, the **default** statement is optional. If no **case** matches and no **default** is present, then no further action is taken.

The **break** statement is used inside the **switch** to terminate a statement sequence. When a **break** statement is encountered, execution branches to the first line of code that follows the entire **switch** statement. This has the effect of "jumping out" of the **switch**.



```
public class switchuse
{
        public static void main(String arg[])
        {
                int weekday=3;

                switch(weekday)
                {
                        case 1: System.out.println("Sunday");break;
                        case 2: System.out.println("Monday");break;
                        case 3: System.out.println("Tuesday");break;
                        case 4: System.out.println("Wednesday");break;
                        case 5: System.out.println("Thrusday");break;
                        case 6: System.out.println("Friday");break;
                        case 7: System.out.println("Saturday");break;
                        default: System.out.println("Not a valid day");break;
                }
        }

}
```

The switch…case statement example

The **break** statement is optional. If you omit the **break**, execution will continue on into the next **case**. It is sometimes desirable to have multiple **cases** without **break** statements between them. For example, consider the following program:

```
    // In a switch, break statements are optional.

    class MissingBreak {

        public static void main(String args[ ]) {

            for(int i=0; i<12; i++)

                switch(i) {
                        case 0:
                        case 1:
                        case 2:
                        case 3:
                        case 4:
                                System.out.println("i is less than 5");
                                break;
                        case 5:
                        case 6:
                        case 7:
                        case 8:
                        case 9:
                                System.out.println("i is less than 10");
                                break;
                        default:
                                System.out.println("i is 10 or more");
                }
        }
}
```

This program generates the following output:

```
i is less than 5
i is less than 5
i is less than 5
i is less than 5
i is less than 5
i is less than 10
```

```
i is less than 10

i is less than 10

i is less than 10

i is less than 10

i is 10 or more

i is 10 or more
```

Nested switch Statements

You can use a **switch** as a part of the statement sequence of an outer **switch**. This is called a *nested* **switch**. Since a **switch** statement defines its own block, no conflicts arise between the **case** constants in the inner **switch** and those in the outer **switch**. For example, the following fragment is perfectly valid:

```
switch(count) {
case 1:
    switch(target) { // nested switch
    case 0:
        System.out.println("target is zero");
        break;
    case 1: // no conflicts with outer switch
        System.out.println("target is one");
        break;
    }
    break;
case 2: // ...
```

Here, the **case 1:** statement in the inner switch does not conflict with the **case 1:** statement in the outer switch. The **count** variable is only compared with the list of cases at the outer level. If the **count** is 1, then **target** is compared with the inner list cases.

To summarize it all, there are three important features of the **switch** statement to note:

- The **switch** differs from the **if** in that **switch** and can only test for equality, whereas **if** can evaluate any type of **Boolean** expression. That is, the **switch** looks only for a match between the value of the expression and one of its **case** constants.
- No two **case** constants in the same **switch** can have identical values. Of course, a **switch** statement enclosed by an outer **switch** can have **case** constants in common.
- A **switch** statement is usually more efficient than a set of nested **if'**s.

The last point is particularly interesting because it gives insight into how the Java compiler works. When it compiles a **switch** statement, the Java compiler will inspect each of the **case** constants and create a "**jump table**" that it will use for selecting the path of execution depending on the value of the expression. Therefore, if you need to select among a large group of values, a **switch** statement will run much faster than the equivalent logic coded using a sequence of **if-else**. The compiler can do this because it knows that the **case** constants are all of the same type and must be compared for equality with the **switch** expression. The compiler has no such knowledge of a long list of **if** expressions.

Summary

Here are the key takeaways:

- ➢ The switch statement is Java's multi-way branch statement.
- ➢ The **break** statement is used inside the **switch** to terminate a statement sequence.
- ➢ A **switch** statement is usually more efficient than a set of nested **if**'s.