Java Theory Note 1:

MENU

​  
[11](https://nilabja.wordpress.com/computers/theory-of-java/#comments)**What is Java?**  
Java is a 3rd Generation Object Oriented programming Language, developed by James Gosling in the year 1995. It was initially codenamed as Green and hypothetically called Oak named after the Oak tree outside Gosling’s house.  
It is case sensitive and has a unique two step translation process that includes both Compilation and Interpretation which makes Java Platform independent.  
**What is Procedure Oriented Programming Language?**  
Pop is a Procedural Oriented Programming Language. In this, the stress is laid on function rather than data. Data may be kept floating throughout the Programming. Hence by scanning the whole program from beginning to the end and we can rectify any error caused. E.g. of POP based language are GW-BASIC, C, etc…  
Characteristics of POP  
1) Emphasis is laid on functions (Logical steps).  
2) Functions share global data.  
3) Data values can keep floating from one function to another.  
4) Uses top down approach of programming.  
**Drawbacks of POP**  
1) As Data values are global to all the functions, you may require making necessary changes in the functions due to change in the data values.  
2) It is not suitable to solve complex problems in real situations.  
What is **Object Oriented Programming Language**?  
OOP is an Object Oriented Programming Language. It is a modular approach to programming in which equal stress is given to data as well as methods and it allows the data be applied within the stipulated program area. It also provides reusability features to develop productivity logic.  
**Features of OOP**  
1. It gives equal stress on data and function.  
2. It makes the program simple by dividing it into a number of objects.  
3. The objects can be used as a bridge to have data flow from one function to another.  
4. Data can be easily modified without any change in the function.  
Advantages of OOP  
1) We can extend the use of existing class through inheritance.  
2) Using the concept data hiding can generate secured program.  
3) We can create different modules in our project through objects.  
4) Multiple instances of an object can be generated to co-exist without any interference.  
5) It is highly beneficial to solve complex problems.  
6) It is easy to modified and maintain software complexity.  
Examples of OOP based languages are Simula, C++, Java, Python, and Smalltalk etc…  
**Basic Principles of OOP**  
1) Encapsulation – Binding up of data members and member functions together into a single unit called (class) is called Encapsulation.  
2) Data Abstraction – Act of representing only essential features without including its background details is called Data Abstraction.  
3) Polymorphism – The ability of a method to behave in more than one form is called polymorphism. Function Overloading is a way to implement it, in which more than one function has the same name but different argument list.  
4) Inheritance – The ability of a class to derive the properties from another class is called Inheritance. The class that inherits is called the Derived / Sub / Daughter class and the class from which it is derived is called Base / Super / Parent class. The keyword used to implement Inheritance is ‘extends’.  
5) Object – It is an identifiable entity with some characteristics and behaviour.  
6) Class – It is a blueprint that represents a set of objects which share common characteristics and behaviour.  
Related Terms :  
**Data binding**– is the process to link to the function call with function signature at run-time i.e., during execution of a program.  
**Attribute**:-The characteristics which make every object having its identity make them distinguished is called attribute.  
**Behaviour**– The behaviour of any class or object is represented through various functions and methods.  
**Message Passing** – When object need to interact with one another they do so by passing information to and from one another, this is called Message Passing  
**Data Hiding** – is the data which cannot be accessed directly outside, class premises although they are available in the same program  
**Garbage Collection** – The Garbage-collected Heap is where the objects in Java programs are stored. Whenever we allocate an object using new operator, the heap comes into picture and memory is allocated from there. Java does this automatically using Garbage collection mechanism, using an algorithm named Mark & Sweep logic. Remember that the local object reference resides on Stack but the actual object resides in Heap only. Also, arrays in Java are objects, hence they also resides in Garbage-collected Heap.  
**Different Types of Java Programs**  
1) **Standalone Application** – An application program that is developed by programmers which consists of various methods and these methods being invoked from within a ‘main ( )’ method is called standalone application.  
2) **Internet Applets**– These are small java programs that are embedded in a web page and which requires a web browser to be run/executed.  
3) **Servlets**– They extend the functionality of web browser.  
Source Code – The High level language code written by programmers which is not understandable by the computer is known as Source Code. It has an extension of \*.java  
Byte Code – is an intermediate code that consists of a set of pseudo machine language instructions that are understood by the JVM and are independent of the underlying hardware. It is called byte code because each chunk of code is of 8 BITS (1 byte = 8 Bits).  
The byte code files have an extension of \*.class  
**Object Code**– The interpreter of Java named JVM (which stands for Java Virtual Machine) then converts the intermediate Byte Code into machine specific executable which runs on the host machine.  
API (Application Programming Interface) – consists of libraries of pre compiled code that programmers can use in their application and programs for designing softwares. Hence we can say that Java API consists of functions and variables that programmers can reuse.  
**Java Packages** – Packages are collection of similar nature classes. A package can be created by using the keyword ‘package’ and the keyword used to include a package in our program is ‘import’. Java contains an extensive library of pre-written classes grouped together into packages –  
 θjava.lang.\* It is a default package containing String, Math, Integer, thread etc.  
 θjava.io.\* It is the basic Input Output package of Java  
 θjava.util.\* The java utility package  
 java.applet.\* The applet packageθ  
 θjava.net.\* The Networking package of Java  
 θjava.awt.\* It stands for Abstract Windows Toolkit, for designing Windows based applications.  
**Basic Features of Java:-**  
1) Write once run anywhere(WORA)  
2) Platform Independent  
3) Offers many security features to make its program safe and secure  
4) Built in graphics & supports multimedia  
5) Light Weight code  
6) Open Product… i.e., freely available to all.  
**Blue – J**  
Blue-J is an **Integrated Development Environment (IDE)** for writing, editing, compiling, testing, executing and debugging the Java programming language, developed mainly for educational purposes, but also suitable for small-scale software development. Blue J was initially designed at University of Kent by **David Barns** and **Michael Kollins**  
The main screen graphically shows the class structure of an application under development (in an UML-like diagram), and objects can be interactively created and tested. This interaction facility, combined with a clean, simple user interface, allows easy experimentation with objects under development.  
**Features of Blue – J**  
1. Simplicity of the interface – The GUI user friendly interface is simpler than in full-scale professional environments, and thus easier to learn.  
2. The “Code Pad” – The code pad is a tool that instantly evaluates arbitrary Java expressions and statements.  
3. Regression testing  
4. Provision for syntax highlighting  
5. Built in debugger.  
6. Program input/output i.e. In Blue-J both the parameter values for and the return values from method calls can be entered / inspected directly.  
7. Java ME support – Java ME (Micro Edition) projects can be developed and deployed from Blue-J.  
**IDE for Java (other than Blue J)**  
Sl IDE Brief Description  
1 Eclipse Free, open source IDE. The most popular of the Java IDEs, but harder to set up and configure than the commercial ones.  
2 J Builder Lots of J2EE support including UML and app server integration.  
3 Emacs This is a powerful, free editor that does color syntax highlighting, automatic indentation, and parenthesis/brace balancing.  
4 M7 Nitro X Expensive but very powerful J2EE IDE based on Eclipse.  
**Java Tokens & Character Set**  
It is a set of valid characters that a language can recognize. The character set of Java is UNICODE. This has been selected due its vastness of characters. It is 2 byte character code. The first 128 characters of Unicode are identical to the ASCII character set.  
Tokens are smallest individual units of a Java program. There are 5 different types of tokens in Java –  
1. Keywords  
2. Identifiers  
3. Literals  
4. Operators  
5. Separators Hint: Try to remember KILOS  
**Keywords**are the reserved words which convey a special meaning to a language compiler. They can not be used for any other purpose like function name, variable name or object names.  
There are around 48 keywords, of which const and goto have been reserved even though they are not currently being used for any purpose true, false and null might appear to be keywords but actually they are literals, and hence are reserved words.  
**Identifiers**are fundamental building blocks of a program. They are named memory locations or simply names given to variables, objects, classes, functions, arrays etc… They are case sensitive.  
There are certain Identifier naming rules –  
1. They can consist of infinite number of alphabets and digits  
2. They must start with an alphabet and not with a digit  
3. They can not be a keyword or reserved word  
4. They cannot have any spaces  
5. They cannot contain any special character other than \_(underscore), $(dollar)  
While naming identifiers there are certain conventions that make them identifiable  
1. Class names begins with an Uppercase character  
2. Identifiers having multiple words, the first character of 2nd and subsequent words are in Uppercase  
3. Constants are in all CAPITALS  
Comments (Remarks):- refers to statements which are ignored by the java compiler and increase the readability of a program. These are used to state a purpose of the instructions used in a program.  
**Literal**are the fixed values that are assigned to variables which do not change its values during program execution. They are of 5 types –  
**1.Integer Literal**  
i.Decimal  
ii.Octal  
iii.Hexadecimal  
To represent Decimal Integers we simply use the integer type number as literal, e.g. int p = 45; So, System.out.println(p); // Shows 45 as output  
To represent Octal Integers however we precede the number with a 0 (zero) as literal, e.g. int p = 045; System.out.println (p); // Shows 37 as output coz, (45) in octal is 37 in decimal number system.  
Moreover to represent any Hexadecimal Integer we use 0x (Zero and lowercase letter x) preceding the number as hexadecimal literal e.g. int p = 0x45;  
System.out.println (p); // Shows 69 as output coz, (45) in hexadecimal is 69 in decimal number system.  
2. **Real Literals**  
i. Floating point literals have 7 digits of precision after the decimal point  
ii. Double literals have 16 digits of precision after decimal point  
3. **Character Literals** are enclosed within a pair of single quotes  
char ch = ‘A’;  
Character type user inputs are taken in the following way –  
e.g. char ch = (char) in.read ( ) ;  
4. **Boolean Literal**  
There are only 2 types of boolean literals i.e. true & false the default being false.  
5. **Null Literal**  
Operator – are the entities or symbols that tell the compiler that what operation has to be performed with one, two or three number of operands within a given expression.  
There are approximately a set of 37 different operators that Java uses  
Operators can either operate on 1, 2 or 3 operands and accordingly named as –  
 Unary (works on 1 operand) e.g. + + and – –θ  
 Binary (works on 2 operands) e.g. +, – , \* , / ,θ >, , = , > , << )  
 Special Operators (this , dot (.) , new, (type) , instanceof )θ  
**Data types** – are the entities that tell the compiler that which variable will hold what kind of values.  
Primitive data types – Also called as Basic Data type. They are pre defined or built in data types because they are already built in java. E.g. byte, short, int, long, float, double, char, boolean.  
Non Primitive Data Types: Directly or indirectly dependent on primitive data types. They store the data values with reference to the address or locations. So, they are called as Reference Data Types. E.g. array, classes interfaces etc…  
**Range, Size and hierarchy of various primitive data types**  
Type ….Size…. Range  
boolean 1 byte (uses 1 bit) true and false  
byte 8 bits (1 bytes) – 128 to + 127  
char 16 bits (2 bytes) 0 to 65,535  
short 16 bits (2 bytes) – 32768 to +32767  
int 32 bits (4bytes) – 2 31 to + 231 – 1  
long 64 bits (8bytes) – 2 63 to + 263 – 1  
float 32 bits (4bytes) – 3.4E+38 to +3.4E+38  
double 64 bits (8bytes) – 1.7E +308 to 1.7E+308  
Type conversion The process of convert one primitive datatype to another either from lower precision to higher or vice versa I known as type conversion. This can be achieved in two ways –  
**Implicit Data Type Conversion** – Such Type of conversion is performed by the compiler without programmer’s intervention. In this type of conversion a datatype of lower precision gets converted to a datatype with higher precision.  
e.g. int a = 10;  
double d = a;  
System.out.println(d);  
The above code snippet gives 10.0 as output; this clearly indicates that the int datatype got automatically converted to double.  
**Explicit data type Conversion** – Such type of conversion is user defined conversion which forces an expression to be converted into specific type. Generally this is done to force a conversion from higher precision data type to lower.  
e.g. double pi = 3.1415;  
int i = pi;  
The above statement is considered illegal and hence gives an ERROR message “POSSIBLE LOSS OF PRECISION”  
Hence the above code should be Type Casted in the following manner –  
int i = (int) pi;  
System.out.println(i);  
Now, the output to the above code will be 3  
**Potential problems with Type Casting**  
1. When a datatype is converted from double to float results in loss of precision i.e. a double datatype has 16 digits after decimal point which gets truncated to 7 digits.  
2. When a datatype is converted from double or float to integer type results in loss of entire fractional part leaving behind only the integral part.  
3. When a higher order integer datatype like int or long is converted to byte or short might result in the data going out of range.  
Take an example  
int a = 130;  
byte b = (byte) a;  
System.out.println(b);  
The output for the above code is – 126  
Explanation for the above –  
The above code tries to type cast int to byte, the data range for int is – 2 31 to + 231 – 1 whereas that of byte is – 128 to + 127, hence think in terms of a Number Line, when we start filling 130 in the number line starting from 0 (zero) we reach the end of the line at 127 and so the filling process continues from the –ve end and finally reaches – 126, hence the output…  
**Operators in Java**  
**Operator**– are the entities or symbols that tell the compiler that what operation has to be performed with one, two or three number of operands within a given expression.  
There are approximately a set of **37** different operators that Java uses  
Operators can either operate on 1, 2 or 3 operands and accordingly named as –

1. **Unary** (works on 1 operand) e.g. + + and – –
2. **Binary** (works on 2 operands) e.g. +, – , \* , / , >, <, == etc…
3. **Ternary** (works on 3 operands) e.g. ? :

**Operators can categorized as –**

1. Arithmetic Operators (+ , – , \* , / , %)
2. Relational Operators (> , < , >= , <= , == , !=)
3. Logical Operators (&&, || , !)
4. Conditional Operators (? : )
5. Bitwise Operators (&, | , ~ , ^)
6. Shift Operators (>> , << )
7. Special Operators (this , dot (.) ,  new, (type) , instanceOf )

**Shortcut Operations in Java**  
Java implements shortcut methods to work with basic arithmetic operators like  
+, – , \* , / and %  
Hence, a = a + 5; can be written as a + = 5;  
And the above + = operator is called Addition Assignment operator.  
Similarly we have –

* Subtraction Assignment – =
* Multiplication Assignment \*=
* Division Assignment /=
* Modulus Assignment %=

**Unary Operators**  
*Post/Pre – Increment and Post/Pre – Decrement Operators*  
a = a + 1 can also be written as either a++ (**Post Increment**) or ++a (**Pre Increment**)  
a = a – 1 can also be written as either a– – (**Post Decrement**) or – –a (**Pre Decrement**)  
**The difference between Pre and Post is** **–**  
In a **Postfix** operator like Post Increment or Post Decrement the value of the operand is first used and then increased or decreased as the case may be for e.g.  
int a = 5;  
System.out.println(a++);  
Gives 5 as output as the value of ‘a’ is first used and then increased to 6, so if we have one more print statement after the above stated line will give 6 as output.  
System.out.println(a);          // Output is 6  
In a **Prefix** operator like Pre Increment or Pre Decrement the value of the operand is first increased or decreased as the case may be and then used for e.g.  
int a = 5;  
System.out.println(++a);  
Gives 6 as output as the value of ‘a’ is first incremented and then displayed.  
**Relational Operators**are used to find the relationship between two quantities, whether they are equal, un-equal, greater than less than etc…  
**Symbol**  
**Relationship**  
  
>  
Greater than  
  
<  
Less than  
  
>=  
Greater than or equal to  
  
<=  
Less than or equal to  
  
==  
Equal to  
  
!=  
Not equal to  
**Logical Operators**are used to join 2 or more relational operators and of 3 types –

1. && (and) returns true when both or all the relations are true
2. || (or) returns true when any one of the relations are true
3. ! (not) inverts true to false and false to true

Say if the value of a=4, b=6 then…  
if(a>5 && b>5) will return false coz both the conditions are not true  
if(a>5 || b>5) will return true coz one of the conditions is true  
if(!true) will return false  
It is not mandatory to have a relation always inside an ‘if’, even without any ‘if’ clause we can check the return of any relation like –  
Say the value of x=7; then  
System.out.println(x==7); will give the output as true, and if we place (!) in front of the relation, System.out.println( ! (x==7)); will invert the output from true to false.  
**Bitwise and Shift Operators**  
**Operator**  
**Name**  
**Description**  
**Example**  
**Result**  
  
a & b  
and  
1 if both bits are 1 and 0 in all other cases  
3 & 5  
1  
  
a | b  
or  
0 if both the bits are 0, rest all other cases 1  
3 | 5  
7  
  
a ^ b  
xor  
1 if two bits are different and 0 in all other cases  
3 ^ 5  
6  
  
~a  
not  
Inverts the bits, after transforming the entire number into full bit representation  
~3  
-4  
  
n << p  
left shift  
Shifts the bits of n towards left by p positions thereby adding Zeros to the extreme LHS  
3 << 2  
12  
  
n >> p  
right shift  
Shifts the bits of n towards right by p positions resulting in loss of bits from LHS  
5 >> 2  
1  
  
**In all the above examples the operands (values) must be converted to Binary and then implemented with respective operators except for Right and Left Shift operators where only the variable ‘n’ is converted to binary and not ‘p’**  
Don’t confuse **&&**, which is the ***logical****and*, with **&**, which is the uncommon ***bitwise****and*. Although the bitwise *and*can also be used with boolean operands, this is extremely rare and is almost always a programming error.  
**Conditional Operator**  
The only Ternary operator of Java is the Conditional Operator, **? :** which can be used as an alternative to if…else  
Syntax: variable = condition ? true : false ;  
e.g. int n = 100>50 ? 1 : 0;  
In the above code the value of ‘n’ becomes 1 as the condition 100>50 is true, had it been wrong the value of ‘n’ would have been 0;  
**Special Operators**  
q     **[ ]** this operator is used to assign the size for an array and to access any particular element within an array.  
q     **( )** the parenthesis operator is used to enclose the arguments to a function in the function prototype line and invoke a function by enclosing the list of arguments.  
q     **Dot (.)** the dot operator is used to access the instance/class members of a class through an object or class name.  
q     **new** – the new operator is used to assign and allocate memory to new objects and arrays.  
q     **instanceof** – this operator checks whether the first argument is an instance of the second argument or not.  
e.g. **str** instanceof **String**                  // will return true if **str** is a **String** variable  
  
  
**Precedence Table**  
  
Operator Precedence  
  The entire list 1.      ( ) . [] (args) post ++ -- 2.      ! ~ unary + - pre ++ -- 3.      (type) new 4.      \*/ % 5.      + - 6.      << >> >>> 7.      < <= > >= instanceof 8.      == != 9.      & 10.  ^ 11.  | 12.  && 13.  || 14.  ?: 15.  = += -= etc  Try to Remember only these

1. ( )
2. unary operators
3. \* / %
4. + –
5. comparisons
6. && ||
7. ? :
8. = assignments

**Math class functions in Java** (The **Math** class belongs to the default package of Java named **java.lang**)  
**SlFunction Name**  
**Description**  
  
1  
Math.sin(r)Returns sine of angle ‘r’ given in radians  
2  
Math.cos(r)Returns cosine of angle ‘r’ given in radians  
3  
Math.tan(r)Returns tangent of angle ‘r’ given in radians  
4  
Math.asin(x)Returns the angle in radians whose sine value is ‘x’  
5  
Math.acos(x)Returns the angle in radians whose cosine value is ‘x’  
6  
Math.atan(x)Returns the angle in radians whose tangent value is ‘x’  
7  
Math.toDegrees(r)Converts Radians to Degrees  
8  
Math.toRadians(d)Converts Degrees to Radians  
9  
Math.min(a, b)Returns the smaller among a and b  
10  
Math.max(a, b)Returns the larger among a and b  
11  
Math.abs(a)Convert -ve magnitude to +ve magnitude.  
12  
Math.sqrt(x)Returns the square root of ‘x’  
13  
Math.pow(x, y)Returns x to the power of y  
14  
Math.exp(x)Returns ‘e’ to the power of ‘x’ where e=2.718 called Euler’s  Constant just like pi  
15  
Math.cbrt(x)Returns the cube root of ‘x’  
16  
Math.log(x)Returns the Natural logarithm of ‘x’  
17  
Math.floor(x)Returns the integral value of x less than or equal to ‘x’ but datatype returned is double, hence Math.floor(7.9) will give 7.0  
18  
Math.ceil(x)Returns the integral value of x greater than or equal to ‘x’ but datatype returned is double, hence Math.ceil(7.9) will give 8.019  
Math.rint(x)Returns the integral value of x by rounding off the fractional part to the nearest integer , but data type returned is double, hence Math.rint(7.9) will give 8.0  
20  
Math.round(x)Returns the integral value of x by rounding off the fractional part to the nearest integer, but datatype returned is int, hence Math.round(7.9) will give 8  
21  
Math.random()Generates a double type random number between 0 to 1  
22  
Math.PIThis returns the 16 digits precise value of PI(Note that it is not a function)**Comment Lines in Java**  
 // comments — single line commentθ  
After the two // characters, Java ignores everything to the end of the line. This is the most common type of comment.  
 /\* … \*/ comments — multiple line commentθ  
After the /\* characters, Java will ignore everything until it finds a \*/. This kind of comment can cross many lines, and is commonly used to “comment out” sections of code — making Java code into a comment while debugging a program.  
Always write braces. It is good programming style to always write the curly braces, {}, although they are not needed if the clause contains only a single statement. There are two reasons this is good.  
• **Reliability**. When code is modified, the indentation is such a strong indicator of structure that the programmer may not notice that the addition of a statement at the “correct” indentation level really isn’t included in the scope of the if statement. This is a surprisingly common error.  
• **Readability**. It is faster to read code with the braces because the reader doesn’t have to keep in mind whether they are dealing with an un-braced single statement or a braced block.  
Whitespace (e.g., a blank line, spaces)  
Insert blank lines to separate sections of your program. It’s like starting a new paragraph in English. The compiler ignores them — it’s for us humans.  
**Exception Handling in Java**  
Exception refers to any contradictory or unusual situation which can be encountered during a program execution. In Java Exceptions (or Errors) has been categorized broadly under three heading –  
1. **Syntax Errors** – A Syntax error occurs when the programmer writes a grammatically incorrect code that does not support the rules of programming of that particular language. This type of error gets detected by the compiler. E.g. missing semicolon, using keywords as identifier name, unmatched pair of { }, undefined functions, undeclared and un-initialized variables etc…  
2. **Logical Errors** – A Logical error is born out of a programmer who misunderstands the logic of the program and writes a syntactically correct code but which does not comply to the algorithm of the problem. E.g. The program demanded squaring of an integer type number but the programmer writes a code that determines the square root of the number.  
3. **Runtime Errors** – A Runtime error is shown when a syntactically and logically correct program with a set of inputs which is invalid for a given expression. The same expression is valid for another set of values. E.g. int quo = div / n, the expression is valid for all sets of positive values of n but not if n = 0.  
**Exception handling is ideal for –**  
1. Processing exceptional situations.  
2. Processing exceptions for components which cannot handle them directly.  
3. Processing exceptions for widely used components that should not process their own exceptions.  
4. Large Projects that requires uniform error processing.  
**Advantages** of exception handling:-  
1. It separates error handling code from the normal code.  
2. It enhances the readability.  
3. It makes clear and strong program code.  
The process of exception handling is implemented using a try … catch block, in which the try block contains the code which has probability of having errors during runtime and the catch block contains code to handle the situation.  
**Common Errors**  
0 < x < 100  
Comparison operators can be used with two numbers. Although you can write 0 < x = or <= instead of ==. For example, because the decimal number 0.1 can not be represented exactly in binary, (0.1 + 0.1 + 0.1) is not equal to 0.3  
String issues using = = and equals ( )  
Usually you want to know if the two Strings have the same value, for e.g.  
if (name == “Californication”) // Legal, but SURELY WRONG  
Reason is the = = operator is not comparing the values of name and Californication instead it is just comparing the references i.e. it is checking whether both of them are strings or not and hence returns true and to the programmer it reveals to be perfectly LEGAL. To prove this we can take another e.g. say, –  
String name = “Californication”;  
System.out.println (name. substring (0,3)); // Gives the output as Cal  
But when they are compared using = = gives the output as Un-Equal  
if (name. substring (0,3) = = “Cal”)  
System.out.println (“Equal”);  
else  
System.out.println (“Un-Equal”);  
Hence always use equals ( ) method of String class to compare the values of String class objects.  
**NaN**  
No exceptions are generated by floating-point operations. Instead of an interruption in execution, the result of an operation may be positive infinity, negative infinity, or NaN (not a number). Division by zero or overflow produces infinity. Subtracting two infinities produces a NaN.  
**Infinity**  
Similarly no exceptions are generated for statements like Math.pow(0,–1) and such statements will result in Infinity as output  
**Flow Control in Java Programs**  
**if Statement**  
The purpose of the if statement is to make decisions, and execute different parts of your program depending on a boolean true/false value.  
The ‘if’ statement has this form, where condition is true or false  
if (condition)  
{  
… // Do this clause if the condition is true.  
}  
Alternative forms of if statements –  
if (condition)  
{  
… // Do this clause if the condition is true  
}  
else  
{  
… // Do this clause if the condition is false  
}  
Always make sure that the number of if clause should not exceed the number of else clause, because it will result in ‘dangling else problem’, in such case make use of if…else if…else construct as shown below –  
if (condition 1)  
{  
… // Do this clause if the condition is true  
}  
else if (condition 2)  
{  
… // Do this clause if the condition is false  
}  
else  
{  
… // Do this clause if the condition is false  
}  
**switch…case Statement**  
The if statement allows you to select one of two sections of code to execute based on a boolean value (only two possible values). The switch statement allows you to choose from many statements based on an integer (including char)  
switch (expr)  
{  
case c1: … break;  
case c2: … break;  
…  
default:  
}  
**switch**  
The switch keyword is followed by a parenthesized integer expression, which is followed by the cases, all enclosed in braces.. The switch statement executes the case corresponding to the value of the expression. Normally the code in a case clause ends with a break statement, which exits the switch statement and continues with the statement following the switch. If there is no corresponding case value, the default clause is executed. If no case matched and there is no default clause, execution continues after the end of the switch statement.  
**case**  
The case keyword is followed by an integer constant and a colon. This begins the statements that are executed when the switch expression has that case value.  
**default**  
If no case value matches the switch expression value, execution continues at the default clause. This is the equivalent of the “else” for the switch statement. It is written after the last case be convention, and typically isn’t followed by break because execution just continues out the bottom of switch if this is the last clause.  
**break**  
The break statement causes execution to exit to the statement after the end of the switch. If there is no break, execution falls through into the next case. Flowing directly into the next case is almost always an error.  
A switch statement can often be rewritten as a substitute for ‘if’ statement in a straightforward manner. For example, the preceding switch statement could be written as follows. When one of a number of blocks of code is selected based on a single value, the switch statement is generally easier to read. The choice of if or switch should be based on which is more readable.  
**Limitations of switch case**  
Java’s switch statement, which was taken directly from C++ to increase its attractiveness to C++ programmers, is not well loved.  
• No ranges. It doesn’t allow ranges, eg case 90-100:. Many other languages do.  
• Integers only. It requires integers and doesn’t allow useful types like String. Many other languages do.  
• Error-prone. It is error-prone and a common source of bugs – forgetting break or default silently ignores errors. Some languages have eliminated these dangerous situations.  
• Fall through. Another disadvantage of switch…case is that in absence of break statement in the respective cases will cause the flow control to cascade down and execute all the cases that does not contain ‘break’ statements until it reaches end of switch or a case that implements a break. This is called fall through.  
**Iteration through Loops**  
The purpose of loop or iterative statements is to repeat a block of Java statements several times depending upon the condition mentioned within the loop. There are two kinds of looping statements in Java.  
1. Entry Controlled Loop  
2. Exit Controlled  
An **Entry Controlled** loop which is also known as Pre Tested Loop the condition of the loop is first checked and then the loop body is executed, which means if the condition is false the loop doesn’t get executed even for once.  
E.g. a. for LOOP  
b. while LOOP  
An **Exit Controlled** loop which is also known as Post Tested Loop the condition is checked at the end of the loop body, which means even if the condition is false the loop gets executed at least for once.  
E.g. a. do…while LOOP  
If the body of a loop has more than one statement, you must put the statements inside braces. If there is only one statement, it is not necessary to use braces { }. However, many programmers think it is a good idea to always use braces to indicate the scope of statements. Always using braces allows the reader to relax and not worry about the special single statement case.  
while statement  
The while statement is used to repeat a block of statements while some condition is true. The condition must become false somewhere in the loop, otherwise it will never terminate.  
Syntax:  
initialization;  
while (condition)  
{  
Body;  
update;  
}  
//… While loop to display squares of natural nos. less than 10  
int i = 1;  
while (i <=10)  
{  
System.out.println(i\*i);  
i++;  
}  
**for statement**  
Many loops consist of three operations surrounding the body:  
(1) initialization of a variable,  
(2) testing a condition, and  
(3) updating a value before the next iteration.  
The for loop groups these three common parts together into one statement, making it more readable and less error-prone than the equivalent while loop. For repeating code a known number of times, the for loop is the right choice.  
There are three clauses in the ‘for’ statement.  
1. The **initialization** statement is done before the loop is started, usually to initialize an iteration variable.  
2. The **condition**/**test expression** is tested before each time the loop is done. The loop isn’t executed if the boolean expression is false (the same as the while loop).  
3. The **update** statement is done after the body is executed. It typically increments an iteration variable.  
Syntax:  
for (initialization ; condition/test expression; update)  
{  
body  
}  
//… For loop to to display squares of natural nos. less than 10  
for (int i = 1; i <= 10; i++)  
{  
System.out.println(i\*i);  
}  
**do…while statement**  
When you want to test at the end to see whether something should be repeated, the do…while statement is the natural choice  
Syntax  
do  
{  
. . .  
} while (condition);  
Always remember that a do…while statement has a semi-colon at its end.  
The for loop is shorter, and combining the initialization, test, and increment in one statement makes it easier to read and verify that it’s doing what you expect. The for loop is better when you are counting something. If you are doing something an indefinite number of times, while loop may be the better choice.