Topic Covered:
The if statement, if-else statement, nested if, else-if structure, Random Class, switch statement, for loop, nested for loops, while Loop and do-while Loop

IF STATEMENT

Following is the syntax of if statement:

```
if(Boolean_expression) {
   // Statements will execute if the Boolean expression is true
}
```

If the Boolean expression evaluates to true then the block of code inside the if statement will be executed. If not, the first set of code after the end of the if statement (after the closing curly brace) will be executed.

IF-ELSE STATEMENT

Following is the syntax of an if...else statement:

```
if(Boolean_expression) {
   // Executes when the Boolean expression is true
}
else {
   // Executes when the Boolean expression is false
}
```

If the boolean expression evaluates to true, then the if block of code will be executed, otherwise else block of code will be executed.

PROGRAM 1: Demonstrating basic if-else statement.

```
import java.util.Scanner;
class PassingGrade
{
  public static void main( String [ ] args )
  {
    Scanner scan = new Scanner( System.in );
    System.out.print( "Enter Marks : " );
    int marks = scan.nextInt( );
    String message;
    if ( marks >= 60 )
      message = "You passed";
    else
      message = "You failed ";
    System.out.println( message ) ;
  }
}
```

NESTED IF AND ELSE-IF STRUCTURE

It is always legal to nest if-else statements which means you can use one if or else if statement inside another if or else if statement.

The syntax for a nested if is as follows:

```
if(Boolean_expression 1) {
   // Executes when the Boolean expression 1 is true
   if(Boolean_expression 2) {
      // Executes when the Boolean expression 2 is true
   }
}
```

You can nest else if...else in the similar way as we have nested if statement:

```
if(Boolean_expression 1)
     {
   // Executes when the Boolean expression 1 is true
     }
   //else Executes when the Boolean expression 1 is false
     else if(Boolean_expression 2)
     {
      // Executes when the Boolean expression 2 is true
     }
     else
     {
      // Executes when the Boolean expression 2 is false
     }
```

PROGRAM 2: Demonstrating if-else-if statements.

```
import java.util.Scanner;
class LetterGrade
{
public static void main( String [] args )
{
Scanner scan = new Scanner( System.in );
char grade;
System.out.print( "Enter your test grade: " );
int score = scan.nextInt( );
if ( score >= 90 )       grade = 'A';
else if ( score >= 80 )  grade = 'B';
else if ( score >= 70 )  grade = 'C';
else if ( score >= 60 )  grade = 'D';
else  grade = 'F';
System.out.println("Your test score "+score+" has "+grade+" Grade" );
}
}
```

PROGRAM 3: Demonstrating if-else-if statements.

```
class IfElse
{
public static void main(String args[])
{

int month = 4; // April
String season;

if(month == 12 || month == 1 || month == 2)
season = "Winter";
else if(month == 3 || month == 4 || month == 5)
season = "Spring";
else if(month == 6 || month == 7 || month == 8)
season = "Summer";
else if(month == 9 || month == 10 || month == 11)
season = "Autumn";
else
season = "Bogus Month";
System.out.println("April is in the " + season + ".");
}
}
```

GENERATING RANDOM NUMBERS WITH THE RANDOM CLASS

The Random class, which is in the java.util package, uses a mathematical formula to generate a sequence of numbers, feeding the formula a seed value, which determines where in that sequence the set of random numbers will begin.

***Random* Class Constructor**	
Random( ) creates a random number generator	
The *nextInt* Method	
Return value	**Method name and argument list**
int	**nextInt(int number)** returns a random integer ranging from 0 up to, but not including, *number* in uniform distribution

PROGRAM 4: Demonstrating if-else-if statements and Random class.

```
import java.util.Random;
import java.util.Scanner;

class GuessANumber
{
public static void main( String [ ] args )
{

Random random = new Random( );
int secretNumber = random.nextInt( 10 ) + 1;

Scanner scan = new Scanner( System.in );

System.out.print( "I'm thinking of a number" + " between 1 and 10. What
is your guess? " );
int guess = scan.nextInt( );

if ( guess < 1 || guess > 10 )
   {
   System.out.println( "Well, if you're not going to try," + " I'm not
playing." );
   }

else
   {
   if ( guess == secretNumber )
      System.out.println( "Hoorah. You win!" );
   else
      {
      System.out.println( "The number was " + secretNumber );
      if ( Math.abs( guess - secretNumber ) > 3 )
         System.out.println( "You missed it by a mile!" );
      else
         System.out.println( "You were close." );
      System.out.println( "Better luck next time." );
      }
   }
}
}
```

SWITCH STATEMENT

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

```
switch(expression) {
   case value :
      // Statements
      break; // optional

   case value :
      // Statements
      break; // optional

   // You can have any number of case statements.
   default : // Optional
      // Statements
}
```

The following rules apply to a switch statement:

- The variable used in a switch statement can only be integers, convertible integers (byte, short, char), strings and enums. Beginning with JDK 7, you can use a string to control a switch statement.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- The value for a case must be the same data type as the variable in the switch and it must be a constant or a literal.
- When the variable being switched on is equal to a case, the statements following that case will execute until a break statement is reached.
- When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- Not every case needs to contain a break. If no break appears, the flow of control will fall through to subsequent cases until a break is reached.
- A switch statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.
- You can use a switch as part of the statement sequence of an outer switch. This is called a nested switch.

PROGRAM 5: Demonstrating simple switch.

```
class SampleSwitch  {
public static void main(String args[])  {
int i = 3;
     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;
     case 3:
     System.out.println("i is three.");
     break;
     default:
     System.out.println("i is greater than 3.");
     }
}
}
```

PROGRAM 6: Demonstrating that in a switch, break statements are optional.

```
class MissingBreak {
public static void main(String args[]) {
int i = 6;
     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");
     }
}
}
```

PROGRAM 7: Demonstrating switch

```
class Switch {
public static void main(String args[]) {
int month = 4;
String season;
switch (month) {
case 12:
case 1:
case 2:
season = "Winter";
break;
case 3:
case 4:
case 5:
season = "Spring";
break;
case 6:
case 7:
case 8:
season = "Summer";
break;
case 9:
case 10:
case 11:
season = "Autumn";
break;
default:
season = "Bogus Month";
}
System.out.println("April is in the " + season + ".");
}
}
```

FOR LOOP AND NESTED FOR LOOPS

A for loop is a repetition control structure that allows you to efficiently write a loop that needs to be executed a specific number of times. A for loop is useful when you know how many times a task is to be repeated.
The syntax of a for loop is:

```
for(initialization; Boolean_expression; update) {
   // Statements
}
```

Here is the flow of control in a for loop:

- The initialization step is executed first, and only once. This step allows you to declare and initialize any loop control variables and this step ends with a semi colon (;).
- Next, the Boolean expression is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop will not be executed and control jumps to the next statement past the for loop.
- After the body of the for loop gets executed, the control jumps back up to the update statement. This statement allows you to update any loop control variables. This statement can be left blank with a semicolon at the end.

- The Boolean expression is now evaluated again. If it is true, the loop executes and the process repeats (body of loop, then update step, then Boolean expression). After the Boolean expression is false, the for loop terminates.

PROGRAM 8: Demonstrate the for loop.

```
class ForTick {
public static void main(String args[]) {
int n;
for(n=10; n>0; n--)
   System.out.println("tick " + n);
}
}
```

PROGRAM 9: Declare a loop control variable inside the for.

```
class ForTick {
public static void main(String args[]) {
// here, n is declared inside of the for loop
for(int n=10; n>0; n--)
   System.out.println("tick " + n);
}
}
```

PROGRAM 10: Using the comma in for loop

```
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);
   }
}
}
```

PROGRAM 11: 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++;
   }
}
}
```

PROGRAM 12: Nested for loop.

```
class Nested {
public static void main(String args[]) {
int i, j;
for(i=0; i<10; i++) {
   for(j=i; j<10; j++)
      System.out.print(".");
   System.out.println();
   }
}
}
```

WHILE LOOP

A while loop statement in Java programming language repeatedly executes a target statement as long as a given condition is true.

The syntax of a while loop is:

```
while(Boolean_expression) {
   // Statements
}
```

PROGRAM 13: Demonstrate while loop

```
import java.util.Scanner;

class MyClass
{
public static void main(String[] args)
{
char ch = 'y';
double number, answer;
Scanner myObj = new Scanner(System.in);

while (ch == 'y')
   {
   System.out.println("Enter a number:");

   number = myObj.nextInt();
   answer = Math.sqrt(number);
   System.out.println("Square root is " + answer);
   System.out.println("Do you want square root of another number?"    );
   System.out.println( "Press y for yes... ");
   ch = myObj.next().charAt(0);
   }
}
}
```

DO-WHILE LOOP

A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Following is the syntax of a do...while loop:

```
do {
   // Statements
}
while(Boolean_expression);
```

PROGRAM 14: Demonstrate do-while loop

```
class DoWhile {
public static void main(String args[]) {
int n = 10;
do {
   System.out.println("tick " + n);
   n--;
   } while(n > 0);
}
}
```