UNIT - I

INTRODUCTION TO JAVA

JAVA HISTORY:

Java is a general-purpose, object-oriented programming language developed by Sun Microsystems of USA in 1991.Originally called Oak by James Gosling (one of the inventor of the language). Java was invented for the development of software for consumer electronic devices like TVs, testers, etc. The main aim had to make java simple, portable and reliable.

Year	Progress
1990	Sun decided to developed software that could be used for electronic devices. And the project called as Green Project head by James Gosling.
1991	Announcement of a new language named “Oak”
1992	The team verified the application of their new language to manage a list of home appliances using a hand held device.
1993	The World Wide Web appeared on the Internet and transformed the text-based interface to a graphical rich environment.
1994	The team developed a new Web browser called “Hot Java” to locate and run Applets.
1995	Oak was renamed to Java, as it did not survive “legal” registration. Many companies such as Netscape and Microsoft announced their support for Java.
1996	Java language is now famous for Internet programming as well as a general purpose OO language.
1997	Sun releases Java Development Kit(JDK 1.1)
1998	Sun releases Software Development Kit (SDK 1.2)
1999	Sun releases Java 2 platform Standard Edition (J2SE) and Enterprise Edition(J2EE).
2000	J2SE with SDK 1.3 was released.
2002	J2SE with SDK 1.4 was released.
2004	J2SE with JDK 5.0 was released.

JAVA FEATURES: **[5 or 10 Marks]**

As we know that the Java is an object oriented programming language developed by Sun Microsystems of USA in 1991. Java is first programming language which is not attached with any particular hardware or operating system. Program developed in Java can be executed anywhere and on any system.

Features of Java are as follows:

1. Compiled and Interpreted
2. Platform Independent and portable
3. Object- oriented
4. Robust and secure
5. Distributed
6. Familiar, simple and small
7. Multithreaded and Interactive
8. High performance
9. Dynamic and Extensible

1. Compiled and Interpreted - Basically a computer language is either compiled or interpreted. Java comes together both these approach thus making Java a two-stage system.

Java compiler translates Java code to Byte code instructions and Java Interpreter generate machine code that can be directly executed by machine that is running the Java program. **[2 Marks]**

2. Platform Independent and portable - Java supports the feature portability. Java programs can be easily moved from one computer system to another and anywhere. Changes and upgrades in operating systems, processors and system resources will not force any alteration in Java programs. This is reason why Java has become a trendy language for programming on Internet which interconnects different kind of systems worldwide. Java certifies portability in two ways. First way is, Java compiler generates the byte code and that can be executed on any machine. Second way is, size of primitive data types are machine independent. **[2 Marks]**

3. Object- oriented - Java is truly object-oriented language. In Java, almost everything is an Object. All program code and data exist in objects and classes. Java comes with an extensive set of classes; organize in packages that can be used in program by Inheritance. The object model in Java is trouble-free and easy to enlarge.

4. Robust and secure - Java is a most strong language which provides many securities to make certain reliable code. It is design as garbage – collected language, which helps the programmers virtually from all memory management problems. Java also includes the concept of exception handling, which detain serious errors and reduces all kind of threat of crashing the system.

Security is an important feature of Java and this is the strong reason that programmer use this language for programming on Internet.

The absence of pointers in Java ensures that programs cannot get right of entry to memory location without proper approval.

5. Distributed - Java is called as Distributed language for construct applications on networks which can contribute both data and programs. Java applications can open and access remote objects on Internet easily. That means multiple programmers at multiple remote locations to work together on single task.

6. Simple and small - Java is very small and simple language. Java does not use pointer and header files, goto statements, etc. It eliminates operator overloading and multiple inheritance.

7. Multithreaded - and Interactive Multithreaded means managing multiple tasks simultaneously. Java maintains multithreaded programs. That means we need not wait for the application to complete one task before starting next task. This feature is helpful for graphic applications.

8. High performance - Java performance is very extraordinary for an interpreted language, majorly due to the use of intermediate bytecode. Java architecture is also designed to reduce overheads during runtime. The incorporation of multithreading improves the execution speed of program.

9. Dynamic and Extensible - Java is also dynamic language. Java is capable of dynamically linking in new class, libraries, methods and objects. Java can also establish the type of class through the query building it possible to either dynamically link or abort the program, depending on the reply.

DIFFERENCE BETWEEN C AND JAVA [5 Marks]

C	Java
Procedure Oriented Programming Language	True Object oriented Programming Language
C was developed by **Dennis Ritchie**..	Java was developed by **James Gosling** and his team. Development began in 1991.
C is a **compiled language**.	Java is both compiled and interpreted.
C programs are **platform dependent**. They need to be compiled for a particular platform.	Java programs are platform independent.
C supports global variables	Java does not supports global variables
C fully support **pointers**.	Java has restricted support for pointers. Pointers are supported internally you cannot writer pointer programs.
C supports **structures and union**.	Java does not support structures and union.
C supports Preprocessors header files	Java does not C supports Preprocessors header files
C supports **goto** statement (however the use of goto is discouraged as not considered a good practice)	Java does not support goto statement (although goto is a reserved keyword in Java)
C++ provides support both for **call by value** and **call by reference**.	Java supports only call by value.

DIFFERENCE BETWEEN C++ AND JAVA [5 Marks]

C++	Java
C++ was developed by **Bjarne Stroustrup**. Development began in 1979.	Java was developed by **James Gosling** and his team. Development began in 1991.
C++ is a **compiled language**.	Java is both compiled and interpreted.
C++ programs are **platform dependent**. They need to be compiled for a particular platform.	Java programs are platform independent.
C++ does support **operator overloading**. Function overloading is also available.	Java does not support operator overloading. However, function overloading is possible.
C++ fully support **pointers**.	Java has restricted support for pointers. Pointers are supported internally you can not writer pointer programs.
C++ supports **structures and union**.	Java does not support structures and union.
C++ supports manual **object management** through new and delete keywords**.**	Java relies on automatic garbage collection. It does not support destructors the way C++ does.
C++ supports **goto** statement (however the use of goto is discouraged as not considered a good practice)	Java does not support goto statement (although goto is a reserved keyword in Java)
C++ supports **multiple inheritance**.	Java does not really support multiple inheritance. But similar results can be achieved through the use of interfaces.
C++ provides support both for **call by value** and **call by reference**.	Java supports only call by value.

C++ has no support for the **unsigned right shift** operator (>>>).	Java supports the unsigned right shift >>> operator.

JAVA ENVIRONMENT: [5 Marks]

Java environment includes a number of development tools, classes and methods. The development tools are part of the system known as Java Development Kit (JDK) and the classes and methods are part of the Java Standard Library (JSL), also known as the Application Programming Interface (API).

Java Development kit (JDK) – The JDK comes with a set of tools that are used for developing and running Java program. It includes:

1. Appletviewer(It is used for viewing the applet)
2. Javac(It is a Java Compiler)
3. Java(It is a java interpreter)
4. Javap(Java diassembler,which convert byte code into program description)
5. Javah(It is for java C header files)
6. Javadoc(It is for creating HTML document)
7. Jdb(It is Java debugger)

For compiling and running the program we have to use following commands: [2 Marks]

a) javac (Java compiler) In java, we can use any text editor for writing program and then save that program with ".java" extension. Java compiler convert the source code or program in bytecode and interpreter convert ".java" file in ".class" file.

Syntax: C:\javac filename.java

If my filename is "abc.java" then the syntax will be

C:\javac abc.java

b) java(Java Interpreter) As we learn that, we can use any text editor for writing program and then save that program with ".java" extension. Java compiler convert the source code or program in bytecode and interpreter convert ".java" file in ".class" file.

Syntax: C:\java filename

If my filename is abc.java then the syntax will be

C:\java abc

SIMPLE JAVA PROGRAM: **[5 Marks]**

```
class Demo
{
        public static void main(String args[])
        {
                System.out.println (“Welcome to JAVA”);
        }
}
```

Class declaration – The first line class Demo declares a class. Where class is a keyword and Demo is a java identifier that specifies the name of the class to be defined.

Opening braces { - Every class definition in java begins with an opening brace and ends with matching closing brace.

The main line
public static void main(String args[])
it defines a method named main this is similar to main function in C or C++. Every java application program must include the main method. This is a starting point for the interpreter to begin the execution of the program.

public – The keyword public is an access specifier that declares the main method as unprotected.

static – The main method must always be declared as static since the interpreter uses this main method before many objects are created.

void – The type modifier void states that the main method does not return any value.

main - The main indicates that the startup of our program.

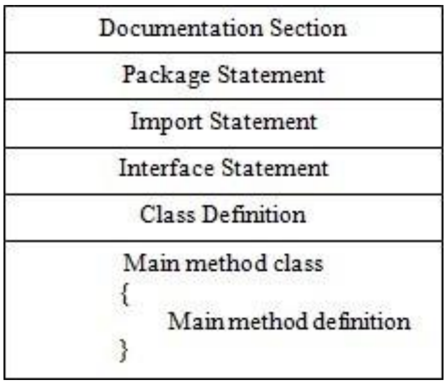
String args[] – here String args[] declares a parameter named args which contains an array of object of the class type String.

The output line -- System.out.println (“Welcome to JAVA”);

This is similar to printf function is C or cout construct of C++. Println method is the member of the out object which is static data member of System class.

JAVA PROGRAM STRUCTURE

Describe the structure of typical java program. **[5 Marks]**



Documentation Section
Package Statement
Import Statement
Interface Statement
Class Definition

Main method class
{
Main method definition
}

1. Documentaion Statement

Documentation section is nothing but the comment lines giving the name of the program, the author and other details.

2. Package Statement

The first statement allowed in java file is package statement. This statement declares the package name and informs the compiler that the classes defined here belongs to this package.

3. Import statement

The next thing after a package statement may be a number of import statements. This is similar to the #include statement in C.

4. Interface Statement

An interface is like a class but includes a group of methods declarations.

5. Class Definition

A java program may contain multiple class definitions. Classes are primary and essential elements of java program.

6. main method class

Every java stand alone program requires a main method as its starting point, this class is the essential part of java program.

TOKENS IN JAVA: **[5 Marks]**

There are five types of token as follows:

1. Literals
2. Identifiers
3. Keywords
4. Operators
5. Separators

1 Literals: Literals in Java are a sequence of characters (digits, letters and other characters) that characterize constant values to be stored in variables. Java language specifies five major types of literals are as follows:

1. Integer literals 2. Floating point literals 3. Character literals 4. String literals 5. Boolean literals

2 Identifiers: Identifiers are programmer-created tokens. They are used for naming classes, methods, variables, objects, labels, packages and interfaces in a program.

Java identifiers follow the following rules:
1. They can have alphabets, digits, and the underscore and dollar sign characters.

2. They must not start with a digit.

3. Uppercase and lowercase letters are individual.

4. They can be of any length.

5. Identifier must be meaningful, easily understandable and descriptive.

For example: Private and local variables like "length". Name of public methods and instance variables begin with lowercase letter like "addition"

3 Keywords: Keywords are important part of Java. Java language has reserved 50 words as keywords. Keywords have specific meaning in Java. We cannot use them as variable, classes and method. Following table shows keywords.

abstract char catch boolean default finally do implements if long throw private package static break double this volatile import protected class throws byte else float final public transient native instanceof case extends int null const new return try for switch interface void while synchronized short continue goto super assert const

4 Operator: Java carries a broad range of operators. An operator is symbols that specify operation to be performed may be certain mathematical and logical operation. Operators are used in programs to operate data and variables. They frequently form a part of mathematical or logical expressions.

5 Separator: Separators are symbols. It shows the separated code.they describe function of our code. **[2 Marks]**

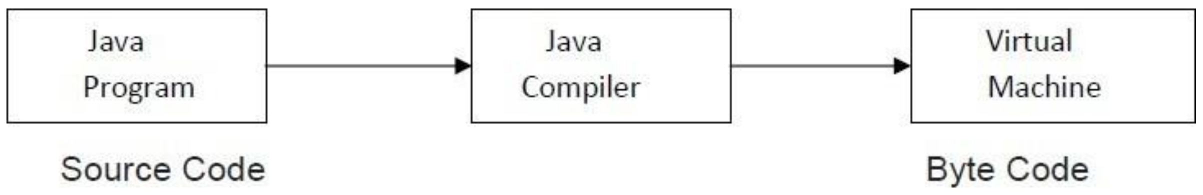
Name	Use
()	Parameter in method definition, containing statements for conditions,etc.
{}	It is used for define a code for method and classes
[]	[] It is used for declaration of array
;	It is used to show the separate statement
,	;, It is used to show the separation in identifier in variable declarartion
.	It is used to show the separate package name from sub- packages and classes, separate variable and method from reference variable.

JVM (JAVA VIRTUAL MACHINE) [2 Marks]

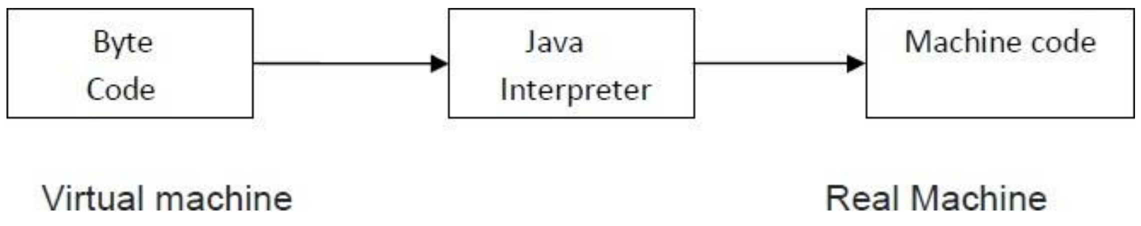
JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

As we know that all programming language compilers convert the source code to machine code. Same job done by Java Compiler to run a Java program, but the difference is that Java compiler onvert the source code into Intermediate code is called as bytecode. This machine is called the Java Virtual machine and it exits only inside the computer memory.

Following figure shows the process of compilation.



The Virtual machine code is not machine specific. The machine specific code is generated. By Java interpreter by acting as an intermediary between the virtual machine and real machines shown below



JAVA COMMAND LINE ARGUMENTS [5 Marks]

- The java command-line argument is an argument i.e. passed at the time of running the java program.
- The arguments passed from the console can be received in the java program and it can be used as an input.

In this example, we are receiving only one argument and printing it. To run this java program, you must pass at least one argument from the command prompt.

```
class CommandLineExample
{
        public static void main(String args[])
        {
                System.out.println("Your first argument is: "+args[0]);
         }
}
```

```
compile by > javac CommandLineExample.java
run by > java CommandLineExample BCA
```

Output: Your first argument is: BCA

CONSTANT: Constant means fixed value which is not change at the time of execution of program. In Java The types of constant as follows:

1. Numeric Constants
2. Integer constant
3. Real constant
4. Character Constants
5. Character constant
6. String constant

1 Integer Constant: An Integer constant refers to a series of digits. There are three types of integer as follows: a) Decimal integer Embedded spaces, commas and characters are not allowed in between digits. For example: 23 411 7,00,000 17.33

b) Octal integer It allows us any sequence of numbers or digits from 0 to 7 with leading 0 and it is called as Octal integer.

For example:
011
00
0425
c) Hexadecimal integer It allows the sequence which is preceded by 0X or 0x and it also allows alphabets from 'A' to 'F' or 'a' to 'f' ('A' to 'F' stands for the numbers '10' to '15') it is called as Hexadecimal integer. For example: 0x7 00X 0A2B
2 Real Constant It allows us fractional data and it is also called as floating point constant. It is used for percentage, height and so on. For example: 0.0234 0.777 -1.23

3 Character Constant It allows us single character within pair of single count. For example: 'A' '7' '\'

4 String Constant It allows us the series of characters within pair of double coute. For example: "WELCOME" "END OF PROGRAM" "BYE …BYE" "A"

5 Symbolic constant: In Java program, there are many things which is requires repeatedly and if we want to make changes then we have to make these changes in whole program where this variable is used. For this purpose, Java provides 'final' keyword to declare the value of variable as follows: Syntax: final type Symbolic name=value;

For example: If I want to declare the value of 'PI' then:

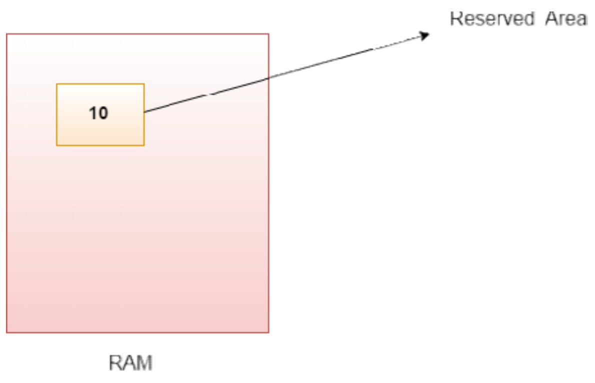
final float PI=3.1459

the condition is, Symbolic name will be in capital letter(it shows the difference between normal variable and symbolic name) and do not declare in method.

6 Backslash character constant: Java support some special character constant which are given in following table. Constant Importance '\b' Back space '\t' Tab '\n' New line '\\' Backslash '\" Single quote '\"' Double quote

VARIABLES **[5 Marks]**

Variable is name of reserved area allocated in memory. In other words, it is a name of memory location. It is a combination of "vary + able" that means its value can be changed.

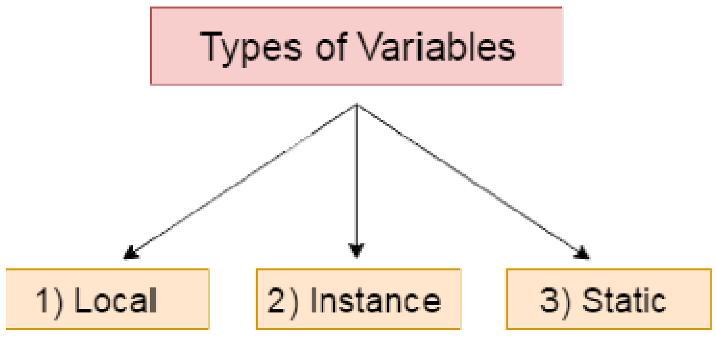


1. int data=50;//Here data is variable

Types of Variable

There are three types of variables in java:

- local variable
- instance variable
- static variable



1) Local Variable

A variable which is declared inside the method is called local variable.

2) Instance Variable

A variable which is declared inside the class but outside the method, is called instance variable. It is not declared as static.

3) Static variable

A variable that is declared as static is called static variable. It cannot be local.

Example to understand the types of variables in java

```
class A
{
        int data=50;//instance variable
        static int m=100;//static variable
        void method()
        {
                int n=90;//local variable
        }
}//end of class
```

DATA TYPES IN JAVA **[5 Marks]**

There are two data types available in Java –

- Primitive Data Types
- Reference/Object Data Types

Primitive Data Types
There are eight primitive datatypes supported by Java. Primitive datatypes are predefined by the language and named by a keyword. Let us now look into the eight primitive data types in detail.

byte

- Byte data type is an 8-bit signed two's complement integer
- Minimum value is -128 (-2^7)
- Maximum value is 127 (inclusive)(2^7 -1)
- Default value is 0
- Byte data type is used to save space in large arrays, mainly in place of integers, since a byte is four times smaller than an integer.
- Example: byte a = 100, byte b = -50

short

- Short data type is a 16-bit signed two's complement integer
- Minimum value is -32,768 (-2^15)
- Maximum value is 32,767 (inclusive) (2^15 -1)
- Short data type can also be used to save memory as byte data type. A short is 2 times smaller than an integer
- Default value is 0.
- Example: short s = 10000, short r = -20000

int

- Int data type is a 32-bit signed two's complement integer.
- Minimum value is - 2,147,483,648 (-2^31)
- Maximum value is 2,147,483,647(inclusive) (2^31 -1)
- Integer is generally used as the default data type for integral values unless there is a concern about memory.
- The default value is 0
- Example: int a = 100000, int b = -200000

long

- Long data type is a 64-bit signed two's complement integer
- Minimum value is -9,223,372,036,854,775,808(-2^63)
- Maximum value is 9,223,372,036,854,775,807 (inclusive)(2^63 -1)
- This type is used when a wider range than int is needed
- Default value is 0L
- Example: long a = 100000L, long b = -200000L

float

- Float data type is a single-precision 32-bit IEEE 754 floating point
- Float is mainly used to save memory in large arrays of floating point numbers
- Default value is 0.0f
- Float data type is never used for precise values such as currency
- Example: float f1 = 234.5f

double

- double data type is a double-precision 64-bit IEEE 754 floating point
- This data type is generally used as the default data type for decimal values, generally the default choice
- Double data type should never be used for precise values such as currency
- Default value is 0.0d
- Example: double d1 = 123.4

boolean

- boolean data type represents one bit of information
- There are only two possible values: true and false
- This data type is used for simple flags that track true/false conditions
- Default value is false
- Example: boolean one = true

char

- char data type is a single 16-bit Unicode character
- Minimum value is '\u0000' (or 0)
- Maximum value is '\uffff' (or 65,535 inclusive)
- Char data type is used to store any character
- Example: char letterA = 'A'

TYPE CASTING **[2 or 5 Marks]**

Type casting- Converting one data type to another data type is called type casting.

There are two types of type casting

- Implicit type casting
- Explicit type casting

Implicit type casting- an implicit type casting will take place if the following two conditions are met

1. The two types are compatible
2. The destination type is larger than the source type.

Explicit type casting- To create a conversation between two incompatible types is called explicit type casting.

Example

```
class TypeCasting
{
        public static void main(String args[])
        {
                byte b=10;
                int i=30;
                float f=15.5f;
                double d=12.3;
                System.out.println("The values before type casting");
                System.out.println("");
                System.out.println("byte b = "+b);
                System.out.println("int i    = "+i);
                System.out.println("float f = "+f);
                System.out.println("double d = "+d);
                System.out.println("");
                System.out.println("The values after explicit type casting");
                System.out.println("");
                byte b1=(byte)i;
                System.out.println("byte b1 = "+b1);
                int   i1=(int)d;
                System.out.println("int i1 = "+i1);
                System.out.println("");
                System.out.println("The values after implicit typepromotion");
                System.out.println("");
                float f1=b+i;
                System.out.println("float f1 = "+f1);
                double d1=f+i;
                System.out.println("double d1 = "+d1);
        }
}
```

OPERATORS IN JAVA **[5 or 10 Marks]**

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups:

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Increment or Decrement operators
- Conditional Operators
- Assignment Operators
- Special Operators

TheArithmeticOperators:

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators:

Assume integer variable A holds 10 and variable B holds 20, then:

Show Examples

SL.NO	Operator and Example
1	**+ (Addition)** Adds values on either side of the operator **Example:** A + B will give 30
2	**- (Subtraction)** Subtracts right hand operand from left hand operand **Example:** A - B will give -10
3	*** (Multiplication)** Multiplies values on either side of the operator **Example:** A * B will give 200
4	**/ (Division)** Divides left hand operand by right hand operand **Example:** B / A will give 2
5	**% (Modulus)** Divides left hand operand by right hand operand and returns remainder **Example:** B % A will give 0

TheRelationalOperators:

There are following relational operators supported by Java language

Assume variable A holds 10 and variable B holds 20, then:

SR.NO	Operator and Description
1	**== (equal to)** Checks if the values of two operands are equal or not, if yes then condition becomes true. **Example:** (A == B) is not true.
2	**!= (not equal to)** Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. **Example:** (A != B) is true.
3	**> (greater than)** Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. **Example:** (A > B) is not true.
4	**< (less than)** Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. **Example:** (A < B) is true.
5	**>= (greater than or equal to)** Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. **Example** (A >= B) is not true.
6	**<= (less than or equal to)** Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. **example**(A <= B) is true.

TheBitwiseOperators: **[2 Marks]**

Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.
Bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60; and b = 13; now in binary format they will be as follows:
a = 0011 1100
b = 0000 1101
----
a&b = 0000 1100
a|b = 0011 1101
a^b = 0011 0001
~a = 1100 0011
The following table lists the bitwise operators:
Assume integer variable A holds 60 and variable B holds 13 then:

SR.NO	Operator and Description
1	**& (bitwise and)** Binary AND Operator copies a bit to the result if it exists in both operands. **Example:** (A & B) will give 12 which is 0000 1100
2	**| (bitwise or)** Binary OR Operator copies a bit if it exists in either operand. **Example:** (A | B) will give 61 which is 0011 1101
3	**^ (bitwise XOR)** Binary XOR Operator copies the bit if it is set in one operand but not both. **Example:** (A ^ B) will give 49 which is 0011 0001
4	**~ (bitwise compliment)** Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. **Example:** (~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
5	**<< (left shift)** Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand **Example:** A << 2 will give 240 which is 1111 0000
6	**>> (right shift)** Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. **Example:** A >> 2 will give 15 which is 1111

TheLogicalOperators:

The following table lists the logical operators:

Assume Boolean variables A holds true and variable B holds false, then:

Show Examples

Operator	Description
1	**&& (logical and)** Called Logical AND operator. If both the operands are non-zero, then the condition becomes true. **Example** (A && B) is false.
2	**|| (logical or)** Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true. **Example** (A || B) is true.
3	**! (logical not)** Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false. **Example** !(A && B) is true.

IncrementorDecrementoperator

1	**++ (Increment)** Increases the value of operand by 1 **Example:** B++ gives 21
2	**-- (Decrement)** Decreases the value of operand by 1 **Example:** B-- gives 19

TheAssignmentOperators:

There are following assignment operators supported by Java language:

Show Examples

SR.NO	Operator and Description
1	**=** Simple assignment operator, Assigns values from right side operands to left side operand. **Example:** C = A + B will assign value of A + B into C
2	**+=** Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand. **Example:** C += A is equivalent to C = C + A
3	**-=** Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand. **Example:**C -= A is equivalent to C = C – A
4	***=** Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand. **Example:** C *= A is equivalent to C = C * A
5	**/=** Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand **Example**C /= A is equivalent to C = C / A

ConditionalOperator(?:) **[2 Marks]**

Conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide which value should be assigned to the variable. The operator is written as:

```
variable x = (expression) ? value if true : value if false
```

Following is the example:

```
public class Test
{
  public static void main(String args[])
  {
    int a, b;
    a = 10;
    b = (a = = 1) ? 20: 30;
    System.out.println( "Value of b is : " + b );
    b = (a == 10) ? 20: 30;
    System.out.println( "Value of b is : " + b );
  }
}
```

This would produce the following result –

```
Value of b is : 30

Value of b is : 20
```

SpecialOperator

instanceofOperator:

This operator is used only for object reference variables. The operator checks whether the object is of a particular type (class type or interface type). instanceof operator is written as:

```
( Object reference variable ) instanceof (class/interface type)
```

If the object referred by the variable on the left side of the operator passes the IS-A check for the class/interface type on the right side, then the result will be true.

Following is the example:

```
public class Test {

   public static void main(String args[]){
      String name = "James";
      // following will return true since name is type of String
      boolean result = name instanceof String;
      System.out.println( result );
   }
}
```

This would produce the following result:

```
true
```

This operator will still return true if the object being compared is the assignment compatible with the type on the right. Following is one more example:

```
class Vehicle {}

public class Car  extends  Vehicle {
   public static void main(String args[]){
      Vehicle a = new Car();
      boolean result = a instanceof Car;
      System.out.println( result );
   }
}
```

This would produce the following result:

```
true
```

BRANCHING STATEMENTS **[5 or 10 Marks]**

When a program breaks the sequential flow and jumps to another part of the code, it is called branching.

There are two types of decision making statements in Java. They are:

• if statements

• switch statements

The Java if statement is used to test the condition. It checks boolean condition: true or false. There are various types of if statement in java.

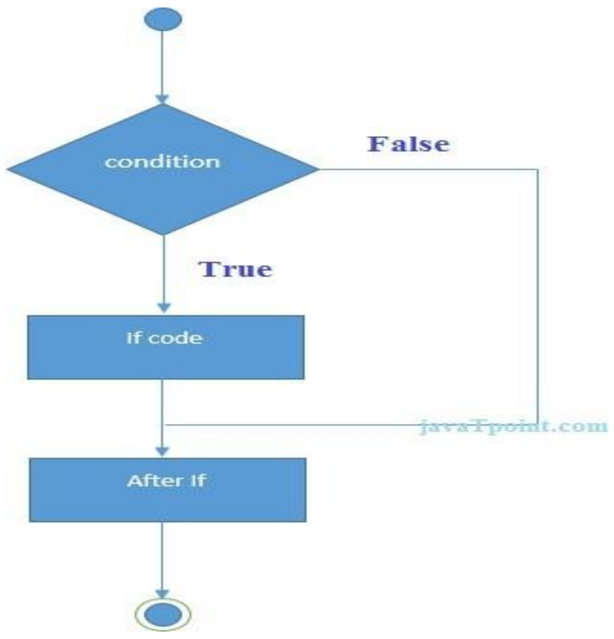
- if statement
- if-else statement
- nested if statement
- if-else-if ladder

The if Statement:

An if statement consists of a Boolean expression followed by one or more statements.

The syntax of an if statement is:

```
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.

Example:

```
public class Test
{
        public static void main(String args[])
        {
                int x =10;
                if( x <20)
                {
                        System.out.print("This is if statement");
                }
        }
}
```

This would produce the following result:

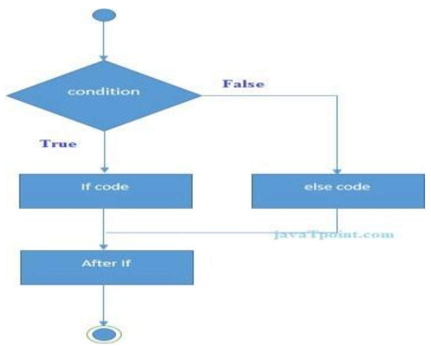
```
This is if statement
```

The if...else Statement:

An if statement can be followed by an optional else statement, which executes when the Boolean expression is false.

Syntax:

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



Example:

```
public class Test
{
        public static void main(String args[])
        {
                int x =30;
                if(x <20)
                {
                        System.out.print("This is if statement");
                }
                else
                {
                        System.out.print("This is else statement");
                }
        }
}
```

This would produce the following result:

```
This is else statement
```

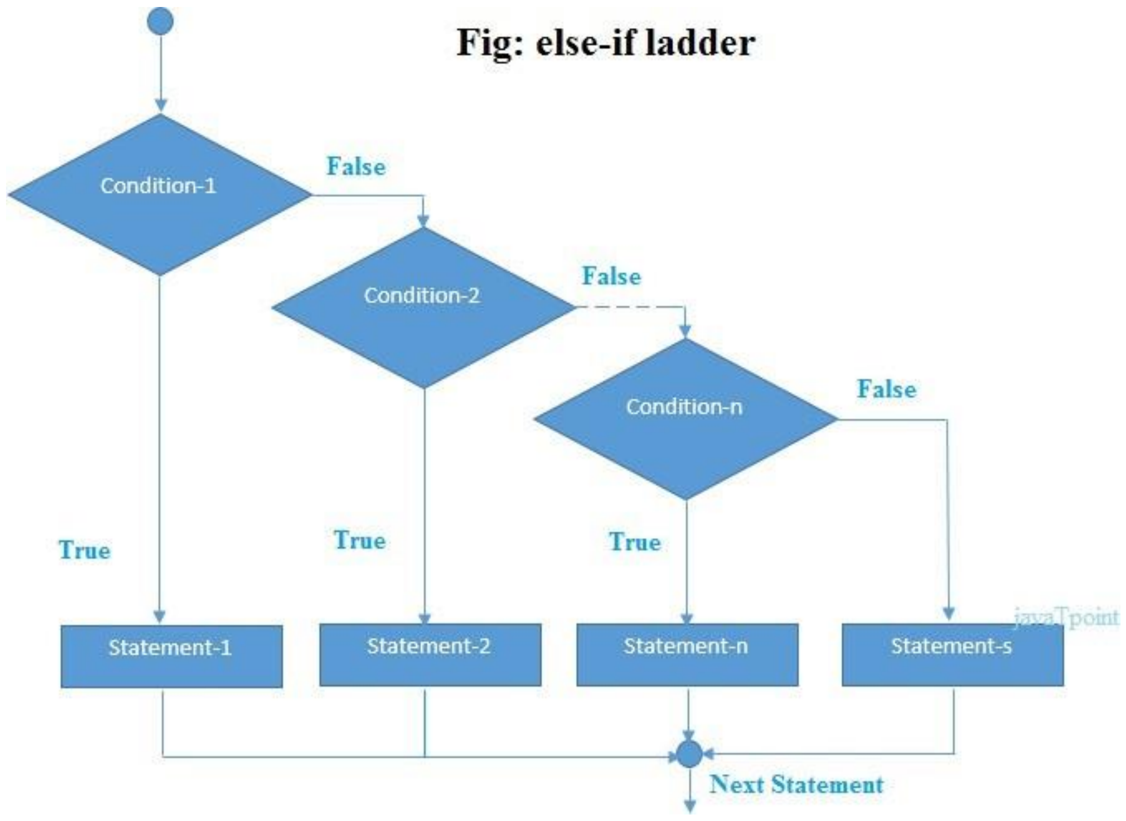
Java IF-else-if ladder Statement

The if-else-if ladder statement executes one condition from multiple statements.

Syntax:

```
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
}
...
els
e
{   //code to be executed if all the conditions are false

}
```



Example:

```
public class IfElseIfExample
{
        public static void main(String[] args)
        {
                    int
                     marks=65;
                     if(marks<50
                     )
                    {
                                System.out.println("fail");
                    }
                    else if(marks>=50 && marks<60)
                    {
                                System.out.println("D grade");
                    }
                    else if(marks>=60 && marks<70)
                    {
                                System.out.println("C grade");
                    }
                    else if(marks>=70 && marks<80)
                    {
                                System.out.println("B grade");
                    }
```

```
                  else if(marks>=80 && marks<90)
                  {
                          System.out.println("A grade");
                  }
                  else if(marks>=90 && marks<100)
                  {
                          System.out.println("A+  grade");
                  }
                  els
                  e
                  {         System.out.println("Invalid!");

                   }
          }
    }
```

Output:
C grade

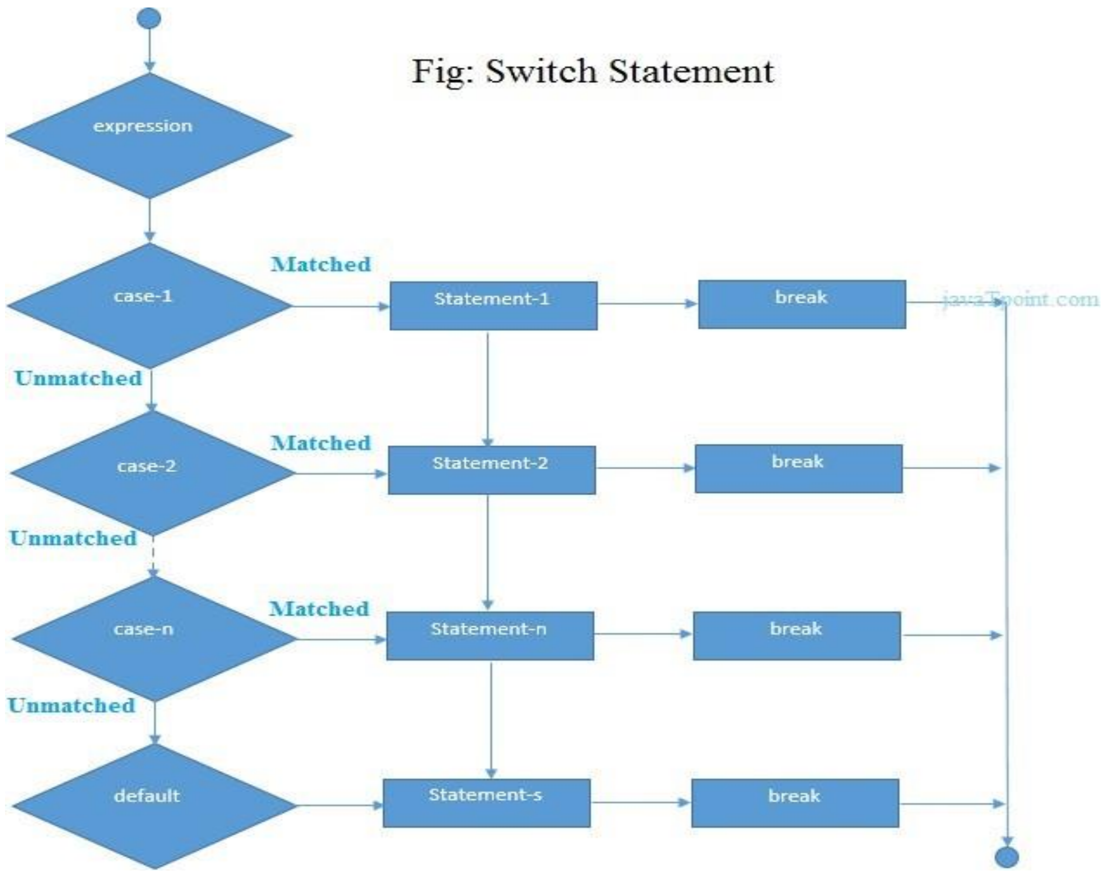
The switch Statement: **[5 Marks]**

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.

Syntax:

The syntax of enhanced for loop is:

```
switch(expression)
{
        case value :
                //Statements
                break;//optional
        case value :
                //Statements
                break;//optional
                //You can have any number of case statements.
        default://Optional
        //Statements
}
```



Example:

```
public class Test
{
        public static void main(String args[])
        {
                char grade = args[0].charAt(0);
                switch(grade)
                {
                        case 'A': System.out.println("Excellent!");
                        break;
                        case'B': System.out.println("Well done");
                        break;
                        case'C': System.out.println("You passed");
                        case'D': System.out.println("Better try again");
                        break;
                        default: System.out.println("Invalid grade");
                }
                System.out.println("Your grade is "+ grade);
        }
}
```

LOOPING STATEMENTS **[5 Marks]**

The iteration statements allow a set of instructions to be performed repeatedly until a certain condition is fulfilled. The iteration statements are also called loops or looping statements. Java provides three kinds of loops: while loop, do-while loop, and for loop.

A Computer is used for performing many Repetitive types of tasks The Process of Repeatedly performing tasks is known as looping

In The loop generally there are three basic operations are performed

1) Initialization
2) Condition check
3) Increment

There are the three types of loops in the java

1) while
2) do-while
3) for

all these are used for performing the repetitive tasks until the given condition is not true.

<u>While</u>

While Loop is Known as Entry Controlled Loop because in The while loop first we initialize the value of variable or Starting point of Execution and then we check the condition and if the condition is true then it will execute the statements and then after it increments or decrements the value of a variable. But in the Will Condition is false then it will never Executes the Statement

Syntax:

```
while(condition)
{
//code to be executed
}
```

flowchart of java while loop

Example:

```
public class WhileExample
 {
        public static void main(String[] args)
         {
                int i=1;
                while(i<=10)
               {
                        System.out.println(i);
                         i++;
                }
        }
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

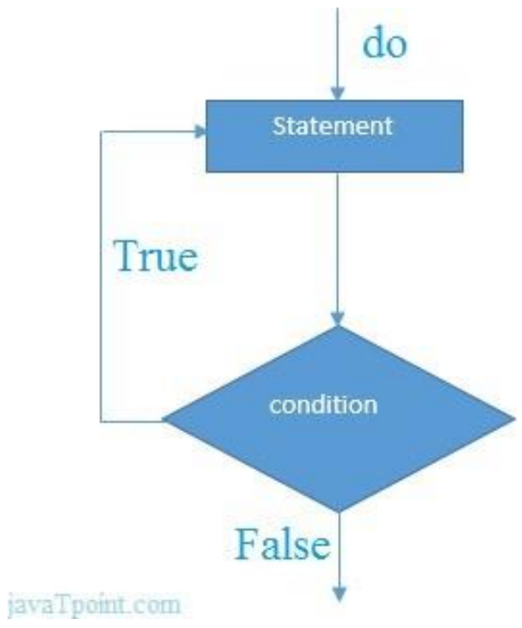
do while

This is Also Called as Exit Controlled Loop we know that in The while loop the condition is check before the execution of the program but if the condition is not true then it will not execute the statements so for this purpose we use the do while loop in this first it executes the statements and then it increments the value of a variable and then last it checks the condition So in this either the condition is true or not it Execute the statement at least one time.

Syntax:

```
Do
{
//code to be executed
}while(condition);
```

flowchart of do while loop in java



Example:

```
public class DoWhileExample
{
            public static void main(String[] args)
             {
                        int i=1;
                        do
                       {
                                   System.out.println(i);
                                    i++;
                       }while(i<=10);
} }
```

Output:

```
1
2
3
4
5
6
7-
8
9
10
```

DIFFERENCE BETWEEN WHILE AND DO WHILE LOOP **[2 Marks]**

While loop	**Do while loop**
It is a looping construct that will execute only if the test condition is true	It is a looping construct that will execute atleast once if the test condition is false
It is entry controlled loop	It is exit controlled loop
It is generally used for implementing common looping situations	It typically used for implementing menu based programs where menu is required to be printed at least once

Sysntax
While(condtion)
{
}
	Syntax
	Do
	{
	}while(condition);

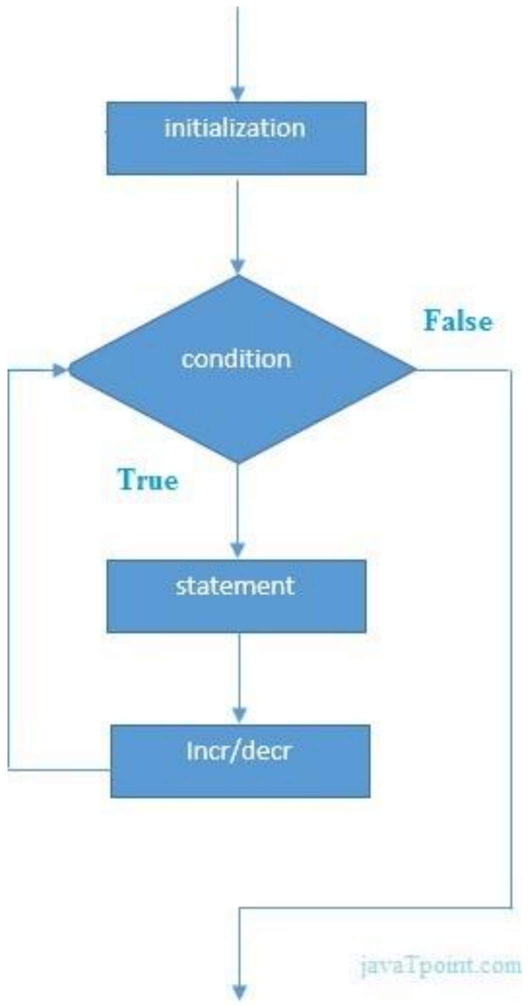
For Loop

In This loop all the basic operations like initialization ,condition checking and incrementing or decrementing all these are performed in only one line. this is similar to the while loop for performing its execution but only different in its syntax.

Syntax:

```
for(initialization;condition;expression)
{
        //code to be executed
}
```

Flowchart



Example

```
class ForLoop
{
        public static void main(String args[])
        {
                int i;

                for(i =1; i <=10; i++)
                {
                        System.out.print(" " +i);
```

```
                    }
            }
    }
```

Output : 1 2 3 4 5 6 7 8 9 10

JUMP STATEMENTS

Explain jump statement used in Java. **[5 Marks]**

Java supports three jump statements: break, continue, and return. These statements transfer control to another part of your program.

1. break.
2. continue.
3. return.

1 The break statement

- This statement is used to jump out of a loop.
- On encountering a break statement within a loop, the execution continues with the next statement outside the loop.
- The remaining statements which are after the break and within the loop are skipped.
- Break statement can also be used with the label of a statement.
- A statement can be labeled as follows.
  statementName : SomeJavaStatement

- When we use break statement along with label as
  break statementName;

An example of break statement

```
class break1
{
    public static void main(String args[])
    {
        int i = 1;
        while (i<=10)
        {
            System.out.println("\n" + i);
            i++;
            if (i==5)
            {
                break;
            }
        }   }}
```

Output :

```
1
2
3
4
```

An example of break to a label

```
class break3
{
        public static void main (String args[])
        {
                boolean t=true;
                a:
                {
                        b:
                        {
                                c:
                                {
                                        System.out.println("Before the break");
                                        if(t)
                                                break b;
                                        System.out.println("This will not execute");
                                }
                                System.out.println("This will not execute");
                        }
                        System.out.println("This is after b");
                }
        }
}
```

Output :

```
Before the break
This is after b
```

2 Continue statement

- This statement is used only within looping statements.
- When the continue statement is encountered, the next iteration starts.
- The remaining statements in the loop are skipped. The execution starts from the top of loop again.

The program below shows the use of continue statement.

```
class continue1
{
        public static void main(String args[])
        {
                for (int i=1; i<1=0; i++)
                {
                        if (i%2 == 0)
                                continue;

                        System.out.println("\n" + i);
                }
        }
}
```

Output :

```
1
3
5
7
9
3
```

3. The return statement

- The last control statement is return. The return statement is used to explicitly return from a method.
- That is, it causes program control to transfer back to the caller of the method.
- The return statement immediately terminates the method in which it is executed.

The program below shows the use of return statement.

```
class Return1
{
      public static void main(String args[])
     {
          boolean t = true;
          System.out.println("Before the return.");
          if(t)
             return;      // return to caller
         System.out.println("This won't execute.");
    }
}
```

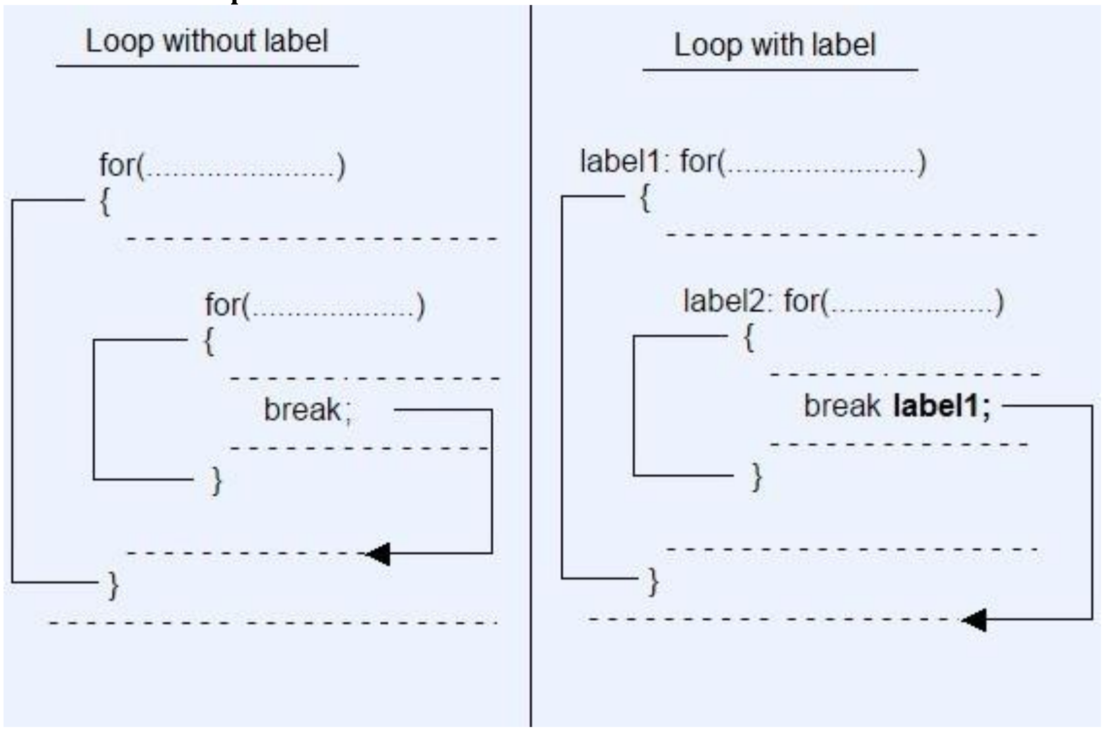
Output :

```
Before the return.
```

LABELLED LOOPS **[5 Marks]**

According to nested loop, if we put break statement in inner loop, compiler will jump out from inner loop and continue the outer loop again. What if we need to jump out from the outer loop using break statement given inside inner loop? The answer is, we should define **lable** along with colon(:) sign before loop.

Syntax of Labelled loop


Loop without label
for(.......................)
{
for(....................)
{
break;
}
}
Loop with label
label1: for(.......................)
{
label2: for(....................)
{
break label1;
}
}

Example without labelled loop

```
        //WithoutLabelledLoop.java

  class WithoutLabelledLoop
  {
     public static void main(String args[])
     {
        int i,j;

        for(i=1;i<=10;i++)
        {
           System.out.println();

           for(j=1;j<=10;j++)
           {
              System.out.print(j + " ");

              if(j==5)
                 break;       //Statement 1
           }
        }
     }
  }

  Output :

        1 2 3 4 5
        1 2 3 4 5
        1 2 3 4 5
        1 2 3 4 5
        1 2 3 4 5
        1 2 3 4 5
        1 2 3 4 5
        1 2 3 4 5
        1 2 3 4 5
        1 2 3 4 5
```

In th above example, statement 1 will break the inner loop and jump outside from inner loop to execute outer loop.

Example with labelled loop

```
          //WithLabelledLoop.java

    class WithLabelledLoop
    {
      public static void main(String args[])
      {
        int i,j;

        loop1: for(i=1;i<=10;i++)
        {
          System.out.println();

          loop2:   for(j=1;j<=10;j++)
          {
            System.out.print(j + " ");

            if(j==5)
              break loop1;     //Statement 1
          }
        }
      }
    }

    Output :

          1 2 3 4 5
```

In th above example, statement 1 will break the inner loop and jump outside the outer loop.