Unit-I

History of Java -

- James Gosling developed the 'java' in 1995
- project begin in 1991.
- The small team of sun engineers called Green Team.
- Sun Microsystem Inc. developed java.
- Firstly, it was called "Greentalk" & extension .gt
- After that it was called oak.

features of Java -

① Simple -
Java is very easy to learn & its syntax is clear, simple & easy to understand.
As to sun microsystems
- Syntax is based on C++
- Java has removed many complicated & rarely used features like explicit pointers, operator overloading.

② Platform Independent
Platform is not bound.
Two types of platform
- Software based
- Hardware based
Java is the based platform

Java code be created on multiple platforms

③ Robust -
- It uses strong memory management
- There is a lack of pointer that avoids security problem.
- Java provides automatic garbage collection which runs on the JVM to get rid of unreferenced obj.

④ Portable -
Because it facilitates you to carry the java bytecode to any platform. It doesn't require any implementation.

⑤ High performance -
- Java is faster than other traditional interpreting programming language b/c java bytecode is "close" to native code.
- It is little bit slower than the compiled language i.e.

⑥ Distributed -
It facilitates users to create distributed application in java. This feature make us able to access by calling the method from any machine on the Internet.

⑦ Multi-Threaded -
We can write java programs that deal with many tasks at once by defining multiple threads.
Main Advantage
It doesn't occupy memory for each thread.
It shares a common memory area.

(8) Dynamic -

→ Java is a dynamic language
→ It support dynamic loading of class means classes are loaded on demand
→ It also ~~provides~~ support function from its native languages like C, C++.

Java support dynamic compilation & automatic garbage collection.

C++ Vs Java

(1) Platform Independent - C++ is platform-dependent
Java is platform-independent

(2) Mainly used for - C++ is mainly used for system programming
Java is mainly used for application programming. Widely used in window based, web based & mobile applications.

(3) Goto - C++ support goto statement
Java doesn't support

(4) Multiple Inheritance - C++ support
Java doesn't support only achieved by using interfaces

(5) Operator overloading - C++ support while java doesn't

(6) Pointers - C++ support pointers & you can write a pointer program in C++.
Java support pointer internally. You can't write pointer program in java.

(7)

Call by value & Call by reference — C++ support both
Java support only call by value

(8) Structure & union — C++ support
Java doesn't

(9) Thread support — C++ does not have built-in support
∴ Java support built-in support

(10) Documentation Comment — C++ doesn't
Java support

(11) Hardware — C++ nearer to hardware
Java is not interactive with hardware

(12) Java doesn't default argument like C++

(13) Java does not support header files like C++.
∴ import keywords to include different class & method

syntax

```
#include<iostream>
using namespace std;
int main(){
cout<<"Hello";
return 0;
}
```

Output
C++

```
class simple{
public static void main(){
System.out.println("Hello");
}
}
```

output
Java

Page No.
Date

Java Architecture :-

① JVM

→ Java Virtual Machine is a abstract machine
→ It is called a virtual machine b/c it doesn't physically exist.

→ It is a specification that provides to runtime environment in which java bytecode can be executed.
→ It can run those programs which are written in other languages & compile to java bytecode.

→ It has Three notion of the JVM
specification, Implementation & instances

→ JVM perform main task.
- loads code
- verifies
- Execute
- provide runtime environment

② JRE-

→ Stands for Java Runtime environment
→ It is set of the tools which are used for developing the java application.
→ It is used to provide runtime environment
→ It is the implementation of JVM
→ Its physically exist
→ It contain set of libraries + other files that JVM uses at runtime.

(iii) JDK -

→ Java Development Kit.
→ It is a Java development environment which is used to java application & applet
→ It physically exist
→ It contain JRE + development tools

JDK contain private JVM & few other resources such as interpreter / loader (java),
a compiler (javac)
an archiver (jar)
a documentation generator (javadoc) etc.
to complete the development of java applets.

Java Variables -

A variable is a name of a reserved area allocated in memory

Eg - int data = 50

Types of variables -

(1) Local Variable -

A variable declare inside the body of the method is called local variable.
A local variable cannot be defined "static" keyword.

(2) Instance variable -

A variable declared inside the class but outside the body of the method is called instance variable.
It is not declared with 'static' keyword

Page No.
Date

③ Static Variable -

→ A variable that is declared as static is called static variables.

→ It cannot be local

→ You can create single copy of the static variable & share it among all the instances of the class.

→ Memory allocation for static variables happens only at one when the class is loaded in the memory.

Eg- public class A {
    static int m = 100; // static Variables
    void method() {
        int n = 90; // local variables
    }
    public static void main (String args[]) {
        int data = 100; // Instance variables
    }
}

Eg- Type casting

public class A {
    public static void main (String args[]) {
        float f = 10.5;
        int a = int(f);
        System.out.println(f);
        System.out.println(a);
    }
}

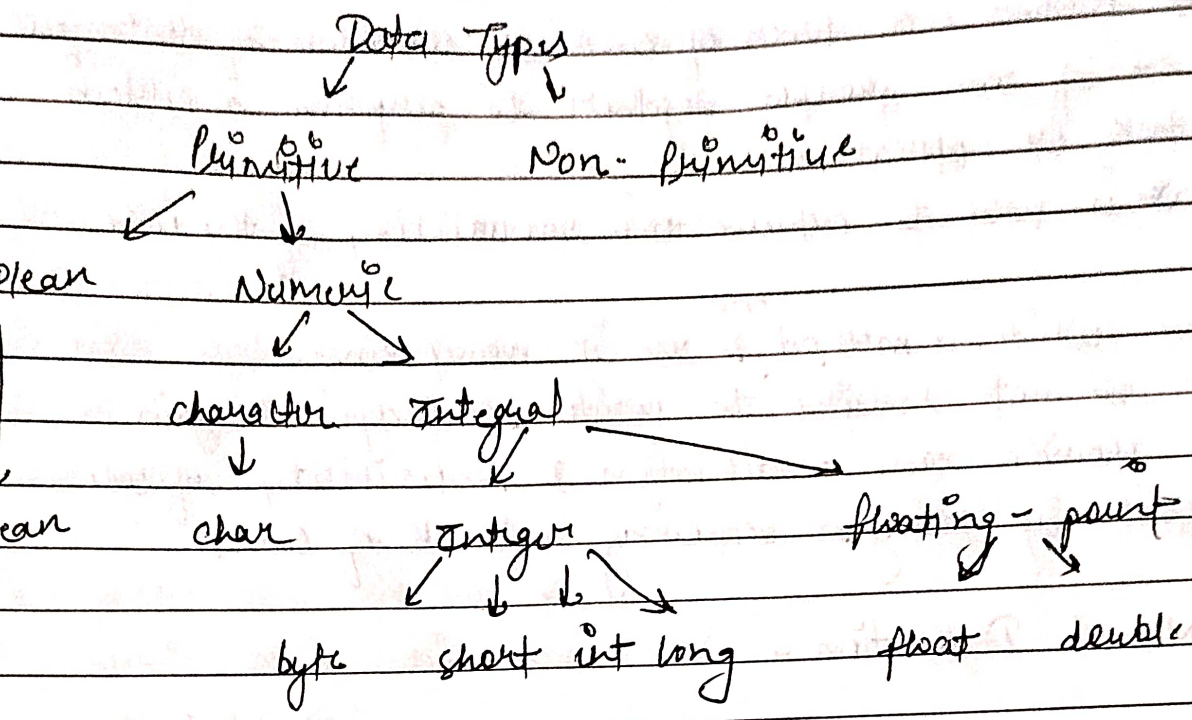
output

10.5

10

Page No.
Date

Data Types



Page No.
Date

Introducing classes -

Method in Java -

→ A method is a block of code or collection of statement or a set of code grouped together to perform a certain task or operation.

→ It is used to achieve the reusability of the code.

→ We write a method once & use it many times.

→ We do not require to write code again & again.

→ It provide easy modification & readability of code, just by adding or removing a chunk of code.

Method Declaration -

```
public   int    sum   (int a, int b)   ← Method Header
  ↓       ↓      ↓        ↓
Access  return  Method   Parameter
specifier type  name     list
```

Types of Method -

(1) Predefined Method -

→ In java, predefined method are the method that is already defined in the java class libraries is known as

→ It is also known as standard library method or built-in method

→ Eg. length(), equal(), compareTo(), sqrt().

Eg- public class Demo {
public static void main (String args[]) {
System.out.println("The max. num is " + Math.max(9,7));
}
}

Output = The max. num is 9

② user-defined method -
A method written by the user or programmer is called user-defined method.

Eg- public class Addition {
public static void main (String args[]) {
int a = 19;
int b = 5;
// method calling
int c = add(a,b); // a & b are actual parameters
System.out.println("The sum of a & b" + c);
}
// user-defined method
public static int add (int n1, int n2) { // formal parameters
int s;
s = n1 + n2;
return s; // returning the sum.
}
}

output -
The sum of a & b is = 24,

Page No.
Date

Constructor in Java -

→ In java, a constructor is a block of code similar to the method.

→ It is called when the instance of the class is created.

→ At the time of calling constructor, memory for the object is allocated.

→ It is a special type of method which is used to initialize the object.

Rules -

1. Constructor name must be the same as class name.
2. A constructor have no explicit return type.
3. A java constructor cannot be abstract, static, final & synchronized.

Types of constructor

① Default Constructor (no- arg constructor) -
A constructor is called default constructor if it does not any parameter,

Syntax - <class_name> () {}

Eg -

```
class Bike1 {
  // creating a default constructor
  Bike1 () {
  System.out.println ("Bike is created");
  }

public static void main (String args[]) {
    // calling a constructor
    Bike1 b = new Bike1 ();
  }
}
```

Output is = Bike is created

Eg- of default constructor of displaying the default values

```
class Student3 {
  int id;
  String name;

void display () {
System.out.println (id + " " + name);
  }

public static void main (String args[]) {
  Student3 s1 = new Student3 ();
  s2 = ;

s1.display ();
s2.display ();
  }
}
```

output
0 null
0 null

② Parameterized Constructor

A constructor which has a specific number of parameters is called parameterized constructor.

→ It is used to provide distinct value to distinct object.

Eg- class Student4 {
int id;
string name;
// creating a parameterized constructor
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");
S2 = new Student4 (112, "John");

S1.display ();
S2.display ();
}
}

Output 111 Karan
222 Aryan

Page No.
Date

Java static Keyword -

The static keyword in java used for memory management mainly.

① Java static Variable -

→ If you declare any variable as static, it is known as static variables

→ The static variable can be used to refer to the common property of all objects.

→ Memory allocation for the static variables happens only once at the time of class loading.

Advantage - memory efficient (it saves memory)

Eg:-
```
class student {
    int rollNo;
    String name;
    static String college = "ITS";

    Student (int r, String n) {
    rollno = r;
    name = n;
    }

void display () {
System.out.println("Roll No " + rollno + " " + name + " " + college);
}
```

public class Test_static_variable {

public static void main (String args[]) {

Student s1 = new Student("Karan");
Student s2 = new Student("Aryan");

s1.display();
s2.display();
}
}

Output
Karan ITS
Aryan ITS

② Java static Method-
If you apply static keyword with any method, it is known as static keyword

① A static method can access static data member & can change the value of it.
② A static method can be invoked without creating the instance of the class
③ A static method belong to the class rather than the object of the class.

eg- class Student {
int rollno;
String name;
static String college = "ITS";

static void change() {
college = "BBIT";
}

```
student (int r, String n) {
    rollno. = r;
name String = n;
    }

void display () {
System.out.println ("rollno." + " " + name + " " + college);
    }

public class Test static variable1 {
    public static void main ( ) {
    Student s1 = new 'student ( '111', 'Karan');
    Student s2 = ——— ——— (222, 'Aryan');
    ——— s3 = ——— ——— (333, 'Sonoo');

    s1. display ();
    s2. display ();
    s3. display ();
    }
}
```

Output -

111	Karan	BB-DIT
222	Aryan	———
333	Sonoo	———

③ Java static block -

① It is used to initialize the static data member.
② It is executed before the main method at the time of class loading

```
Eg- class A2 {
    static {
    System.out.println ("static block is");
    public static void main (String args[]) {
    System.out.println ("Hello main");
    }
}
```

The most common uses:,
to eliminate confusion b/w attributes & parameters with the same name.

Page No.

This Keyword
In Java, this keyword is used to refer the current object inside a method or constructor

Eg - this → State / Behaviour of object

Usage of Java this Keyword

(1) this can be used refer current class instance variable
(2) to invoked current class method
(3) To invoked current class constructor
(4) this() can be passed as an argument in the method call
(5) as an argument in the constructor call
(6) to return the current class instance from the method

Eg (1) -
```
class Student {
    int rollno;
    String name;
    float fee;

    Student (int rollno, String name, float fee) {
        this.rollno = rollno;
        this.name = name;
        this.fee = fee
    }

    void display() {
        System.out.println(rollno + " " + name + " " + fee);
    }
}
```

or

```
rollno = rollno
name = name
fee = fee
```

```
class TestThis2 {
  public static void main(String args[]) {
    Student s1 = new Student(111, "ankit", 5000f);
    Student s2 = new Student(112, "sumit", 6000f);
    s1.display();
    s2.display();
  }
}
```

Output
111 ankit 5000.0
112 sumit 6000.0

A - data box
Output
0 null 0.0 } formal arguments
0 null 0.0 } is same as instance variable
And this keyword to distinguish them

Eg ② class A {

① void m {
System.out.println("hello m");
}

② void n() {
this.m();
}
}

```
class TestThis4 {
  public static void main(String[] args) {
    A a = new A();
    a.n();
  }
}
```

Output
hello n
hello m

Java final Keyword -

In Java, final keyword is used to denote constant. It can be used with method, variable, class.

Once any entity (variable, method or class) is declared as final, it can be assigned only once. That is,

- final variable cannot be reinitialized with another value
- final method cannot be overridden
- final class cannot be extended

① Java final variable

```
class Bike9 {
    final int speedlimit = 90;
    void run() {
        speedlimit = 100;
    }

    public static void main (String args[]) {
        Bike9 obj = new Bike9();
        obj.run()
    }
}// end of the class
```

output - Compile Time error

Page No.
Date

② Java final method -

```
class Bike {
final void run() {
System.out.println("running");
}

class Honda extend Bike {
void run() {
system.out.println("running safely with 100Kmph");
}

public static void main (String arg[]) {
Honda honda = new Honda();
honda.run();
}
}
```

output - Compile Time error

③ Java final class

```
final class Bike {}

class Honda1 extends Bike {
void run() {
System.out.println("running ---");
public static void main { } {
Honda1 honda = new Honda1;
honda.run();
}
}
```

output : Compile Time error

Can we initialize blank final variable?
Yes, but only in constructor

→ Java Method overloading -

If a class have multiple methods same name but different in parameter, it is known as method overloading.

Advantage - It increases the readability of the program

Different way to overload the method -
① By changing the no. of argument
② By changing the data type

① By changing the no. of argument -

```
class Adder {
  static int add(int a, int b) {
      return a+b;
  }
  static int add(int a, int b, int c) {
      return a+b+c;
  }
}

class overloading {
  public void main() {
  System.out.println(Adder.add(1, 2));
  (Adder.add(4, 9, 6));
  }
}
```

output - 3
15

Page No.
Date

(2) Method Overloading : ~~No. of~~ changing data type of argument.

```
class Adder {
static int add (int a, int b) {
        return a+b;
static int add (double a, double b) {
        return a+b;
        }

class Overloading {
p.s.v.m ( ) {
System.out.println (Adder.add (11,11));
                   (Adder.add (12.3, 12.6));
}}
```

output

22
24.9

Java Garbage Collection -

In Java, garbage means unreferenced object

Garbage collection is a process of reclaiming the runtime unused memory automatically. It means, to destroy the unused objects.

To do so, we were using free() function in C language delete() in C++.

But in Java, it is performed automatically.

So Java provides better memory management.

Advantage -

① It makes Java memory efficient because garbage collector remove unreferenced object from heap memory

② It is automatically done by the garbage collector (a part of JVM) so we don't to make extra efforts.

How to unreferenced the object -

1. By nulling a reference

```
Employee e = new Employee();
e = null;
```

2. By assigning reference to another:

```
Employee e1 = new Employee();
Employee e2 = new Employee();
e1 = e2;
```

(3) By anonymous object
new Employee();

finalize() method -
→ The finalize() method is invoked each time before the object is garbage collected.
→ This method can be used to perform clean up processing.
→ This method is defined in object class as;

protected void finalize() {}

gc() method -
The gc() method is used to invoked the garbage collector to perform clean up processing.
The gc() method is found in system & runtime classes.

public static void gc() {}

eg - public class TestGarbage1 {
public void finalize() {
System.out.println("object is garbage collected");
}
public static void main(String args[]) {
TestGarbage1 s1 = new TestGarbage1();
TestGarbage2 s2 = new TestGarbage1();
s1 = null;
s2 = null;
System.gc();
}
}

output -
object is garbage collected
object is garbage collected

Java wrapper class -

The wrapper class in java are used to convert primitive data types (int, char, float etc) into corresponding object

OR

A way to use primitive type as object

Primitive Data Type	Wrapper class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

Creating wrapper class -

```
public class Main {
    public static void main (String[] args) {
        Integer myInt = 5;
        Double myDouble = 5.99;
        Character myChar = 'A';
        System.out.println (myInt);
                           (myDouble);
                           (myChar);
    }
}
```

Page No.
Date

(2)

```
System.out.println(myInt.intValue());
                   (myDouble.doubleValue());
                   (myChar.charValue());
}
```

Same output

(3) Another useful method is the toString()

```
public class Main {
    public static void main() {
        Integer myInt = 100;
        String myString = myInt.toString();
        System.out.println(myString.length());
    }
}
```

Output = 3

Java I/O Input stream -

Java I/O is used to process the input & produce the output.

Java uses the concept of stream to make I/O operation fast.

java.io package contains all the classes required for input & output operations.

Stream -

→ A stream is a sequence of data

→ In java, a stream is composed of bytes.

In java, 3 stream are created automatically for us

① System.out : standard output stream
② System.in : input
③ System.err : error

Java FileOutputStream class -

→ Java FileOutputStream is an output stream used for writing data to a file.

→ You can write byte-oriented as well as character-oriented data through FileOutputStream class.

→ for character-oriented data, it is preferred to use FileWriter

→

Page No.
Date

Java InputStream class -

The InputStream class of java.io package is a abstract class that represent an input stream of byte

Subclass of InputStream -
- FileInputStream
- ByteArrayInputStream
- ObjectInputStream

Create an InputStream -

first, we must import [java.io.InputStream] package

```
// creates an InputStream
InputStream object1 = new FileInputStream();
```

→ We have created input stream using FileInputStream.

Method -
1. read() - read one byte of data
2. close() -

Create a file -

```
import java.io.*;
class createfile {
    public static void main(String args[]) {
        File f = new File("C:\Users\Knowoll\Desktop\c.txt");
        try {
            if (f.createNewFile()) {
                System.out.println("File successfully created");
            }
            else {
                System.out.println("File Already exist");
            }
        }
        catch (IOException i) {
            System.out.println("Exception handled");
        }
    }
}
```

To display the information -

```
import java.io.File;
class FileDemo
    {
public static void main(          ) {
        File f = new File("                    ");
        if (f.exists()) {
System.out.println("File name:" + f.getName());
                   ("File location:" + f.getAbsolutePath());
                   ("File writeable:" + f.canWrite());
```

```
        ("file Readable : " + f.canRead());
        ("file Size : " + f.length());
    }
    else {
    System.out.println("Doesn't exist");
    }
    }
    }
```

(3) - To write into a file -

```
import java.io.*;
class FileWriter {
    public static void main (    ) {
        try {
FileWriter f = new FileWriter("                    ");
        try {
            f.write("Java is a programming language");
            }
            finally {
                f.close();
            }
System.out.println("Successfully Data wrote");
    }
    catch (IOException i) {
        System.out.println(i);
    }
}
```

(i (r.read()) != -

④ Read a text from file -

```
import java.io.*;
class file reader {
public static void main (          ) {
try {
file Reader r = new file Reader ("                    ");
try {
int i;
while ((i = r.read()) != -1) {
System.out.print ((char) i));
}
}
finally {
r.close ();
system.out.print ("file closed");
}
}
catch (IOException e) {
System.out.println ("exception handled ..!");
}
}
}
```

⑤ Rename a file -

```
import java.io.file;
class rename file {
p. s. v. M      () {

file f = new file ("c:\\users\\        \\Desktop\\LC.txt");
file r = new file ("                    \\DT.txt");
```

```
if (f.exists) {
    System.out.println(f.getName() + " is a file");
}
else {
    System.out.println("Doesn't exist");
}
}
```

Byte oriented -

Java FileOutputStream -

```
import java.io.FileOutputStream;
public class FileOutputStreamExample {
    public static void main(String[] args) {
        try {
            FileOutputStream fout = new FileOutputStream("D:\\testout.txt");
            fout.write(65);
            fout.close();
            System.out.println("success");
        }
        catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Output

Success...

Content is testout.txt

A

Page No.
Date

Multithreading -

Multithreading in java is a process of executing multiple threads simultaneously.

A thread is a lightweight sub process, the smallest unit of processing.

However, multithreading uses than multiprocessing because threads uses a shared memory area.
They don't allocate separate memory area so saves memory & context-switching b/w threads takes less time than process.

Java Multithreading is mostly used in games, animation etc.

Advantages -

(1) It doesn't block the user because threads are independent & you can perform multiple operation at the same time.
(2) You can perform many operation together, so it save time
(3) Threads are independent so it does not affect the other threads if an exception occur.

Life Cycle of Thread -

(1) New - A new thread begins its life cycle in the new state. It remains in this state until program start the thread. It is also referred as born thread.

② Runnable - After a newly born thread is started, a thread becomes runnable.
A thread in this state is considered to be executing its task.

③ Waiting - Sometime, a thread transition to the waiting state. while a thread waits for other thread to perform a task.
A thread transition back to the runnable state only when another thread signal to the waiting thread to continue executing.

④ Timed waiting - A runnable thread can enter the timed waiting state for a specified interval of time.
A thread in this state transition back to runnable state when that time interval expires or when the event is waiting for occurs.

⑤ Terminated (Dead) - A runnable thread enter the terminated state when it complete its task.

Java threads -
① By extending Thread class.
② By implementing Runnable Interface -

Thread class -
Thread class provide constructor & method to create & perform operation on a thread.
Thread class extends object class & implement runnable interface

Commonly used constructor of Thread class:

Thread()
Thread(String name)
Thread(Runnable r)
Thread(Runnable r, String name)

Runnable Interface -

→ It should be executed by any class whose instances are intended to be executed by a thread.
→ It uses only one method named run()

① By extending class

file name - Multi.java

```
class Multi extends Thread{
public void run(){
System.out.println("Thread is Running");
}
public static void main(){
Multi t1 = new Multi();
t1.start();
}
}
```

Output
Thread is running

② By implementing Runnable Interface -

```
class Multi3 implements Runnable{
public void run(){
System.out.println("Thread is Running");
}
public static void run(){
Multi3 m1 = new Multi3();
Thread t1 = new Thread(m1); // using the constructor
t1.start();
}
```

(2) Using the Thread class : Thread (String name)

file name : MyThread1.java

```
public class MyThread1 {
    // Main Method
    p.s.v.m ( ) {
// creating an obj. of the Thread class using the constructor thread
Thread t = new Thread ("My first thread");
    t.start();

// getting the thread name by invoking the getName() method
    String str = t.getName();
    System.out.println(str);
    }
}
```

output -
My first Thread

(3) Using the thread class : Thread (Runnable r, String name)

```
public class Threads2 implement Runnable {
    public void run() {
System.out.println("Now the thread is running");
}

// main method
p.s.v.m ( ) {
    Runnable r1 = new MyThread2();
Thread th1 = new Thread (r1, "My new Thread");
    th1.start();
    String str = th1.getName();
    System.out.println(str);
}
}
```

My new thread
Now the thread is running