THE JAVA SWITCH STATEMENT

Like the `if-else` statement, the `switch` statement introduces a "fork in the road" for the program's execution path. Unlike the `if-else` (1) the switch statement's fork has many tines and (2) these execution paths are not necessarily separate because the flow of execution can end up moving through several of them.

Java Switch Statement Syntax

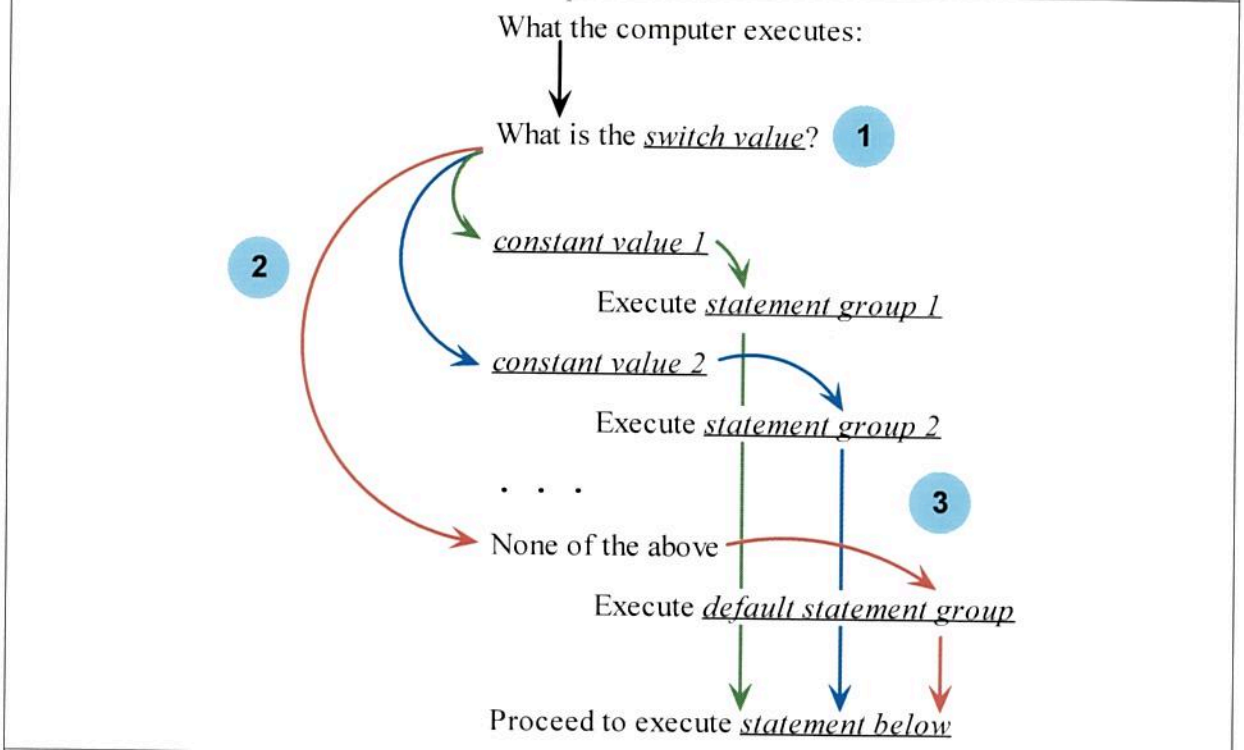
```
switch ( switch value )
{
    case constant value 1 :
        statement group 1
    case constant value 2 :
        statement group 2
    . . .
}
statement below
```

```
switch ( switch value )
{
    case constant value 1 :
        statement group 1
    case constant value 2 :
        statement group 2
    . . .
    default :
        default statement group
}
statement below
```

switch value is a variable or expression that is evaluated at run-time. It must be of the data type `byte`, `short`, `int`, `long`, `char` or, as of Java 7, `String`.

Each *constant value* must be determinable at compile time.[1] Also, each must be of the same data type as the *switch value* and no two of them can have the same value. It can be a literal, a previously declared constant identifier or an expression involving literals and constant identifiers.

Java Switch Statement Semantics



As illustrated in the picture, the computer begins execution of the `switch` statement by (1) evaluating the *switch value*. The computer (2) branches to the *statement group* immediately following the *constant value* that matches the *switch value* and (3) continues executing each subsequent *statement group*. If no *constant value* matches the *switch value* then execution branches to the *default statement group*, if there is one.

The switch Statement

- The switch statement takes the form:

```
switch (SwitchExpression)
{
  case CaseExpression:
    // place one or more statements here
    break;
  case CaseExpression:
    // place one or more statements here
    break;
default:
    // place one or more statements here
}
```

SwitchDemo.java

```
// Determine the number entered.
    switch (number)
    {
      case 1:
        System.out.println("You entered 1.");
        break;
      case 2:
        System.out.println("You entered 2.");
        break;
      case 3:
        System.out.println("You entered 3.");
        break;
      default:
        System.out.println("That's not 1, 2, or 3!");
    }
```

int numbee = value of any below ;

PetFood.java

```
switch(foodGrade)
    {
      case 'a': case 'A':
        System.out.println("30 cents per lb.");
        break;
      case 'b': case 'B':
        System.out.println("20 cents per lb.");
        break;
      case 'c': case 'C':
        System.out.println("15 cents per lb.");
        break;
      default:
        System.out.println("Invalid choice.");
    }
```

Example

This code illustrates the semantics of the `switch` statement without a `default` part. The `switch` statement takes an integer representing a month (1 for January, 2 for February, etc.) and calculates the number of days in all the months prior to and including it.

```
 1  int totalDays = 0;
 2  switch ( month )
 3  {
 4     case 12:
 5        totalDays += 31;
 6     case 11:
 7        totalDays += 30;
 8     case 10:
 9        totalDays += 31;
10     case 9:
11        totalDays += 30;
12     case 8:
13        totalDays += 31;
14     case 7:
15        totalDays += 31;
16     case 6:
17        totalDays += 30;
18     case 5:
19        totalDays += 31;
20     case 4:
21        totalDays += 30;
22     case 3:
23        totalDays += 31;
24     case 2:
25        totalDays += 28;
26     case 1:
27        totalDays += 31;
28  }
29  System.out.println( "days = " + totalDays );
```

If `month` is 4, the `switch` executes lines 21-27. The output is:	If `month` is 12, the `switch` executes lines 5-27. The output is:	If `month` is 13, the `switch` skips everything. The output is:
`days = 120`	`days = 365`	`days = 0`

Example

This code illustrates the use of the **break** statement to insure that each *statement group* is executed at most once. The **switch** statement takes a **String** object representing a student's grade and prints his or her award.

```
 1  switch ( grade )
 2  {
 3     case "A+" :
 4        System.out.println( "Highest honors" );
 5        break;
 6     case "A" :
 7     case "A-":
 8        System.out.println( "Honors" );
 9        break;
10     case "B+":
11     case "B" :
12        System.out.println(  "Favorable Mention" );
13        break;
14  }
15  . . .
```

If **grade** is **A+**, the **switch** executes lines 4 and 5. Line 5 breaks to line 15. The output is:	If **grade** is **A-**, the **switch** executes lines 8 and 9. Line 9 breaks to line 15. The output is:	If **grade** is **B+**, the **switch** executes lines 12 and 13. Line 13 breaks to line 15. The output is:
`Highest honors`	`Honors`	`Favorable Mention`

In the example above, the **break** at line 13 is not necessary since (without it) line 15 is the next to execute. A seasoned Java programmer includes it, anticipating that he or she may later add additional cases to the **switch** statement.

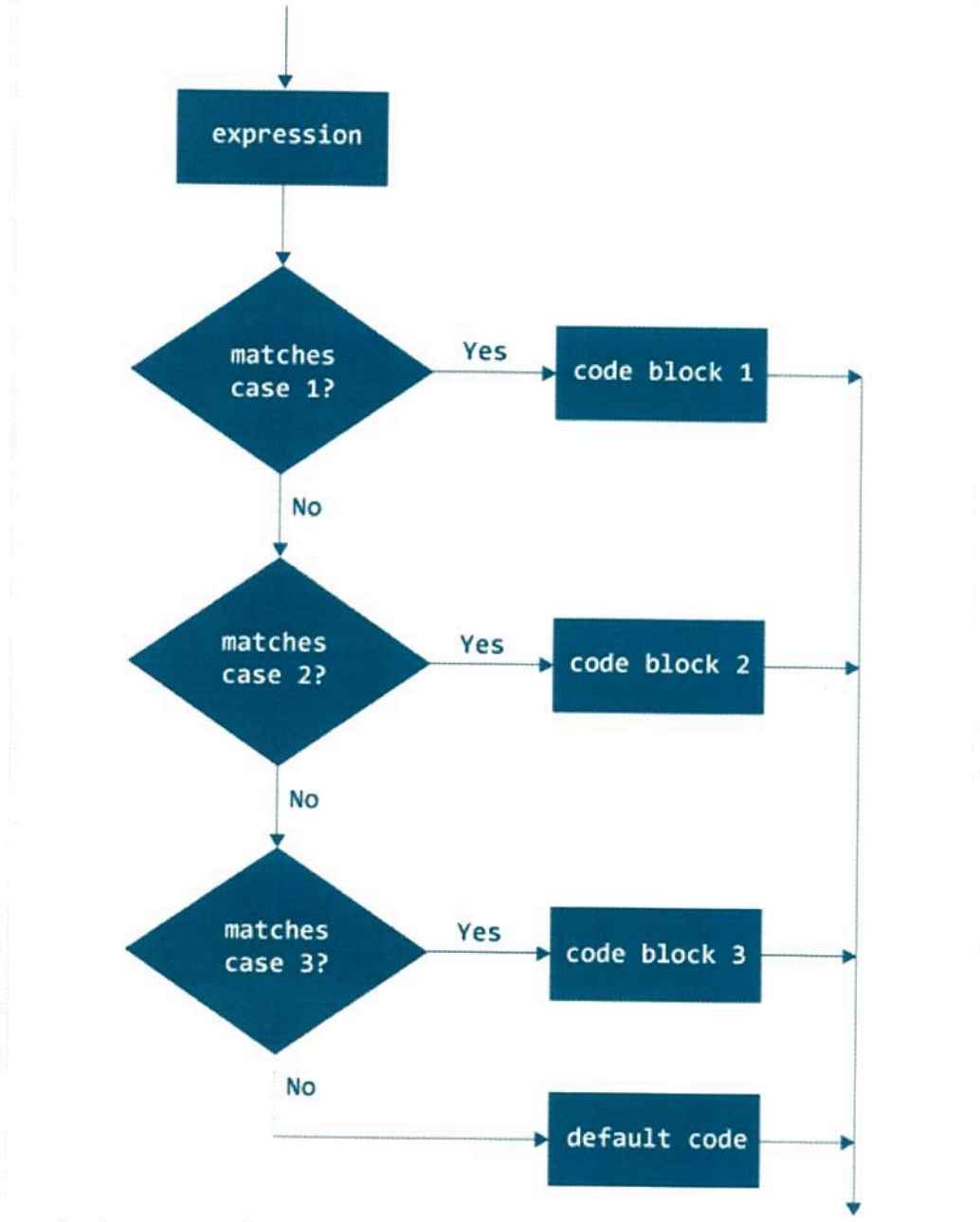
Example

This code illustrates the semantics of the `switch` statement with a `default` part. The switch statement takes an integer representing a month (1 for January, 2 for February, etc.) and sets the variable `lastDate` to its last date, following the old rhyme:

Thirty days hath September, April, June and November.
All the rest have 31, save February alone,
Which has 28 and, in leap year, 29.

```
 1  switch ( month )
 2  {
 3     case  9:
 4     case  4:
 5     case  6:
 6     case 11:
 7        lastDate = 30;
 8        break;
 9     case 2:
10        if ( new GregorianCalendar( ).isLeapYear( year ) )
11           lastDate = 29;
12        else
13           lastDate = 28;
14        break;
15     default:
16        lastDate = 31;
17        break;
18  }
19  System.out.println( "last date = " + lastDate );
```

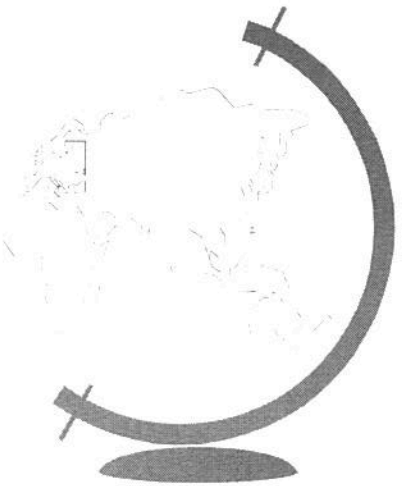
If `month` is 9, the code executes lines 7, 8 and 19, outputting:	If `month` is 2, the code executes lines 10, 11, 14 and 19, outputting, if `year` is a leap year:	If `month` is 12, the code executes lines 16, 17 and 19, outputting:
`last date = 30`	`last date = 29`	`last date = 31`



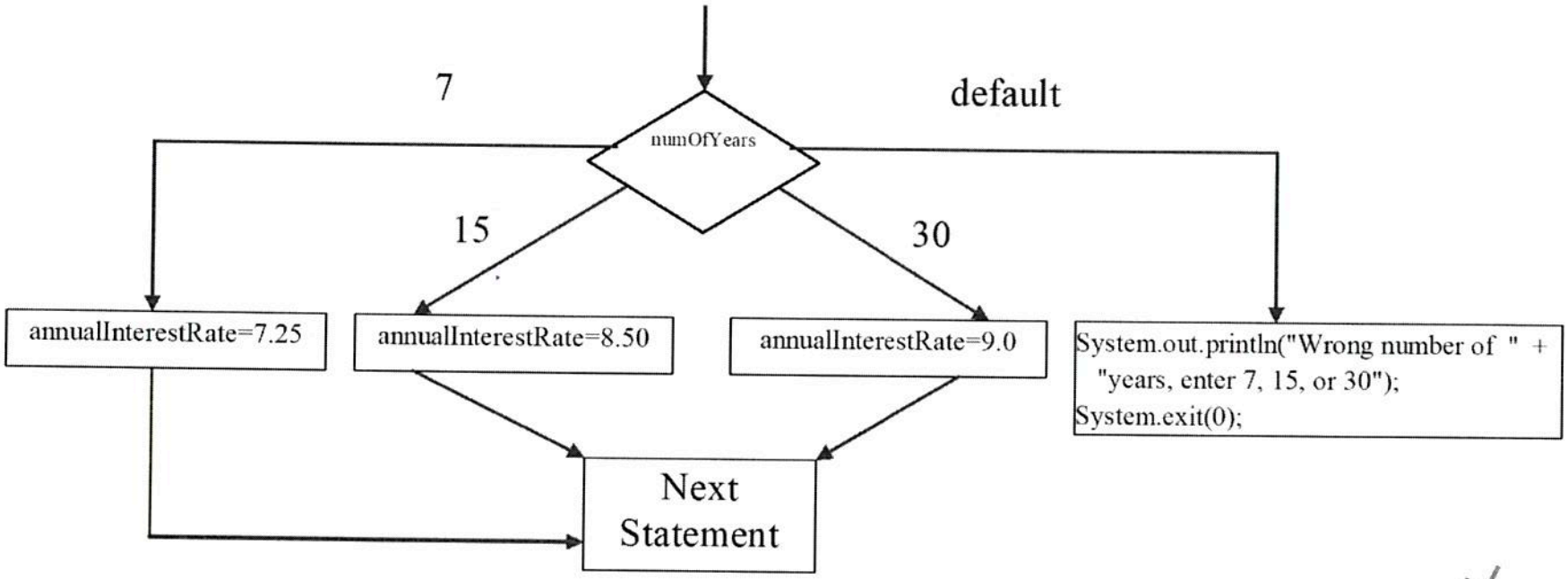
It's also important to note that `switch` statement in Java only works with:

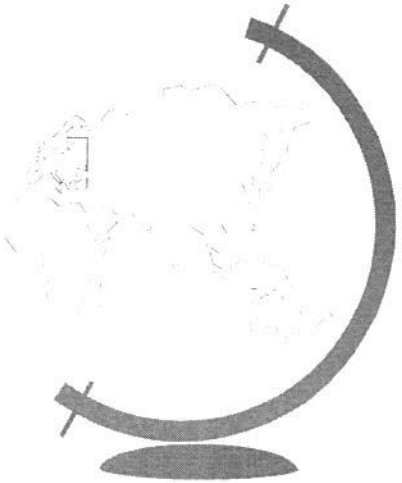
`switch` Statements

```
switch (year) {
  case 7:  annualInterestRate = 7.25;
           break;
  case 15: annualInterestRate = 8.50;
           break;
  case 30: annualInterestRate = 9.0;
           break;
  default: System.out.println(
   "Wrong number of years, enter 7, 15, or 30");
}
```



`switch` Statement Flow Chart





Exercises

For each code fragment below, give the output if the `char` variable `letter` has the value `a`. If `letter` has the value `b`. If `letter` is `c`. If `letter` is `d`. If `letter` is `e`.

1.
```
char letter;
. . .
int  r = 0;
switch ( letter )
{
   case 'a':
      r += 1;
   case 'b':
   case 'c':
      r += 2;
   case 'd':
      r += 3;
}
System.out.println( "r = " + r );
```

2.
```
char letter;
. . .
int  r = 0;
switch ( letter )
{
   case 'a':
      r += 1;
      break;
   case 'b':
   case 'c':
      r += 2;
      break;
   case 'd':
      r += 3;
      break;
}
System.out.println( "r = " + r );
```

For each code fragment below, give the output if the `char` variable `letter` has the value `a`. If `letter` has the value `b`. If `letter` is `c`. If `letter` is `d`. If `letter` is `e`.

3.
```
char letter;
. . .
int r = 0;
switch ( letter )
{
   case 'a':
      r += 1;
      break;
   case 'b':
   case 'c':
      r += 2;
      break;
   case 'd':
      r += 3;
      break;
   default:
      r += 4;
      break;
}
System.out.println( "r = " + r );
```

For each `switch` statement below, circle what's wrong and explain. None of them is correct.

4.
```
int code;
. . .
Switch ( code ) ;
{
   Case 0:
      msg = "System operating normally";
      break;
   Case 1:
      msg = "System startup error";
      break;
}
```

For each **switch** statement below, circle what's wrong and explain. None of them is correct.

5.
```
short code;
. . .
switch ( code )
   case 0
      msg = "System operating normally";
      break;
   case 1
      msg = "System startup error";
      break;
```

6.
```
final byte ON = 0;
final byte OFF = 1;
byte code;
. . .
switch code {
   case ON:
      msg = "System operating normally";
      break;
   case OFF:
      msg = "System startup error";
      break;
}
```

7.
```
double code;
. . .
switch ( code ) {
   case 0: msg = "System operating normally";
      break;
   case 1: msg = "System startup error";
      break;
}
```

For each **switch** statement below, circle what's wrong and explain. None of them is correct.

8.

```
int code, on, off;
. . .
switch ( code )
{
   case on:
      msg = "System operating normally";
      break;
   case off:
      msg = "System startup error";
      break;
}
```

9. Explain the logic error in this **switch** statement.

```
char code;
. . .
code = '1';
. . .
switch ( code )
{
   case 0:
      msg = "System operating normally";
      break;
   case 1:
      msg = "System startup error";
      break;
}
```

For each of the following write the Java code fragment that uses a **switch** statement to accomplish the result.

10. Given an **int** variable **day** equal to 1 through 7, set a string variable to the name of the weekday Sunday, Monday, etc.

11. Given the **String** variable **day** holding the name of the weekday Sunday, Monday, etc., set an **int** variable to 1, 2, etc.

Beginner Errors on Switch Statements

A common beginner error is to forget to break out of a case; thus, executing all the subsequent cases.

Example

The code segment below (incorrectly) attempts to simulate the toss of a coin by choosing 0 or 1 at random and setting `coinToss` to *HEADS* or *TAILS*, respectively. `coinToss` always comes out *TAILS*. If 0 is chosen, lines 4 and 6 are executed. If 1 is chosen, line 6 is executed.

```
1   switch ( (int)(Math.random( )*2) )
2   {
3      case 0:
4         coinToss = "HEADS";
5      case 1:
6         coinToss = "TAILS";
7   }
```

Another common error is to try to `switch` on a variable that is not of the correct data type.

Example

For the code fragment below, the compiler issues the diagnostic:

```
MyApp.java:30: error: incompatible types
    switch (success)
            ^
      required: int
      found:    boolean
```

```
10  boolean success;
~   . . .
30  switch ( success )
31  {
~      . . .
44  }
```