Programming in Java

Topic: Control Flow Statements (Selection and Iteration)

CONTROL STATEMENTS

SELECTION STATEMENTS

- Java supports two selection statements: **if** and **switch.**

if statement

if *(condition)* statement1;
else statement2;

- Each statement may be a single statement or a compound statement enclosed in curly braces (block).
- The condition is any expression that returns a **boolean value.**
- **The else** clause is optional.
- If the condition is true, then statement1 is executed. Otherwise, statement2 (if it exists) is executed.
- In no case will both statements be executed.

Example

```
// To show the working of  if else statement
class Example
        {
                public static void main(String args[])
                {
                   int number=13;
                   if(number%2==0)
                   {
                      System.out.println("even number");
                   }
                else
                   {
                      System.out.println("odd number");
                   }
                }
        }
```

Java if-else-if ladder Statement

```
if(condition1)
{
//code to be executed if condition1 is true
}
else if(condition2)
{
//code to be executed if condition2 is true
}
else if(condition3)
{
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```

Example

```
// To show the working of  if else if else ladder stateent
class Example
        {
                public static void main(String args[])
                {
                   int number=13;
                   if(number>0)
                                System.out.println("Positive number");
                   else if(number<0)
                      System.out.println("Negative Number");
                   else
                            System.out.println("Number is zero");
                }
        }
```

Nested ifs

- A nested if is an if statement that is the target of another if or else.
- In nested ifs an else statement always refers to the nearest if statement that is within the same block as the else and that is not already associated with an else.

```
if(i == 10) {
if(j < 20) a = b;
if(k > 100) c = d; // this if is
else a = c; // associated with this else
}
else a = d; // this else refers to if(i == 10)
```

Nested ifs-Example

```
//Java Program to demonstrate the use of Nested If Statement.
public class JavaNestedIfExample {
public static void main(String[] args) {
    //Creating two variables for age and weight
    int age=20;
    int weight=80;
    //applying condition on age and weight
    if(age>=18){
        if(weight>50){
            System.out.println("You are eligible to donate blood");
        }
    }
}}
```

switch

- The switch statement is Java's multi-way branch statement.
- provides an easy way to dispatch execution to different parts of your code based on the value of an expression.
- provides a better alternative than a large series of **if-else-if statements.**

```
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, char or String.
- 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 (i.e. constant not variable).
- Duplicate case values are not allowed.
- 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.
- 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.

```
class SampleSwitch {
    public static void main(String args[]) {
        for(int i=0; i<6; i++)
            switch(i) {
                case 0:
                    System.out.println("i is zero.");
                    break;
                case 1:
                    System.out.println("i is one.");
                    break;
                case 2:
                    System.out.println("i is two.");
                    break;
                default:
                    System.out.println("i is greater than 2.");
                }
            }
        }
```

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
```

As you can see, execution falls through each **case** until a **break** statement (or the end of the **switch**) is reached.

Beginning with JDK 7, you can use a string to control a **switch** statement.


```
// Use a string to control a switch statement.
class StringSwitch {
public static void main(String args[]) {
String str = "two";
switch(str) {
case "one":
System.out.println("one");
break;
case "two":
System.out.println("two");
break;
case "three":
System.out.println("three");
break;
default:
System.out.println("no match");
break;
}
}
}
```

Nested switch Statements

- When a switch is used as a part of the statement sequence of an outer switch. This is called a nested switch.

```
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: // ...
```

Difference between ifs and switch

- Switch 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.

- A switch statement is usually more efficient than a set of nested ifs.

- Note: No two case constants in the same switch can have identical values. Of course, a switch statement and an enclosing outer switch can have case constants in common.

Q1

What will be the output of following code?

```
public class First
{
public static void main(String[] args)
{
boolean x=true;
int a=10;
if(x)
a++;
else
a--;
System.out.println(a);
}
}
```

A. 10
B. 11
C. 0
D. Error

Q2

CODERINDEED

OUTPUT??

```
public class First
{
public static void main(String[] args)
{
float x=3.45f;
if(x==3.45)
System.out.println("Hello");
else
System.out.println("World");
}
}
```

A. Hello
B. World
C. Error
D. Nothing will be displayed

Q3

CODERINDEED

OUTPUT??

```
public class First
{
public static void main(String[] args)
{
int a=10;
if(a==11);
System.out.println(++a);
}
}
```

A. 11
B. 10
C. Error
D. Nothing will be displayed

Q4

OUTPUT??

```
public class First
{
public static void main(String[] args)
{
int a=9;
if(a>9)
if(a%2==0)
System.out.println("Hi");
else
System.out.println("Hello");
else
System.out.println("Bye");
}
}
```

A. Hi
B. Hello
C. Bye
D. Error

Q5

OUTPUT??

```
public class First
{
public static void main(String[] args)
{
int a=1;
switch(a)
{
case 1:
a=a+2;
case 2:
a=a*3;
case 3:
a=a/2;
break;
case 4:
a=100;
break;
default:
a=-99;
}
System.out.println(a);
}
}
```

A. 3

B. 4

C. 100

D. -99

Q6

OUTPUT??

```
public class First
{
public static void main(String[] args)
{
int a=1;
switch(a)
{
case 1:
a=12;
break;
case 2-1:
a=13;
break;
case 3:
a=14;
break;
}
System.out.println(a);
}
}
```

A. 12
B. 13
C. Error
D. 14

Q7

CODERINDEED

OUTPUT??

```
public class First
{
public static void main(String[] args)
{
char c='B';
switch(c)
{
case 65:
System.out.print("1");
case 66:
System.out.print("2");
case 67:
System.out.print("3");
break;
}
}
}
```

A. 123
B. 12
C. 23
D. Error

ITERATION STATEMENTS (LOOPS)

Iteration Statements

- In Java, iteration statements (loops) are:
  - for
  - while, and
  - do-while

- A loop repeatedly executes the same set of instructions until a termination condition is met.

while Loop

- while loop repeats a statement or block while its controlling expression is true.
- The condition can be any Boolean expression.
- The body of the loop will be executed as long as the conditional expression is true.
- When condition becomes false, control passes to the next line of code immediately following the loop.

```
while(condition)
  {
    // body of loop
  }
```

```
class While
    {
        public static void main(String args[]) {
        int n = 10;
        char a = 'G';
        while(n > 0)
                {
                  System.out.print(a);
                  n--;
                  a++;
                }
            }
    }
```

Output:

GHIJKLMNOP

- The body of the loop will not execute even once if the condition is false.
- While loop starts with the checking of condition. If it evaluated to true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason it is also called **Entry control loop**
- The body of the while (or any other of Java's loops) can be empty. This is because a null statement (one that consists only of a semicolon) is syntactically valid in Java.
- Example in next slide:

while loop no body of execution

```
public class NoBody {
public static void main(String args[]) {
int i, j;
i = 100;
j = 200;
// find midpoint between i and j
while(++i < --j); // no body in this loop
System.out.println("Midpoint is " + i);//150 is output[i=150,j=150]
}
}
```

do-while

- The do-while loop always executes its body at least once, because its conditional expression is at the bottom of the loop.

```
do {
        // body of loop
} while (condition);
```

- Each iteration of the do-while loop first executes the body of the loop and then evaluates the conditional expression.
- If this expression is true, the loop will repeat. Otherwise, the loop terminates.
- It is known as **exit controlled** loop

Example

```
// Java program to illustrate do-while loop
class dowhileloopDemo
{
        public static void main(String args[])
        {
                int x = 21;
                do
                {
                        // The line will be printed even
                        // if the condition is false
                        System.out.println("Value of x:" + x);
                        x++;
                }
                while (x < 20);
        }
}
```

Output: Value of x:21

for loop

```
for (initialization; condition; iteration)
    {
     // body
    }
```

for loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

- Initialization portion sets the value of loop control variable.
- Initialization expression is only executed once.
- Condition must be a Boolean expression. It usually tests the loop control variable against a target value.
- Iteration is an expression that increments or decrements the loop control variable.

The for loop operates as follows.

- When the loop first starts, the initialization portion of the loop is executed.

- Next, condition is evaluated. If this expression is true, then the body of the loop is executed. If it is false, the loop terminates.

- Next, the iteration portion of the loop is executed.

Note: It is also known as **entry controlled loop**

```
class ForTable
        {
                public static void main(String args[])
                 {
                        int n;
                        int x=5;
                        for(n=1; n<=10; n++)
                          {
                           int p = x*n;
                           System.out.println(x+"*"+n +"="+ p);
                          }
                }
        }
```

Output:

```
5*1=5
5*2=10
5*3=15
5*4=20
5*5=25
5*6=30
5*7=35
5*8=40
5*9=45
5*10=50
```

Example of for loop with no body of execution

```
// The body of a loop can be empty.
class Empty3 {
  public static void main(String args[]) {
    int i;
    int sum = 0;

    // sum the numbers through 5
    for(i = 1; i <= 5; sum += i++) ;

    System.out.println("Sum is " + sum);
  }
}
```

No body in this loop!

The output from the program is shown here:

```
Sum is 15
```

What will be the output?

```
class Loop
    {
        public static void main(String args[])
            {
                int i;
                for(i=0; i<5; i++);
                        System.out.println (i++);
             }
      }
```

Declaring loop control variable inside loop

- We can declare the variable inside the initialization portion of the for.

```
for ( int i=0; i<10; i++)
        {
           System.out.println(i);
        }
```

- Note: The scope of this variable i is limited to the for loop and ends with the for statement.

More points....

Interesting for loop variation. Either the initialization or the iteration expression or both may be absent, as in the following example:

```
// Parts of the for loop can be empty.
class ForVar {
public static void main(String args[]) {
int i;
boolean done = false;
i = 0;
for( ; !done; ) {
System.out.println("i is " + i);
if(i == 10) done = true;
i++;
}
}
}
```

Output: Value of I will be printed from 0 to 10

More points…

CODERINDEED

- We can intentionally create an infinite loop (a loop that never terminates) if you leave all three parts of the **for** empty. For example:

```
for( ; ; ) {
// ...
}
```

- Java permits you to include multiple statements in both the initialization and iteration portions of the **for**. Each statement is separated from the next by a comma. Using the comma, the preceding **for** loop can be more efficiently coded, as shown here:

```
// Using the comma.
class Comma {
public static void main(String args[]) {
int a, b;
for(a=1, b=4; a<b; a++, b--) {
System.out.println("a = " + a);
System.out.println("b = " + b);
}
}
}
```

Output:

```
a = 1
b = 4
a = 2
b = 3
```

Q1

OUTPUT??

```
public class First
{
public static void main(String[] args)
{
int i=5;
while(i)
{
System.out.println("Hello");
i--;
}
}
}
```

A. Hello will be printed 5 times
B. Compile time error
C. Runtime error
D. Infinite loop

Q2(Output??)


```
public class First
{
public static void main(String[] args)
{
int x=5,sum=0;
boolean y=true;
while(y)
{
sum=sum+x;
x--;
if(x==3)
y=!y;
}
System.out.println(sum);
}
}
```

A. 9
B. Compile time error
C. Infinite loop
D. 12

Q3(Output??)

```
public class First
{
public static void main(String[] args)
{
int x=5,y=1;
while(--x!=++y);
System.out.println(x+y);
}
}
```

A. Compile time error
B. 6
C. 4
D. 2

Q4(Output??)

```
public class First
{
public static void main(String[] args)
{
int k=1;
for(int i=1,j=2;i>=1 & i<=3;i++,j++)
{
k=k*j;
}
System.out.println(k);
}
}
```

A. 24
B. 12
C. Compile time error
D. Runtime error

Q5

How many times “Hello” will be printed in the following code?

```
public class First
{
public static void main(String[] args)
{
int i=24;
for(;i>1;i>>=2)
{
System.out.println("Hello");
}
}
}
```

A. 2 times
B. 3 times
C. 5 times
D. 4 times