Introduction

Java is a simple and yet powerful object oriented programming language and it is in many respects similar to C++.

Java is created by James Gosling from Sun Microsystems (Sun) in 1991. The first publicly available version of Java (Java 1.0) was released in 1995.

Java is defined by a specification and consists of a programming language, a compiler, core libraries and a runtime machine(Java virtual machine).
The Java runtime allows software developers to write program code in other languages than the Java programming language which still runs on the Java virtual machine.
The Java platform is usually associated with the Java virtual machine and the Java core libraries.

Java virtual machine

The Java virtual machine (JVM) is a software implementation of a computer that executes programs like a real machine.

Java Runtime Environment vs. Java Development Kit

A Java distribution typically comes in two flavors, the Java Runtime Environment (JRE) and the Java Development Kit (JDK).
The JRE consists of the JVM and the Java class libraries. Those contain the necessary functionality to start Java programs.
The JDK additionally contains the development tools necessary to create Java programs. The JDK therefore consists of a Java compiler, the Java virtual machine and the Java class libraries.

Uses of JAVA

Java is also used as the programming language for many different software programs, games, and add-ons.

Some examples of the more widely used programs written in Java or that use Java include the Android apps, Big Data Technologies, Adobe Creative suite, Eclipse, Lotus Notes, Minecraft, OpenOffice, Runescape, and Vuze.

Following are some of the features of Java :

1. **Simple**

   Java is easy to learn and its syntax is quite simple and easy to understand.

2. **Object-Oriented**

   In java everything is Object which has some data and behaviour. Java can be easily extended as it is based on Object Model.

3. **Platform independent**

   Unlike other programming languages such as C, C++ etc which are compiled into platform specific machines. Java is guaranteed to be write-once, run-anywhere language.
   On compilation Java program is compiled into bytecode. This bytecode is platform independent and can be run on any machine, plus this bytecode format also provide security. Any machine with Java Runtime Environment can run Java Programs.

4. **Secured**

   When it comes to security, Java is always the first choice. With java secure features it enable us to develop virus free, temper free system.

Java program always runs in Java runtime environment with almost null interaction with system OS, hence it is more secure.

5. **Robust**

Java makes an effort to eliminate error prone codes by emphasizing mainly on compile time error checking and runtime checking. But the main areas which Java improved were Memory Management and mishandled Exceptions by introducing automatic Garbage Collector and Exception Handling.

6. **Architecture neutral**

Compiler generates bytecodes, which have nothing to do with a particular computer architecture, hence a Java program is easy to intrepret on any machine.

7. **Portable**

Java Bytecode can be carried to any platform. No implementation dependent features. Everything related to storage is predefined, example: size of primitive data types

8. **High Performance**

Java is an interpreted language, so it will never be as fast as a compiled language like C or C++. But, Java enables high performance with the use of just-in-time compiler.

9. **Multithreaded**

Java multithreading feature makes it possible to write program that can do many tasks simultaneously. Benefit of multithreading is that it utilizes same memory and other resources to execute multiple threads at the same time, like While typing, grammatical errors are checked along.

10. Distributed

We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.

11. Interpreted

An interpreter is needed in order to run Java programs. The programs are compiled into Java Virtual Machine code called bytecode. The bytecode is machine independent and is able to run on any machine that has a Java interpreter. With Java, the program need only be compiled once, and the bytecode generated by the Java compiler can run on any platform.

Pros and Cons of JAVA

Pros:

1. Java is Simple
2. Java is object-oriented because programming in Java is centered on creating objects, manipulating objects, and making objects work together. This allows you to create modular programs and reusable code.
3. One of the most significant advantages of Java is Platform indenpendence.
4. Java is Secure: Java is one of the first programming languages to consider security as part of its design.
5. Java is Multithreaded: Multithreaded is the capability for a program to perform several tasks simultaneously within a program.
6. Java is Robust: Robust means reliable and no programming language can really assure reliability.

Cons:

1. Java can be perceived as significantly slower and more memory-consuming than natively compiled languages such as C or C++.
2. No local constants. In Java, variables that belong to a class can be made constant by declaring them to be final. Variables that are local to a method cannot be declared final, however.
3. Java is predominantly a single-paradigm language. However, with the addition of static imports in Java 5.0 the procedural paradigm is better accommodated than in earlier versions of Java

Environment Setup

We need to install the Java Development Toolkit aka JDK, which is bundled with the Java Runtime Environment.
At this moment, the latest JDK versions is JDK 8.
All you have to do is head to the main download page provided by Oracle , and download the latest version that you will find.

Follow the instructions to download java and run the .exe to install Java on your machine. Once you installed Java on your machine, you would need to set environment variables to point to correct installation directories.

Assuming you have installed Java in c:\Program Files\java\jdk

1. Right-click on 'My Computer' and select 'Properties'.
2. Click on the 'Environment variables' button under the 'Advanced' tab.
3. Now, alter the 'Path' variable so that it also contains the path to the Java executable.

Example, if the path is currently set to
'C:\WINDOWS\SYSTEM32',
then change your path to read
'C:\WINDOWS\SYSTEM32; c:\Program Files\java\jdk\bin'.

Setting up the path for Linux, Ubuntu, UNIX, Solaris
Environment variable PATH should be set to point to where the Java binaries have been installed.
Refer to your shell documentation if you have trouble doing this.

Example, if you use bash as your shell, then you would add the following line to the end of your
'.bashrc: export PATH=/path/to/java:$PATH'

Up until now we have installed a variety of tools towards setting up our Java Development environment.

Since the JDK is already installed (from step one) we could actually jump to coding just by using our text editor of choice (NotePad++, TextPad, NotePad, Ultra Edit etc) and invoking the javac and java commands from the command line.

Free IDE for Java

Netbeans: NetBeans IDE provides Java developers with all the tools needed to create professional desktop, mobile and enterprise applications.

Eclipse: Eclipse is another free Java IDE for developers and programmers and it is mostly written in Java. Eclipse lets you create various cross platform Java applications for use on mobile, web, desktop and enterprise domains.

Executing First Program

```
class Simple
{
public static void main(String args[])
{
System.out.println("Hello Java");
}
}
```

save this file as Simple.java
To compile: javac Simple.java
To execute: java Simple
It will give output as Hello Java

Lets see what this is :
class keyword is used to declare a class in java.

public keyword is an access modifier which represents visibility, it means it is visible to all.

static is a keyword, if we declare any method as static, it is known as static method. The main method is executed by the JVM, it doesn't require to create object to invoke the main method. So it saves memory.

void is the return type of method, it means it doesn't return any value.

main is a entry point of the program. Execution of programs starts from main. It is called by Runtime System

String[] args is used for command line argument. We will learn it later.

System.out.println() is used print statement.

Variables

A variable provides us with named storage that our programs can manipulate.
Each variable in Java has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

You must declare all variables before they can be used.
The basic form of a variable declaration is shown here:
data_type variable = value;

Here data type is one of Java's datatypes and variable is the name of the variable. To declare more than one variable of the specified type, you can use a comma-separated list.

Following are valid examples of variable declaration and initialization in Java:

```
int a, b, c;            // Declares three ints, a, b, and c.

int a = 10, b = 10;     // Example of initialization

double pi = 3.14159;    // declares and assigns a value of PI.

char a = 'a';
// the char variable a iis initialized with value 'a'
```

Constant: During the execution of program, value of variable may change. A constant represents permanent data that never changes.

If you want use some value likes p=3.14159; no need to type every time instead you can simply define constant for p, following is the syntax for declaring constant.
Static final datatype ConstantName = value;

Example: static final float PI=3.14159;

Data Types

Every variable in Java has a data type. Data types specify the size and type of values that can be stored.
Data types in Java divided primarily in two types:

1. Primitive(intrinsic)
2. Non-primitive.

Primitive types contains Integer, Floating points, Characters, Booleans And Non-primitive types contains Classes, Interface and Arrays.

Integer: This group includes byte, short, int and long, which are whole signed numbers.

Floating-point Numbers: This group includes float and double, which represent number with fraction precision.

Characters: This group includes char, which represents character set like letters and number.

Boolean: This group includes Boolean, which is special type of representation of true or false value.

Some data types with their range and size:

byte : -128 to 127 (1 byte)
short : -32,768 to +32,767 (2 bytes)
int : -2,147,483,648 to +2,147,483,647 (4 bytes)
float : 3.4e-038 to 1.7e+0.38 (4 bytes)
double : 3.4e-038 to 1.7e+308 (8 bytes)
char : holds only a single character(2 bytes)
boolean : can take only true or false (1 bytes)

Variable Scope

There are three kinds of variables in Java:

Local Variable

1. variable that is declared inside the method is called local variable.
2. Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor or block.
3. Access modifiers cannot be used for local variables.
4. Local variables are visible only within the declared method, constructor or block.
5. There is no default value for local variables so local variables should be declared and an initial value should be assigned before the first use.

Instance Variable

1. A variable that is declared inside the class but outside the method is called instance variable . It is not declared as static.
2. Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
3. Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
4. Instance variables can be declared in class level before or after use.
5. Access modifiers can be given for instance variables.
6. Instance variables have default values. For numbers the default value is 0, for Booleans it is false and for object references it is null. Values can be assigned during the declaration or within the constructor.
7. Instance variables can be accessed directly by calling the variable name inside the class. However within static methods and different

class (when instance variables are given accessibility) should be called using the fully qualified name . ObjectReference.VariableName.

Class/static variables

1. A variable that is declared as static is called static variable. It cannot be local.
2. Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
3. There would only be one copy of each class variable per class, regardless of how many objects are created from it.
4. Static variables are stored in static memory.
5. Static variables are created when the program starts and destroyed when the program stops.
6. Visibility is similar to instance variables.
7. Static variables can be accessed by calling with the class name ClassName.VariableName.

Example

```
class A
{
int data=50;
//instance variable

static int m=100;
//static variable

void method()
{
int n=90;
//local variable
}
}//end of class A
```

TypeCasting

Casting is an operation that converts a value of one data type into a value of another data type.
The syntax for type casting is to give the target type in parenthesis followed by the variable name.

Example:
float f = (float) 10.1;
Int i = (int)f;
in this case, value of i is 10, the fractional part is discarded, while using type casting there is a chance of lost information that might lead to inaccurate result.

Example:
int i = 10000;
byte s = (short) i;
In this example value of s becomes 10, which is totally distorted, to ensure correctness; you can test if the value is in the correct target type range before using type casting.

Casts that results in no loss of information
byte => short, char, int, long, float, double

short => int, long, float, double

char => int, long, float, double

int => long, float, double

long => float, double

float => double

Operators

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

1. Arithmetic Operators
2. Relational Operators
3. Bitwise Operators
4. Logical Operators
5. Assignment Operators
6. Misc Operators

Arithmetic Operators:

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

- + Additive operator (also used for String concatenation)
- - Subtraction operator
- * Multiplication operator
- / Division operator
- % Remainder operator

Relational Operators:

There are following relational operators supported by Java language

- > Greater than
- < Less than
- == Equal to
- != Not equal to
- >= Greater than or equal to
- <= Less than or equal to

Bitwise Operators:

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.

- ~ Unary bitwise complement
- << Signed left shift
- >> Signed right shift
- >>> Unsigned right shift & Bitwise AND
- ^ Bitwise exclusive OR
- | Bitwise inclusive OR

Logical Operators:

The following table lists the logical operators:

- && Conditional-AND
- || Conditional-OR
- ?: Ternary (shorthand for if-then-else statement)

Assignment Operators:

There are following assignment operators supported by Java language:

- = Simple assignment operator
- += Add AND assignment operator
- -= Subtract AND assignment operator
- *= Multiply AND assignment operator
- /= Divide AND assignment operator
- %= Modulus AND assignment operator
- <<= Left shift AND assignment operator.
- >>= Right shift AND assignment operator
- &= Bitwise AND assignment operator.
- ^= bitwise exclusive OR and assignment operator.
- |= bitwise inclusive OR and assignment operator.

Increment and Decrement Operators

Increment and decrement operators are used to add or subtract 1 from the current value of oprand.

- ++ increment
- -- decrement

Increment and Decrement operators can be prefix or postfix.
In the prefix style the value of oprand is changed before the result of expression and in the postfix style the variable is modified after result.

For eg.

```
a = 9;
b = a++ + 5;
/* a=10 b=14 */

a = 9;
b = ++a + 5;
/* a=10 b=15 */
```

Miscellaneous Operators

There are few other operators supported by Java Language.

- **Conditional Operator (? :)**

  Conditional operator is also known as the ternary operator. The operator is written as: variable x = (expression) ? value if true : value if false

- **Instance of Operator:**

  This operator is used only for object reference variables. instanceof operator is wriiten as: (Object reference variable) instanceof (class/interface type)

Expression

Expressions perform operations on data and move data around. Some expressions will be evaluated for their results, some for their side effects, some for both.

An assignment expression has the following form.

1. variable-expression
2. assignment-operator expression

The variable expression can be just the name of a variable, or it can be an expression that selects a variable using array indices. The value type of the right-hand-side expression must be compatible with the variable type.

An assignment expression is most often used for its side effect: it changes the value of the variable selected by the variable expression to the value of the expression on the right-hand side. The value of the assignment expression is the value that is assigned to the selected variable.

An expression can have three kinds of result:

1. a value, such as the result of: (4 * i)
2. a variable, such as the result of: i = 4
3. nothing (in the case of an invocation of a method declared as void)

In most common assignment expressions, the assignment operator is =.
Then the assignment expression has the following form.
variable-expression = expression

The Java arithmetic and bitwise operators can be combined with = to form assignment operators.
For example, the += assignment operator indicates that the right-hand side should be added to the variable, and the *= assignment operator indicates that the right-hand side should be multiplied into the variable.

Operator Precedence

Certain operators have higher priorities than others. Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. for example, the dot operator has higher precedence than the any other operator.

Precedence 15

- () Parentheses
- [] Array subscript
- · Member selection

Precedence 14

- ++ Unary post-increment
- -- Unary post-decrement

Precedence 13

- + Unary plus
- - Unary minus
- ++ Unary pre-increment
- -- Unary pre-decrement
- ! Unary logical negation
- ~ Unary bitwise complement
- (type) Unary type cast

Precedence 12

- * Multiplication
- / Division
- % Modulus

Precedence 11

- + Addition
- - Subtraction

Precedence 10

- << Bitwise left shift
- >> Bitwise right shift with sign extension
- >>> Bitwise right shift with zero extension

Precedence 9

- < Relational less than
- > Relational greater than
- <= Relational less than or equal
- >= Relational greater than or equal
- instanceof Type comparison (objects only)

Precedence 8

- == Relational is equal to
- != Relational is not equal to

Precedence 7

- & Bitwise AND

Precedence 6

- ^ Bitwise exclusive OR

Precedence 5

- | Bitwise inclusive OR

Precedence 4

- && Logical AND

Precedence 3

- || Logical OR

Precedence 2

- ? : Ternary conditional

Precedence 1

- = Assignment
- += Addition assignment
- -= Subtraction assignment
- *= Multiplication assignment
- /= Division assignment
- %= Modulus assignment

IF-Statement

if statement

An if statement contains a Boolean expression and block of statements enclosed within braces.

if(conditional expression)

//statement or compound statement;
else
//optional
//statement or compound statement;
//optional
If the Boolean expression is true then statement block is executed otherwise (if false) program directly goes to next statement without executing Statement block.

if....else

If statement block with else statement is known as as if...else statement. Else portion is non-compulsory.

```
if ( condition_one ){
//statements
}
else if ( condition_two )
{
//statements
}
else
{
//statements
}
```

If the condition is true, then compiler will execute the if block of statements, if false then else block of statements will be executed.

nested if...else

when a series of decisions are involved, we may have to use more than one if...else statement in nested form as follows:

```
if(test condition1)
{
if(test condition2)
{
//statement1;
}
else
{
//statement2;
}
}
else
{
//statement3;
}
//statement x;
```

Switch Statement

A switch statement is used instead of nested if...else statements. It is multiple branch decision statement.
A switch statement tests a variable with list of values for equivalence. Each value is called a case. The case value must be a constant i.

SYNTAX

```
switch(expression)
{
case constant:
//sequence of optional statements
break; //optional
case constant:
//sequence of optional statements
break; //optional
.
.
.
default : //optional
//sequence of optional statements
}
```

Individual case keyword and a semi-colon (:) is used for each constant. Switch tool is used for skipping to particular case, after jumping to that case it will execute all statements from cases beneath that case this is called as ''Fall Through''.

In the example below, for example, if the value 2 is entered, then the program will print two one something else!

```
switch(i)
{
case 4:
System.out.println("four");
break;
case 3:
System.out.println("three");
break;
case 2:
System.out.println("two");
case 1:
System.out.println("one");
default:
System.out.println("something else!");
}
```

To avoid fall through, the break statements are necessary to exit the switch. If value 4 is entered, then in case 4 it will just print four and ends the switch.

The default label is non-compulsory, It is used for cases that are not present.

WHILE Loop

Java while loop is used to execute statement(s) until a condition holds true.

SYNTAX

```
while (condition(s))
{
// statements
}
```

If the condition holds true then the body of loop is executed, after execution of loop body condition is tested again and if the condition is true then body of loop is executed again and the process repeats until condition becomes false.

1. Condition is always evaluated to true or false and if it is a constant, For example while (c) { …} where c is a constant then any non zero value of c is considered true and zero is considered false.
2. You can test multiple conditions such as

   ```
   while ( a > b && c != 0)
   {
   // statements
   }
   ```

   Loop body is executed till value of a is greater than value of b and c is not equal to zero.

3. Body of loop can contain more than one statement.

   For multiple statements you need to place them in a block using {} and if body of loop contain only single statement you can optionally use {}. It is recommended to use braces always to make your program easily readable and understandable.

DO-WHILE Loop

The while loop makes a test condition before the loop is executed.Therefore, the body of the loop may not be executed at all if the condition is not satisfied at the very first attempt.

On some occasions it might be necessary to execute the body of the loop before the test is performed.
Such situations can be handled with the help of the do statement.

SYNTAX

```
do
{
//statement(s)...
} while (condition);
```

On reaching the do statement, program evaluate the body of loop first.
At the end of the loop, the condition in the while statement is evaluated. If the condition is true, the program continues to evaluate the body of loop again and again till condition becomes false.

```
int i =0;
do
{
System.out.println("i is : " + i);
i++;
}while(i < 5);
```

Output would be i is : 0

```
i is : 1
i is : 2
i is : 3
i is : 4
```

FOR Loop

Java for loop used to repeat execution of statement(s) until a certain condition holds true.
The general form of the for statement can be expressed as follows:

```
for (initialization; termination; increment)
{
statement(s)...
}
```

You can initialize multiple variables, test many conditions and perform increments or decrements on many variables according to requirement. All three components of for loop are optional.

For example, to execute a statement 5 times:

```
for (i = 0; i < 5; i++)
statements...;
```

Another way of doing this is: i = 4;

```
while (i>=0)
statements...;
```

Basics of Classes

Classes

Java is a true object-oriented language and therefore the underlying structure of all Java programs is classes.
A class is nothing but a blueprint or a template for creating different objects which defines its properties and behaviours. Java class objects exhibit the properties and behaviours defined by its class. A class can contain fields and methods to describe the behaviour of an object.

Defining a class

Once the class type has been defined, we can create ''variables'' of that type using declaration that are similar to the basic type declarations. In Java, these variables are termed as instances of classes, which are the actual objects.
The basic form of a class definition is :

```
class classname [extends superclassname]
{
[ filelds declaration; ]
[ methods declaration; ]
}
```

Fields

A Java field is a variable inside a class. For instance, in a class representing an employee, the Employee class might contain the following fields:

- name
- position
- salary
- hiredDate

Field Declaration Syntax
A Java field is declared using the following syntax:

```
[access_modifier] [static] [final] type name [= initial value] ;
```

The square brackets [ ] around some of the keywords mean that this option is optional. Only type and name are required.

Methods

Methods are nothing but members of a class that provide a service for an object or perform some business logic. Java fields and member functions names are case sensitive. Current states of a class's corresponding object are stored in the object's instance variables. Methods define the operations that can be performed in Java programming.

Method Declaration Syntax

```
type methodname (parameter-list)
{
method body;
}
```

Below is an example showing the Objects and Classes of the Cube class that defines 3 fields namely length, breadth and height. Also the class contains a member function getVolume().

```
public class Cube
{
int length;
int breadth;
int height;
public int getVolume()
{
return (length * breadth * height);
}
}
```

Class Objects

As pointed out earlier, an object in java is essentially a block of memory that contains space to store all the instance variables. Creating an object is also referred to as instantiating an object.

In Java, the new keyword is used to create new objects.
There are three steps when creating an object from a class:
Declaration: A variable declaration with a variable name with an object type.
Instantiation: The 'new' key word is used to create the object.
Initialization: The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Example of creating an object is given below:
Rectangle rect;
rect = new Rectangle();

In the first statement we declared the object rect. it declares a variable to hold the object reference and in second we instantiated the object which actually assigns the object reference to the variable.
variable rect is now an object of the Rectangle class.

Accessing class members

Now that we have created objects, each containing its own set of variables, we should assign values to those variables in order to use them in our program.
Remember, all variables must be assigned values before they are used.
Since we are outside the class, we cannot access the instance variables and the methods directly. To do this, we must use the concerned object and the dot operator as shown below:
objectname.variablename = value; objectname.methodname(parameter-list);

Here objectname is the name of the object, variablename is the name of the instance variable inside the object that we wish to access, methodname is the method that we wish to call, and parameter-list is a comma separated list of "actual values" (or expressions) that must match in type and number with the Parameter list of the methodname declared in the class.

The instance variables & method of the Rectangle class may be accessed and assigned values as follows:

```
rect1.width=15
rect1.length=20;
rect2.width=24;
rect2.length=34;
rect1.area();
```

Constructors

Constructor in java is a special type of method that is used to initialize the object.
Java constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object therefore it is known as constructor.

Rules for creating java constructor

There are basically two rules defined for the constructor.

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

Types of java constructors

There are two types of constructors:

1. Default constructor (no-arg constructor)
2. Parameterized constructo

Java Default Constructor

A constructor that have no parameter is known as default constructor.
Syntax of default constructor:

```
class_name()
{
}
```

Example
In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
class Bike1
{
Bike1()
{
System.out.println("Bike is created");
}
public static void main(String args[])
{
Bike1 b=new Bike1();
}
}
```

Java parameterized constructor

A constructor that have parameters is known as parameterized constructor.

Why use parameterized constructor?
Parameterized constructor is used to provide different values to the distinct objects.

Example In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

```
class Student4
{
int id;
String name;
Student4(int i,String n)
{
id = i;
name = n;
}

void display()
{
System.out.println(id+" "+name);
}
public static void main(String args[])
{
Student4 s1 = new Student4(111,"Karan");
Student4 s2 = new Student4(222,"Aryan");
s1.display();
s2.display();
}
}
```

Method Overloading

If a class have multiple methods by same name but different parameters, it is known as Method Overloading.
If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behaviour of the method because its name differs.

So, we perform method overloading to figure out the program quickly.

Argument lists could differ in:

1. Number of parameters.
2. Data type of parameters.
3. Sequence of Data type of parameters.

Method overloading is also known as Static Polymorphism.

Points to Note:

1. Static Polymorphism is also known as compile time binding or early binding.
2. Static binding happens at compile time. Method overloading is an example of static binding where binding of method call to its definition happens at Compile time.

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

1. By changing number of arguments

```
class DisplayOverloading
{
public void disp(char c)
{
System.out.println(c);
}
public void disp(char c, int num)
{
System.out.println(c + " "+num);
}
}

class Sample
{
public static void main(String args[])
{
DisplayOverloading obj = new DisplayOverloading();
obj.disp('a');
obj.disp('a',10);
}
}
```

Output:

```
a
a 10
```

2. By changing the data type

```
class DisplayOverloading
{
```

```
public void disp(char c)
{
System.out.println(c);
}
public void disp(int c)
{
System.out.println(c );
}
}

class Sample2
{
public static void main(String args[])
{
DisplayOverloading obj = new DisplayOverloading();
obj.disp('a');
obj.disp(5);
}
}
```

Output:

```
a
5
```

Method Overridding

If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in java.
In other words, If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

Rules for method overriding:

1. The argument list should be exactly the same as that of the overridden method.
2. The return type should be the same or a subtype of the return type declared in the original overridden method in the superclass.
3. The access level cannot be more restrictive than the overridden method's access level.
4. Instance methods can be overridden only if they are inherited by the subclass.
5. A method declared final cannot be overridden.
6. A method declared static cannot be overridden but can be re-declared.
7. If a method cannot be inherited, then it cannot be overridden.
8. A subclass within the same package as the instance's superclass can override any superclass method that is not declared private or final.
9. A subclass in a different package can only override the non-final methods declared public or protected.
10. Constructors cannot be overridden.

Advantage:

The main advantage of method overriding is that the class can give its own specific implementation to a inherited method without even modifying the parent class(base class).

Example:

```
class Base
{
public void display()
{
System.out.println("this is base class");
}
}

class Child extends Base
{
public void display()
{
System.out.println("this is child class");
}
public static void main( String args[])
{
Child obj = new Child();
obj.display();
}
}
```

output

```
this is child class
```

Using the super keyword:

When invoking a base class version of an overridden method the super keyword is used.

```
class Base
{
public void display()
{
System.out.println("this is base class");
}
}

class Child extends Base
{
public void display()
{
super.display(); // invokes the super class method
System.out.println("this is child class");
}
public static void main( String args[])
{
Child obj = new Child();
obj.display();
}
}
```

output

```
this is base class
this is child class
```

Static Keyword

The static keyword in java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.

The static can be:

1. variable (also known as class variable)
2. method (also known as class method)
3. block
4. nested class

Variables and methods marked static belong to the class rather than to any particular instance of the class. These can be used without having any instances of that class at all. Only the class is sufficient to invoke a static method or access a static variable. A static variable is shared by all the instances of that class i.e only one copy of the static variable is maintained.

```
class Counter
{
static int animalCount=0;
public Counter()
{
count+=1;
}
public static void main(String[] args)
{
new Counter();
new Counter();
new Counter();
System.out.println("The Number of Animals is: "+count);
}
}
```

Output:

```
The Number of Animals is: 3.
```

A static method cannot access non-static/instance variables, because a static method is never associated with any instance. The same applies with the non-static methods as well, a static method can't directly invoke a non-static method. But static method can access non-static methods by means of declaring instances and using them.

Accessing static variables and methods

In case of instance methods and instance variables, instances of that class are used to access them.

```
objectReference.instanceVariable
objectReference.instanceMethod
```

```
class Counter
{
static int count=0;
public Counter()
{
count+=1;
}
public static int getCount()
{
return count;
}
}
class TestAnimal
{
public static void main(String[] args)
{
```

```
new Counter();
new Counter();
new Counter();
System.out.println("The Counter is: "+ Counter.getCount());
```

/* Notice the way in which the Static method is called using the class name followed by static method. */ } } Remember that static methods can't be overridden. They can be redefined in a subclass, but redifining and overriding aren't the same thing. Its called as Hiding.

Inheritance

Inheritance is one of the key features of Object Oriented Programming. Inheritance provided mechanism that allowed a class to inherit property of another class. When a Class extends another class it inherits all non-private members including fields and methods.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

Inheritance defines is-a relationship between a Super class and its Sub class. extends and implements keywords are used to describe inheritance in Java.

Why use inheritance in java

- For Method Overriding (so runtime polymorphism can be achieved).

  For Code Reusability.

Syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
//methods and fields
}
```

extends is the keyword used to inherit the properties of a class.

super keyword

In Java, super keyword is used to refer to immediate parent class of a class. In other words super keyword is used by a subclass whenever it need to refer to its immediate super class.

Types of Inheritance

Syntax of Java Inheritance

Single inheritance is damn easy to understand. When a class extends another one class only then we call it a single inheritance. The below example shows that class B extends only one class which is A. Here A is a parent class of B and B would be a child class of A.

```
class A
{

}
class B extends A
{

}
```

2) Multiple Inheritance

''Multiple Inheritance'' refers to the concept of one class extending (Or inherits) more than one base class. The inheritance we learnt earlier had the concept of one base class or parent. The problem with ''multiple inheritance'' is that the derived class will have to manage the dependency on two base classes.

Most of the new OO languages like Small Talk, Java, C# do not support Multiple inheritance. Multiple Inheritance is supported in C++.

```
class A
{

}
```

```
class B
{

}
class C extends A,B
{

}
```

3) Multilevel Inheritance

Multilevel inheritance refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class. As you can see in below example C is subclass or child class of B and B is a child class of A.

```
class A
{

}
class B extends A
{

}
class C extends B
{

}
```

4) Hierarchical Inheritance

In such kind of inheritance one class is inherited by many sub classes. In below example class B,C and D inherits the same class A. A is parent class (or base class) of B,C & D.

```
class A
{

}
class B extends A
{

}
class C extends A
{

}
class D extends A
{

}
```

5) Hybrid Inheritance

In simple terms you can say that Hybrid inheritance is a combination of Single and Multiple inheritance.

A hybrid inheritance can be achieved in the java in a same way as multiple inheritance can be!! Using interfaces. yes you heard it right. By using interfaces you can have multiple as well as hybrid inheritance in Java.

Final Keyword

The final keyword in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only.

1) Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

Example of final variable There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
class Game
{
final int speedlimit=90; //final variable
void run()
{
speedlimit=400;
}
public static void main(String args[])
{
```

```
Game obj=new Game();
obj.run();
}
}//end of class
```

Output: Compile Time Error

2) Java final method>

If you make any method as final, you cannot override it.

Example of final method

```
class Game
{
final void run()
{
System.out.println("bowling");
}
}
class Cricket extends Game
{
void run()
{
System.out.println("bowling safely with 100kmph");
}
public static void main(String args[])
{
Cricket cricket= new Cricket();
cricket.run();
}
}
```

Output: Compile Time Error

3) Java final class

If you make any class as final, you cannot extend it.

Example of final class

```
final class Game
{

}
class Cricket extends Game
{
void run()
{
System.out.println("bowling safely with 100kmph");
}
public static void main(String args[])
{
Cricket cricket= new Cricket();
cricket.run();
}
}
```

Output: Compile Time Error

Abstraction

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.
Abstraction lets you focus on what the object does instead of how it does it.

There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%)
2. Interface (100%)

Abstract class

A class that is declared as abstract is known as abstract class. It needs to be extended and its method implemented. It cannot be instantiated.

Example abstract class

```
abstract class A
{

}
```

Abstract method

A method that is declared as abstract and does not have implementation is known as abstract method.

Example abstract method

```
abstract void printStatus();
//no body and abstract
```

Arrays

Array is an object the contains elements of similar data type. It is a data structure where we store similar elements. We can store only fixed set of elements in a java array.
Array in java is index based, first element of the array is stored at 0 index.

Types of Array in java

1. Single Dimensional Array
2. Multidimensional Array

1. Single Dimensional Array

Declaring Array Variables:
To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference.
Here is the syntax for declaring an array variable:
dataType[] arrayName;
or
dataType arrayName[];

Instantiating Arrays:
You can instantiate an array by using the new operator with the following syntax:
arrayName = new dataType[arraySize];
The above statement does two things:

1. It creates an array using new dataType[arraySize];
2. It assigns the reference of the newly created array to the variable arrayName.

2. Multidimensional Array

Syntax to Declare Multidimensional Array in java

```
dataType[][] arrayName; or
dataType arrayName[][];
```

Example to instantiate Multidimensional Array

```
int[][] arr=new int[2][3];
//2 row and 3 column
```

Example to initialize Multidimensional Array in java

```
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
arr[1][0]=4;
arr[1][1]=5;
arr[1][2]=6;
```

Passing Arrays to Methods:

Just as you can pass primitive type values to methods, you can also pass arrays to methods. For example, the following method displays the elements in an int array:

```
public static void display(int[] arr)
{
for (int i = 0; i < arr.length; i++)
{
System.out.print(arr[i] + " ");
}
}
```

Returning an Array from a Method:

A method may also return an array.

For example, the method shown below returns an array that is the copy of another array:

```
public static int[] copyarray(int[] list)
{
int[] result = new int[list.length];

for (int i = 0; i < list.length - 1 ; i++)
{
result[i] = list[i];
}
return result;
}
```

Arrays Methods :

Arrays.binarySearch(Object[] a, Object key)
Searches the specified array of Object (Byte, Int , double, etc.) for the specified value using the binary search algorithm. The array must be sorted prior to making this call. This returns index of the search key.

Arrays.equals(long[] a, long[] a2)
Returns true if the two specified arrays of longs are equal to one another. Two arrays are considered equal if both arrays contain the same number of elements, and all corresponding pairs of elements in the two arrays are equal.

Arrays.fill(int[] a, int val)
Assigns the specified int value to each element of the specified array of ints. Same method could be used by all other primitive data types (Byte, short, Int etc.)

Arrays.sort(Object[] a)
Sorts the specified array of objects into ascending order, according to the natural ordering of its elements. Same method could be used by all other primitive data types (Byte, short, Int, etc.)

String

In java, string is basically an object that represents sequence of char values. The Java platform provides the String class to create and manipulate strings.

Creating Strings:

1) The most direct way to create a string is to write:

```
String str1 = "Hello Java!";
```

2) Using another String object

```
String str2 = new String(str1);
```

3) Using new Keyword

```
String str3 = new String("Java");
```

4) Using + operator (Concatenation)

```
String str4 = str1 + str2;
```

or,

```
String str5 = "hello"+"Java";
```

String Length:

length() method returns the number of characters contained in the string object.

```
String str1 = "Hello Java";

int len = str1.length();

System.out.println( "String Length is : " + len );
```

Concatenating Strings:

The String class includes a method for concatenating two strings:

string1.concat(string2);

This returns a new string that is string1 with string2 added to it at the end. You can also use the concat() method with string literals, as in:

"Hello ".concat("Java");

Strings are more commonly concatenated with the + operator, as in:

"Hello " + " Java" + "!"

which results in:

"Hello Java!"

String Methods :

1. char charAt(int index)

returns char value for the particular index

2. int length()

returns string length

3. static String format(String format, Object... args)

returns formatted string

4. static String format(Locale l, String format, Object... args)

returns formatted string with given locale

5. String substring(int beginIndex)

returns substring for given begin index

6. String substring(int beginIndex, int endIndex)

returns substring for given begin index and end index

7. boolean contains(CharSequence s)

returns true or false after matching the sequence of char value

8. static String join(CharSequence delimiter, CharSequence... elements)

returns a joined string

9. static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)

returns a joined string

10. boolean equals(Object another)

checks the equality of string with object

11. boolean isEmpty()

checks if string is empty

12. String concat(String str)

concatinates specified string

13. String replace(char old, char new)

replaces all occurrences of specified char value

14. String replace(CharSequence old, CharSequence new)

replaces all occurrences of specified CharSequence

15. String trim()

returns trimmed string omitting leading and trailing spaces

16. String split(String regex)

returns splitted string matching regex

17. String split(String regex, int limit)

returns splitted string matching regex and limit

18. String intern()

returns interned string

19. int indexOf(int ch)

returns specified char value index

20. int indexOf(int ch, int fromIndex)

returns specified char value index starting with given index

21. int indexOf(String substring)

returns specified substring index

22. int indexOf(String substring, int fromIndex)

returns specified substring index starting with given index

23. String toLowerCase()

returns string in lowercase.

24. String toLowerCase(Locale l)

returns string in lowercase using specified locale.

25. String toUpperCase()

returns string in uppercase.

26. String toUpperCase(Locale l)

returns string in uppercase using specified locale.