Unit - I Introduction to Java Programing language

* Introduction to Java & History

- Java is an Object - oriented programming language and a development platform, its widely used and the syntax is basically inspired from C and C++

- It was developed by sun micro system in 1991 by James Gosling its name was Oak. In 1995 its name has been changed to Java.

* Java Features

1. Simple
2. Object Oriented
3. Distributed
4. Platform independent
5. Architecture neutral
6. Multithreaded
7. Dynamic
8. Portable
9. High Performance
10. Robust
11. Secure
12. Interpreted

Prajwal Yadav

* Java Application

1. Mobile App development
2. Desktop GUI Applications
3. Web - based Applications
4. Gaming Applications
5. Big Data Technologies
6. Distributed Applications
7. Cloud - based Applications
8. IOT Applications.

* Java Components

1. Java Virtual Machine (JVM)

- All language compilers translates source code into machine code for a specific computer.

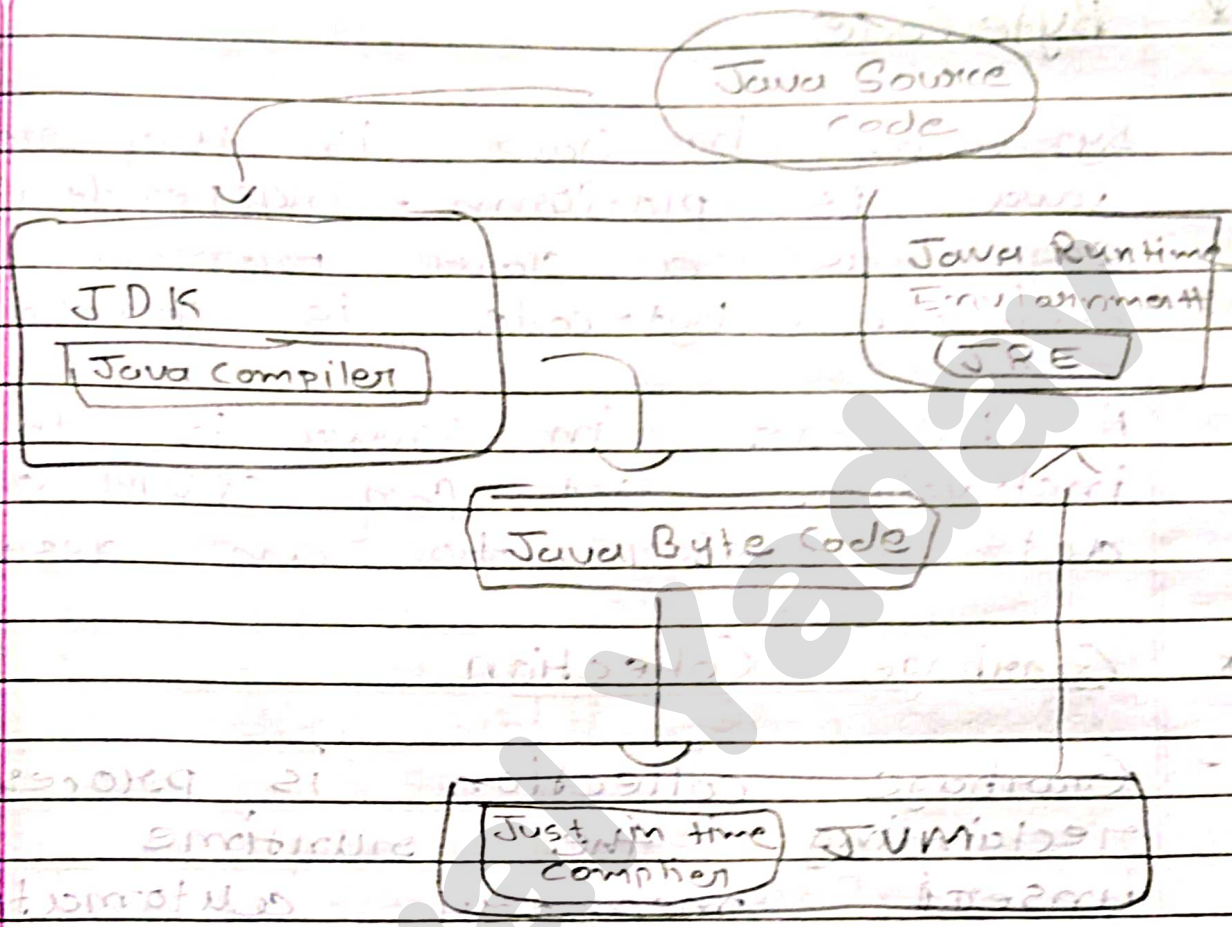
- Java is platform independent or machine neutral. Java compiler produces an intermediate code (byte code) for machine that does not exist.

- JVM exists only inside the computer memory. JVM simulated computer within the computer and does all major function of real computer

Java Program → Java Compiler → Virtual Machine

Virtual Machine → Interpreter → Real Machine

Prajwal Yadav



(2) Java Runtime Envioronment (JRE)

- JRE execute Java Program
- Java Virtual Machine is important part of the JRE, which actually runs the Programs. uses the Java class libraries and the run time libraries to executed those programs.

(3) Java development Ket (JDK)

- Java development Kit is bundle of software components which provides the enviornment to develop and execute the Java Program

Prajwal Yadav

* Byte Code

Bytecode in Java is the reason java is platform - independent, as soon as a Java program is compiled bytecode is generated.

A bytecode in Java is the instruction set for JVM and acts similar to an assembler

* Garbage Collection

- Garbage collection is process of reclaiming the runtime unsed memory automatically

- It is a way to destroy unused Objects.

- We may request JVM for garbage collection by calling System.gc() method.

Prajwal Yadav

Example

```
Class Test
{
    public static void main(String[] args)
    {
        Test t = new Test();
        t = null;
        System.gc();
    }

    public void finalize()
    {
        System.out.println("Garbage collected");
    }
}
```

Output

Garbage collected

Prajwal Yadav

* Structure of Java

1. Documentation Section
2. Package declaration
3. Import statement
4. Interface Statement
5. Class Definition
6. Main Method Class

(1) Documentation Section :- This section allowed to include comments to tell the program's purpose, it improves readability of the program

(2) Package Statement :- A Package Statement allowed to includes statement that provides a package declaration.

(3) Import Statement :-

- Using "import" statement we can use external libraries in our program

- the package contains the many predefined classes and interface. If we want to use any class of a particular package we need to imarpt that clas

Prajwal Yadav

(4) interface statement :- It is an Optional section. we can create an interface in this section if required. we use the interface keyword to create an interface.

- An interface is a slightly different from the class it contains only constants and method declaration.

(5) Class definition :- This section allowed to User able to declared other user defined class above the main method class

(6) Main Method Definition - inside class
Java Program Execution starts from main method. it consists main method definition inside class.

Prajwal Yadav

* Data type in Java

Java supports two type of data type
(1) Primitive Data type
(2) Non-primitive (Reference/object) Data type

(1) Primitive Data type

⇒ **INTEGER** :- Integers group includes byte, short, int, long.

(a) **byte** :- byte is a signed data type of size 8 bits.

- It has a range -128 to 127
- This is smallest integer type

(b) **short** :- short is a signed data type of size 16 bits

- It has range -32,768 to 32,767

(c) **int** :- int is a signed data type of size 32 bits

(d) **long** :- long is a signed data type of size 64 bits

Prajwal Yadav

⇨ **Floating** :- Floating point number group includes float, double

ⓐ **float** :- float is an size 32 bits.

Example :- float f1 = 3.14f;

ⓑ **double** :- double is an of size 64 bits

Example :- double d1 = 3.6;

⇨ **Character**

ⓐ **char** :- char is use to store character size Data type.

- Char data type size is 16 bits

- Example char c1 = 'A';

⇨ **Boolean** :- Boolean is an size of 1 bit. Boolean is a simple type called for logical values.

- Example :- Boolean b1 = true;

Pralyat Yadav

② Non - primitive Data type

ⓐ Class type
ⓑ Interface Type
ⓒ Array Type

* Type Conversion

There is two type - conversion in Java:

① Implicit Type casting
② Explicit Type casting

① Implicit Type casting

→ Convert by System automatically.

Example

```
Class Demo {
    public static void main (String [] args)
    {
        int a = 5 ;
        float c = 10.5f;
    float b = a + c;
        system.out.print (b);
    }
}
```

output : - 15.5

Prajwal Yadav

(2) Explicit Type casting

- Convert by user itself
- Example

```
class Demo
{
  Public static void main (string [] args)
  {
      int a = 10, b = 4;
      float c = (float) a / b;
      System.out.print (c);
  }
}
```

Output 2.5

* Widening

- Converting a lower datatype to a higher datatype is know widening.
- casting / conversion is done automatically

byte → short → int → long → float → double

```
Class Hello
{
  public static void main (String [] args)
  {
      Char ch = 'C';
      int i = ch;
  }  System.out.print (i);
}
```

Output
67

Prajwal Yadav

* Narrowing

- Converting higher datatype to lower datatype
- casting is not done automatically convert explicitly using cast operator " ()" explicitly.

double -> float -> long -> int -> short -> byte

* Scope of variables.

1. Instance Variable
2. Class / static Variable
3. Local Variable

Example

```
Class Demo
{
  ~~public static void main(String[]Args~~
  {

    String Name = "Anand"; //instance

    Static double height = 5.9; //static/cls
    public static void main
                (String []Args)
    {
        int mark = 72; //Local
    }
}
```

Prajwal Yadav

Array

- Array is a collection of similar type of data.

Type of array

1. Single Dimensional Array
2. Multidimensional Array

1. Single Dimensional Array (1D Array)

Syntax

Datatype [] arrayname;

Ex

```
int [] a = {20, 3, 5, 11, 12};
```

(or)

```
int [] a = new int [5];
```

a[0] = 20
a[1] = 3
a[2] = 5
a[3] = 11
a[4] = 12

Prajwal Yadav

* find the maximum of 10 number

```
Class maximum
{
    public static void main (String[]args)
    {
        int [] a = { 33, 78, 58, 63, 10, 95, 32
                     82, 18, 40 };
        int max = a [0];

        for ( int i = 0; i < 10; i++)
        {
            if (max < a [i])
                max = a[i];

            System.out.println(" Maximum is "+ma
        }
    }
```

output

maximum is : 95

(2) Multi dimensional Array. (2D Array)

Syntax

```
int [] [] a = new int [2][3];
```

(or)

```
int [] [] a = {{ 1,2,3},{4,5,6}};
```

Prajwal Yadav

	0	1	2
0	00	01	02
1	10	11	12

Ex

```
Class TwodimArray
{
  public static void main (String [] args)
  {
    int a [] [] = new int [2] [3];

    a [0] [0] = 1;
    a [0] [1] = 2;
    a [0] [2] = 3;
    a [1] [0] = 4;
    a [1] [1] = 5;
    a [1] [2] = 6;
    system.out.print ln (a [0][0] + " " + a[0][1] +
                    " " + a [0] [2]);
    system.out.print ln (a[1][0] + " " + a[1][1] +
                    " " + a[1] [2]);
  }
}
```

Output,

1 2 3
4 5 6

Prajwal Yadav

* Non - Rectangular Array.

- In non - Rectangular array number of rows elements are fixed, but number of columns for each row different.

* Operators

Operator in Java is a symbol that is used to perform operation

(1) Arithmetic Operator

Operator	meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

(2) Bitwise Operator

Operator	meaning
&	bitwise AND
|	bitwise OR
^	bitwise XOR
~	bitwise compliment

Prajwal Yadav

<<	left shift
>>	right shift
>>>	zero fill right shift

(3) Rational / Relational Operator

Operator	Meaning
==	equal to
!=	not equal to
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

(4) Logical Operator

&&	logical AND	True if all
||	logical OR	True if once
!	logical not	True if False

(5) Assignment Operator

=	assign value		a = 3 , b = 2
+=	a = a + b;	→	5
-=	a = a - b;	→	1
*=	a = a * b;	→	6
/=	a = a / b;	→	1
%=	a = a % b;	→	1

Prajwal Yadav

⑥ Conditional / Ternary Operator

Variable x = (expression) ? value if true : value if false

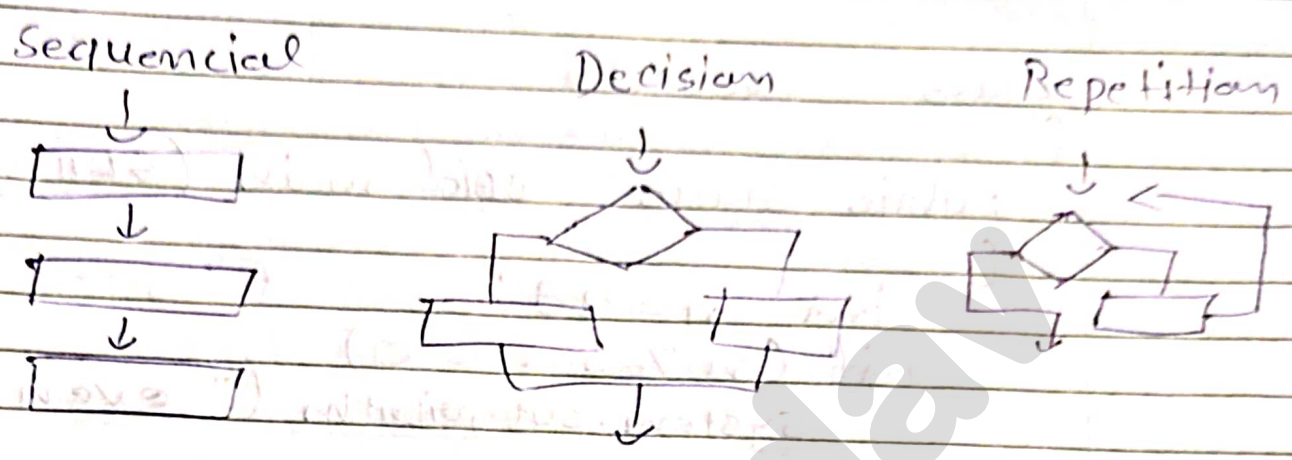
Ex

```
Class Demo
{
    public static void main (String [] arg)
    {
        int a = 2, b = 5;
        int min = (a<b) ? a : b;
        System.out.print (min);
    }
}
```

Output :- 2

Prajwal Yadav

* Control / Decision Statement



* if Statement

Syntax

```
if (condition)
{
    Statement;
}
```

ex

```
if (a < 0)
    system.out.print (" Number is Negative ")
```

* if-else Statement

syntax

```
if (condition)
        Statement-1;
else
        Statement-2;
```

ex

```
if (a > b)
    system.out.println (" a is big ");
else
    system.out.println (" b is big ");
```

Prajwal Yadav

* Print Number is even or odd

```
class evenodd
{
    public static void main (String[] args)
    {
        int n = 29;
        if (n % 2 == 0)
            system.out.println (" even");
        else
            system.out.println (" odd");
    }
}
```

Output :- odd

* if-else-if-else (if else ladder)

Syntax

```
if (condition 1)
        statement - 1;
else if (condition - 2)
        statement 2;
else if (condition 3)
        statement 3;
        :
else if (condition N)
        statement N;
else
        default statement;
```

Prajwal Yadav

Prajwal Yadav

Date
Page

Example - 1

```
if (day == 1)
    System.out.println("Monday");
else if (day == 2)
    System.out.println("Tuesday");
else if (day == 3)
    system.out.println("Wednesday");
else if (day == 4)
    system.out.println("Thursday");
else if (day == 5)
    system.out.println("Friday");
else if (day == 6)
    system.out.println("Saturday");
else if (day == 7)
    system.out.println("Sunday");
else
    system.out.println("Invalid day");
```

Example-2

```
if (P >= 75)
        Distinction
else if (P >= 66)
        First Class
else if (P >= 45)
        Second Class
else if (P >= 35)
        Pass Class
else
        fail
```

Prajwal Yadav

→ Nested if else

Syntax

```
if (condition)
{
    if (condition)
        statement 1;
    else
        statement 2;
}
else
{
    if (condition)
        statement 3;
    else
        statement 4;
}
```

Ex

```
if (a > b)
{
    if (a > c)
        max = a;
    else
        max = c;
}
else
{
    if (b > c)
        max = b;
    else
        max = c;
}
```

Prajwal Yadav

* Switch Statement

Syntax

```
switch (Expression or variable)
{
    case value1 : statement ;
                  break ;
    case value2 : statement ;
                  break ;
      :
    case value N : statement ;
                   break ;
    default : statement ;
}
```

Ex

```
switch (Day)
{
    case 1 : system.out.print ("Monday")
             break ;
    case 2 : s.o.p ("Tuesday") ;
             break ;
      :
    case 7 : s.o.p.("sunday") ;
             break ;
    default : s.o.p ("Invaild Value")
}
```

Prajwal Yadav

```
                    Excellence
case 'A' :  ~~Excellent~~
            break;
case 'B' : First class
           break;
case 'C' : Second class
           break;
case 'D' : Pass
           break;
case 'E' : fail
           break;
default : Invalid
          break;
```

Prajwal Yadav

(4) Loop

(1) while loop

Syntax

```
while (condition)
{
    Statement ;
}
```

ex

```
int n = 1 ;
while (n<=5)
{
    System.out.println(n);
    n++;
}
```

- **output** :-

```
1
2
3
4
5
```

(2) do while loop

Syntax

```
do
{
    Statement ;
} while (condition);
```

ex

```
int n=1)
do
{
    s.o.p (n);
    n++ ;
} while (n<=5);
```

output

```
1
2
3
4
5
```

Prajwal Yadav

③ for loop

Syntax

```
for (initial ; condition ; increment /decrem
{
        Statements ;
}
```

ex

```
for (int n = 1 ; n <= 5 ; n++)
{
    S.O.P (n) ;
}
```

Output

```
1
2
3
4
5
```

Prajwal Yadav

Jump statement

(1) break statement

- break statement took control outside of switch or loops

syntax.

```
while (condition)
{
    ========
    ========
    break;
    ========
}
```

Ex

```
int n = 1;
while (1)
{
    if (n==5)
        break;
    System.out.println(n);
    n++;
}
```

Output

```
1
2
3
4
```

Prajwal Yadav

(5) Continue Statement

- Skip next statement of loop & continue loop.

Syntax

```
while (condition)
{
    ------
    ------
    continue;
    ------
}
```

Ex

```
int n = 1;
while (n <= 10)
{
    if(n%2 == 1)
    {
        n++;
        continue;
    }
    system.o.p (n);
    n++;
}
```

Output :-

2
4
6
8
10

Praiyal Yadav

Unit-2 OOP Concepts

POP	OOP
- Program is divided into small parts called function.	- Program divided into parts called objects.
- Procedure oriented programing	- Object oriented programing
- Top down approch	- Bottom up approch
- does not have any access specifier	- OOP has access specifiers name public, private, protected
- POP Data can move freely from function to function in system	- OOP Object can communicate with each other using member function
- To add new data and function in POP is not so easy	- OOP Provides an easy way to add new data and function
- Ex - C, VB, FORTRAN	- Ex - C++, Java, VB.Net

* Class :- Collection of Objects.

* Object :- Object is instance of class

Prajwal Yadav

* Encapsulation

- Encapsulation is a process of wrapping of data and method in a single Unit.

* Polymorphism

- Polymorphism is the greek work whose meaning is "same Object having different behaviour".

① Compile Time polymorphism
② Run Time polymorphism

* Abstraction

- Abstraction is a process of hiding the implementation details from the user, Only the highlighted set of services provided to the user.

* Inheritance

- The process by which one object can use properties of another object is called inheritance.

Pramod Yadav

* Object & Class Example.

demo.java

```
Class Student
{
        int rollno;
        String name;

        void StoreData (int r, String n)
        {
                rollno = r;
                name = n;
        }
        void printdata ()
        {
            System.out.println (rollno);
            System.out.println (name);
        }
}
Class demo
{
  public static void main (String [] args)
  {
      student S1 = new student();
      S1.storedata (1, "Yash");
      S1.printdata();
      system.out.println ("Name is :"
                          +S1.name);
  }
}
```

Prajwal Yadav

Output

1
Yash
Name is : Yash

* Accessing rules

Access Modifiers

	Private	Package	protected	Public
Same-Class	Yes	Yes	Yes	Yes
Same package different Class	No	Yes	No	Yes
Same package sub class	No		Yes	Yes
Different package sub class	No	No	Yes	Yes
Different Package Different class	No	No	No	Yes

Prajwal Yadav

This Keyword

- "This" keyword is used to store address of object.

- This keyword is always a reference to the object on which method was invoked.

Example

```
Class Student
{
    int rollno;
    Student (int rollno)
    {
        this.rollno = rollno;
    }

    void printdata ()
    {
    system.out.println (rollno);
    }
}
class demo
{
public static void main (
                    String [] args)
{
    student s1 = new student (15)
    s1. printdata ();
}
```

Prajwal Yadav

output

15

* Static Keyword

- Static variable
- Static method / function
- Static Block
- Static Nested Class

Static variable & function

```
Class Corona
{
    Static int count = 5;
    void Storedata()
    {
        count ++;
    }

    Static void printcount()
    {
        System.out.println(count);
    }
}

Class demo
{
    public Static void main
        (String [] args)
    {
```

Prajwal Yadav

```
Corona S1 = new Corona();
S1.Staredata();
Corona S2 = new Corona();
S2.Staredata();
S1.printcount();
S2.printcount();
Corona.printcount();
                // only for static function
}

}
```

Output

7
7
7

Static Block

```
Class Student
{
  Static
  {
   System.out.println("In Static Block
  }

  public static void main(String[] args)
  {
   System.out.println("In main function")

  }
}
```

Output

In Static Block
In main function

* Final Keyword

Final Variable

```
Class Test
{
    final int a = 30; //can not modif
    void displaydata()
    {
        System.out.println(a);
    }
}
Class Finalkeyword
{
  public static void main(String [] args)
  {
      Test t = new Test();
      t.displaydata();
  }
}
```

Output

30

Prajwal Yadav

Final method / function

```
Class Test
{
    int a = 30;
// can not override final void displaydata()
    {
        a = 50;
        System.out.println(a);
    }
}
Class Finalkeyword
{
  public static void main (String [] args)
  {
    Test t = new Test();
    t.displaydata();
  }
}
```

Output

50

Prajwal Yadav

Final Class

```
final Class Test   // can not inherit
{
        int a = 30;
        void display()
        {
            a=50;
            System.out.println(a);
        }
}
class Finalkeyword
{
 public static void main(String[]args)
 {
     Test t = new Test();
     t.display();
 }
}
```

- Output

50

Prajwal Yadav

(*) Constructor

- A constructor is used to Initialize a new Object.
- It has the same name as its class and do not have any return type.
- Constructor is automatically called immediately when the ~~is~~ Object is created.
- Constructors are not method, they aren't inherited.

(1) Defualt Constructor

- This constructor does not have any arguments.

example

```
class Student
{
    int rollno;
    Student ()
    {
        System.out.println("Defualt constructor");
    }
}
```

Prajwal Yadav

```
Class Defult Constructor
{
    public static void main( String [] args)
    {
        Student S = new student ();
    }
}
```

Output

Defualt Constructor.

② Parametorized constructor

- Constructor with arguments is known as Parameterized Constructor.

example

```
Class student
{
    Student ( int r)
    {
        system.out.println( " My Number: " +r
    }
}
Class ParametricConstructor
{
    public Static void main (String [] args
    {
        Student s = new student (16);
    }
}
```

Prajwal Yadav

output

My Number : 16

③ Copy Constructor

- A copy constructor is used to initialize an object from another object, copy constructor takes object as argument.

example

```
Class Student
{
    int roll;
    String name;
    Student (int i, string n)
    {
        roll = i;
        name = n;
    }
    Student (student s)
    {
        roll = s.roll;
        name = s.name;
        System.out.println (roll + " " + name
    }
}
```

Prajwal Yadav

```
class CopyConstructor
{ public static void main (String [] Args)
  {Student S1 = new Student (16, "Ram")
  Student S2 = new Student (S1);
} }
```

Output

16 Ram

(*) Passing & Returning Object from method

```
class Test
{
    Private int x;
    Private int y;
    void setdata (int a , int b)
    {
        x = a;
        y = b;
    }
    Test copydata (Test obj)
    {
        Test p;
        p.x = obj.x;
        p.y = obj.y;
        return p;
    }
    void printdata ()
    {
      System.out.println (x);
      System.out.println (y);
    }
```

Prajwal Yadav

```
Class ObjArgument
{
    public static void main (String[ args)
    {
        Test t1 = new Test();
        Test t2 = new Test();
        t1.setdata(5, 10);
        t2 = t2.copydata(t1);

        t1.printdata();
        t2.printdata();
    }
}
```

⇒ Output

```
5
10
5
10
```

Method Overloading

- Method overloading is a feature in Java that allows a class to have multiple method with the same name but different parameters.

Prajwal Yadav

- different number of Argument
- different data type of Arguments.

ex.

```
Class Test
{
    int area (int l)
    {
        return (l * l);
    }
    int area (int l , int b)
    {
        return (l * b);
    }
    double area (double r)
    {
        return (3.1415 * r * r);
    }
}

Class MethodOverloading
{
public static void main (String [] args)
{
Test t = new Test();
System.out.println ("Area of Square is:" + t.area(30));
" " " ("Area of Rectangle is: " + t.area(20, 40));
" " " ("Area of Circle is: " + t.area(10.0)
```

Prajwal Yadav

Output :

Area of Square is : 900
Area of Rectangle is : 800
Area of circle is : 314.15

Constructor Overloading

```
Class Add
{
    int x = 10
    Add ()
    {
    System.out.println (x);
    }
    Add (int y)
    {
    system.out.println(x + y);
    }
    Add( int y, int z)
    {
    System.out.println (x+y+z);
    }
}

Class ConstructorOverloading
{
public static void main (String [] args)
{
    Add Obj = new Add ();
    Add Obj2 = new Add (10);
```

Prajwal Yadav

```
        Add obj 3 = new Add(10,20);
    }
}
```

Output

10
20
40

Wrapper Class

use to convert primitive Data type into objects.

Primitive		Wrapper Class
char	→	Character
boolean	→	Boolean
byte	→	Byte
Short	→	Short
int	→	Integer
long	→	Long
float	→	Float
double	→	Double

ex

```
int a = 55;
Integer b = new Integer(55);
```

java.util Package & Framworks

only object use

Prajwal Yadav

* Scanner Class

```
import java.util.Scanner;

Class ScannerClass
{
  public static void main (String [] args)
  {
    Scanner s = new Scanner(System.in)
    System.out.println("Enter RollNo &
                        Name :");
    int i = s.nextInt ();
    String st = s.nextLine ();
    System.out.println("Roll.No is :"+i);
    System.out.println(" Name is :"+st);
  }
}
```

Output

```
Enter RollNo & Name :
1
Shiv
Roll No is : 1
Name is : Shiv
```

Prajwal Yadav

String Class

length ()
charAt ()
substring ()
equals () / equalsIgnoreCase()
compareTo ()
indexof () / lastIndexOf ()
contains()
replace ()
toUpperCase ()
to lowerCase()
trim ()
Split ()
format ()

```
Class StringFunction
{
public static void main (string [] args)
{
String S1 = "Good Morning";

System.out.println ( S1.length());
( S1.charAt (5));
( S1.substring (5));
( S1.substring (0, 4));

String S2 = "Good Mor";

( S1.equals (S2));
(S1.equalsIgnoreCase (S2))
( S1.compareTo (S2));
(S1.indexof ('O');
```

Prajwal Yadav

```
(s1.contains("ni"));
(s1.replace("Good", "Bad
(s1.toUpperCase());
(s1.toLowerCase());

String [] s3 = s1.split(" ");
for (String word : s3)
{
  System.out.println(word);
}
String s4 = String.format("Hi,%s",
}
```

Output :-

12	Good
M	Morning
morning	
Good	Hi, Good Mornin
false	
false	
4	
1	
6	
true	
Bad Morning	
GOOD MORNING	
good morning	

Prajwal Yadav

(4) Command Line Arguments

- During Program execution information passed following at program at ~~program~~ ~~in~~ name is CLA.

- pass command line Argument to the main method through an array Strings

Test.java

```
Public Class Test
{
  public static void main(String [] args)
  {
    if (args.length == 0)
      System.out.println(" Ple enter ");

    for (int i = 0; i < args.length ; i++)
    {
      System.out.println(" args[" + i + "] + args[i]
    }
  }
}
```

=> Compile your Java Program

javac Test.java

=> Run your Java Program

java Test Hello, Hi Prajwal

Prajwal Yadav

⇒ Output

```
args [0] : Hello
args [1] : Hi
args [2] : Prajwal
```

Prajwal Yadav

Unit-3 Inheritance, Packages & Interfaces.

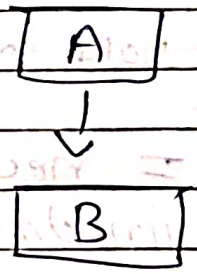
* Inheritance

- The mechanism of deriving a new class from old class is called inheritance.
- When you inherit from an existing class, you can reuse method and fields of the parent class. Also you can add new method and field in your current class also

* Type of Inheritance

1. single
2. Multiple
3. Multilevel
4. Hierarchical
5. Hybrid

1. **Single Inheritance** :- When a class inherits another class it is known as single inheritance.



Example

Prajwal Yadav Notes

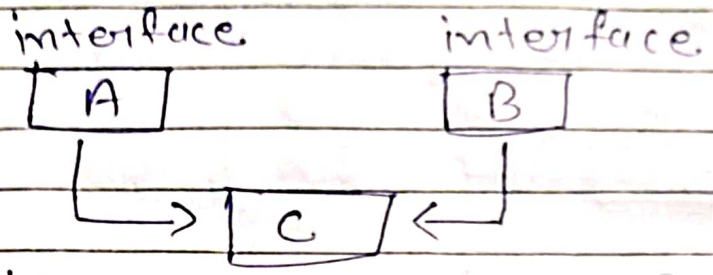
Prajwal Yadav

```
Class A
{
    protected int x;
    A (int a)
    {
        x = a;
    }
}
class B extends A
{
    Protected int y;
    B (int a, int b)
    {
        super (a);
        y = b;
    }
    void printdata ()
    {
        system.out.print ln (x);
        system.out.println (y);
    }
}
Class SingleInheritance
{
    public static void main (string [] args)
    {
        B obj = new B (10, 5);
        Obj. printdata ();
    }
}
```

Output :- 10
5

Prajwal Yadav

(2) Multiple Inheritance :- One class extends properties of several super classes is known as Multiple inheritance



Example

```
interface A
{
    void printdata();
}
interface B
{
    void displaydata();
}
Class C implements A, B
{
    Void printdata()
    {
     system.out.println("Interface A");
    }
    void display data()
    {
     system.out.Priint ln ("Interface B");
    }
}
Class MultipleInheritance
{
  public static void main (String [] args)
  {
```

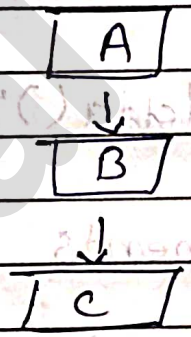
Prajwal Yadav

```
        C obj = new C();
        obj.printdata();
        obj.displaydata();
    }
}
```

Output

Interface A
Interface B

(3) Multilevel Inheritance :- When new class is derived from derived class is known as multilevel Inheritance



Example

```
Class A
{
    protected int x;
    A (int a)
    {
        x = a;
    }
}

Class B extends A
{
    protected int y;
```

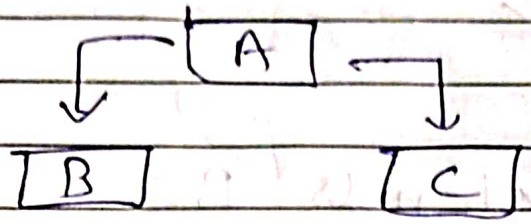
Prajwal Yadav

```
    B (int a , int b)
    {
        super (a);
        y = b;
    }
}
Class  C extends B
{
    protected int z;
    C (int a, int b, int c)
    {
        super (a, b);
        z = c;
    }
    void printdata()
    {
        System.out.println (x);
        System.out.println (y);
        System.out.println (z);
    }
}
Class MultilevelInheritance
{
    Public Static void main (String [] args)
    {
        C obj = new C(10, 5, 15);
        obj. printdata();
    }
}
```

output :- 10
5
15

Praiwal Yadav

(4) Hierarchical inheritance :- when one Super Clas having may subclasses is known as Hierarchical inheritance.




```
Class A {
    protected int x;
    A (int a) {
        x = a;
    }
}
Class B extends A
{
    protected int y;
    B (int a , int b)
    {
        super (a);
        y = b;
    }
    void printdata ()
    {
        System.out.println (x);
        System.out.println (y);
    }
}
Class C extends A
{
    protected int z;
    C (int a , int b)
    {
```

Prajwal Yadav

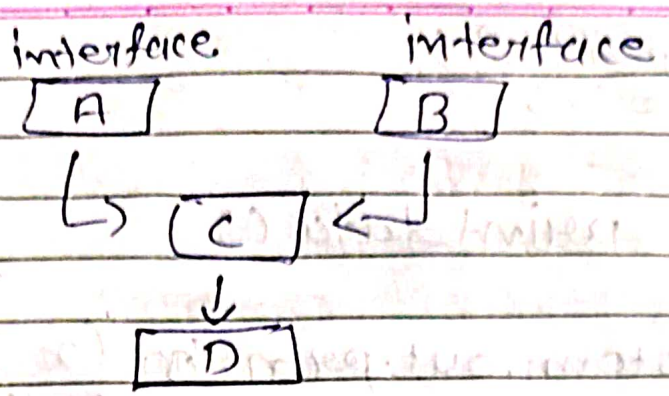
```
        super(a);
        z = b;
    }
    void printdata()
    {
        system.out.println(x);
        system.out.println(z);
    }
}
class Hierarchical inheritance
{
    public static void main(String[] args)
    {
        B obj1 = new B(10,5);
        obj1.printdata();
        C obj2 = new C(50,60);
        obj2.printdata();
    }
}
```

Output

```
10
5
50
60
```

(5) Hybrid Inheritance: We can inherit properties of several classes at same level is know as Hybrid inheritance

Prajwal Yadav




```
interface A
{
    void printdata();
}
interface B
{
    void displaydata();
}
class C implement A, B
{
    Void printdata()
    {
    system.out.println("Interface A");
    }
    void displaydata(){
        system.out.println("Interface B");
    }
}
Class D extends C
{
    protected int x;
    D (int a)
    {
        x=a;
    }
```

Prajwal Yadav

```
        void viewdata ()
        {
          System.out.println (x);
        }
}
class    Hybridinheritance
{
  public static void main(String[]args)
  {
        D obj1 = new D(10);
        obj1.printdata();
        obj1.displaydata();
        obj1.viewdata();
  }
}
```

Output :-

```
Interface A
Interface B
 10
```

Pragyal Yadav

* Method Overriding

- If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.

```
Class A
{
    void printdata()
    {
        System.out.println("In Class A");
    }
}
Class B extends A
{
    void printdata()
    {
        System.out.println("In class B");
    }
}
Class MethodOverriding
{
    public Static void main(String[]args)
    {
        B obj1 = new B();
        obj1.printdata();
        A obj2 = new A();
        obj2.printdata();
    }
}
```

Prajwal Yadav

Output :

In Class A
In Class B

Object Class & override its methods

- ClassgetClass()
- hashcode()
- pubilc boolean equals(Object Obj;)
- Clone()
- public String tostring()
- notify()
- notify All()
- wait(long timeout)
- wait(long timeout, int nanos)
- wait()
- protected void finalize()

```
Class Test
{
    int id;
    String name;
    Test(int i, String n)
    {
        id = i;
        name = n;
    }
    Public String tostring()
    {
        return "Id is : "+id+", Name is
                : "+name;
    }
```

Prajwal Yadav

```
    public boolean equals (object obj)
    {
        Test t = (Test) obj;
        return (id == t.id);
    }
}

Class ObjectClass
{
    Public static void main (String [] args)
    {
        Test t1 = new Test (1, "Krishna");
        Test t2 = new Test (2, "shiv");
        Test t3 = new Test (1, "Ram");
        system.out.println (t1.toString());
        System.out.println (t1.equals (t2));
        system.out.println (t1.equals (t3));
    }
}
```

Output

Id is : 1 , Name is : Krishna
false
True

* **Interface** :- An interface in Java is a blueprint of a class
- It has static constants and abstract methods.
- Interface is a keyword used to define a collection of method definitions and constant values.

Prajwal Yadav

* Abstract Class

- Can not create an objects.
- at least one abstract method is create.
- normal methods, variables, constructors can write.
- In derived class, abstract method is compulsary implemented, otherwise derived class become abstract also.

- Example

```
abstract class Test
{
    void message()
    {
        system.out.println("In abstract class");
    }
    abstract void printdata();
}
class demo extends Test
{
    void printdata()
    {
        system.out.println("In Demo class");
    }
}
class AbstractClass
{
  public static void main(String [] args)
  {
```

Prajwal Yadav

```
        demo d = new demo();
        d.message();
        d.printdata();
    }
}
```

output-

In Abstract Class
In Demo Class

* Final Class

- The final keyword in java is used to restrict the user.
- The final keyword can be used in many context.
  1. Variable
  2. method
  3. Class

Example

```
final Class A
{
    private final int value;
    A(int value)
    {
        this.value = value;
    }
    int getvalue()
    {
        return value;
    }
}
```

Prajwal Yadav

```
Public class FinalClass
{
    public static void main (String [] args)
    {
        A obj = new A (10);
        System.out.println ("value :" + obj.getValue
    }
}
```

Output

value : 10

* Package

- A package is a collection of logically related classes, Interfaces or subpackages.
- Package act as "containers" for class.

Prajwal Yadav

Test.java

```
Package info;
public class Test
{
    public int x=20;
    public void message()
    {
        system.out.Print ln("From class
                                Test");
    }
}
```

Demo.java

```
Package Tech;
import info.Test; // import single class
import info.*; // import all class

Class Demo
{
  public static void main(String [] args)
  {
      Test t = new Test();
      info.Test t = new info.Test();
      system.out.println(t.x);
      t.message();
  }
}
```

Fully qualified name

output :- 20
From class Test

Prajwal Yadav

Unit - 4 Exception Handling & Multithreading

* Type of Error

Error

- Run Time (Exception) → Arithmatic Exception, Array Index out of Bounds Exception
- Compile Time → Syntax
- logical

* Exception Handling Mechanism

1. try
2. catch
3. throw
4. throws
5. finally

* try ... catch Statement

Syntax

```
try
{
    Statement
}
catch (Exception e)
{
    message
}
```

Prajwal Yadav

Example

```
Class    exceptionDemo
{
  public  static void main(String [] args)
  {
      int  a[] = { 10, 20, 0, 30, 40 };
      try
      {
        system.out.println (a[1] / a[2]);
      }
      catch (Arithmatic Exception e)
      {
        system.out.println (e.getMessage ());
      }
  }
}
```

or

("Divide by zero")

Output

/ by zero

or

Divide by zero

* Multiple Catch Blocks

Syntax

```
try
{
    Statement
}
catch (Exception 1)
{
```

Prajwal Yadav

```
        message
    }
    catch (Exception e)
    {
        Message
    }
        ⋮
```

Example

```
class exceptionDemo
{
    public static void main(String[] args)
    {
        int a[] = {10, 20, 0, 30, 40};
        try
        {
        system.out.println(a[1] / a[20]);
        }
        catch (ArithmaticException e)
        {
        System.out.println(e.getMessage());
        }
        Catch (ArrayIndexOutOfBoundsExceptio
                                    e1)
        {
            system.out.println("Index out of
                                range");
        }
    }
}
```

Output :- Index out of range

Prajwal Yadav

throw keyword

(Exception Generate Explicitly)

Example

```
Class test
{
  public static void main (String[] Args)
  {
      int x = 50;
      try
      {
          if (x > 10)
              throw new Exception();
          else
              System.out.println ("Value is valid
      }
      catch (Exception e)
      {
          System.out.println ("value is
                              not valid ");
      }
  }
}
```

Output

value is not valid

Prajwal Yadav

throws Keyword

Example

```
import java.io.*;
Class test
{
    public static void main(String [] Args)
                        throws IOException
    {
        char c= (char) System.in.read();
        if ( c=='N' || c=='n')
            System.out.print("Negative");
        else
            System.out.print("Positive");
    }
}
```

output

```
S
Positive
```

Finally Clause

- finally is a Keyword that executes a block of statements regardless of whether any exception occurred or not.
- The code of finally block always execute whether or not exception has occurred.

Prajwal Yadav

Example

```
Class Test
{
  public static void main (String [] args)
  {
      int a = 10 , b = 0;
      try
      {
          System.out.println (a/b);
      }
      catch (ArithmeticException a)
      {
        System.out.println(a.getMessage());
      }
      finally
      {
          System.out.println ("Done");
      }
  }
}
```

Output

/ by zero

Done

Multithreading

- Multithreading is a conceptual programming paradigm where a program is divided into two or more subprogram, which

Prajwal Yadav

Can be implemented at the same ~~memory space~~ time in parallel.

- Threads in Java are subprogram ~~which can~~ of main application program and share the same memory space.

(*) Multithreaded Process

Process

Thread 1 → operation 1, " 2, " 3

Thread 2 → operation 1, " 2, " 3

fetch
decode
execute

(*) Creation of Thread

Thread

Extends → class (Thread)

implements → interface (Runnable)

Run() (override)

Praiwal Yadav

Thread Class

getName()
setName()
run()
Start()
Sleep()
isAlive()
join()
getPriority()
setPriority()
suspend()
resume()
Stop()

Use Thread Class

Example

```
Class test extends Thread
{
  String msg;
  test (String m)
  {
    msg = m;
    Start();
  }
  public void run()
  {
    System.out.println (msg);
  }
}
```

Prajwal Yadav

```
Class ThreadDemo
{
  Public static void main (String [] args)
  {
    test t = new test("Thread Activated")
  }
}
```

Output

Thread Activated

(*) => Use Runnable Interface

```
Class test implements Runnable
{
    String msg;
    Thread td;
    test (string m)
    {
        msg = m;
        td = new Thread (this);
        td.start()
    }
    public void run()
    {
      System.out.println (msg);
    }
}
Class ThreadDemo
{
  public static void main (String [] args)
  {
```

Prajwal Yadav

```
        test t = new test ("Thread Activated");
    }
}
```

Output

Thread Activated

(4) Multiple Threads

=> Use Thread Class

```
Class test extends Thread
{
    test (string m)
    {
        super (m);
        Start ();
    }
    public void run()
    {
        for (int i=0; i<5; i++)
        {
            System.out.println (Thread.currentThread
                                .getName());
        }
    }
}
Class ThreadDemo
{
    public static void main (String [] args)
    {
```

Praiwal Yadav

```
        test t1 = new test ("Thread 1");
        test t2 = new test ("Thread 2");
    }
}
```

output

Thread 1	Thread 2
Thread 1	Thread 1
Thread 2	thread 1
Thread 2	thread 1
Thread 2	Thread 2

~~Thread 2~~
~~Thread 1~~
~~Thread 1~~
~~Thread 1~~
~~Thread 2~~

=> Runnable Interface

```
Class test implements Runnable
{
    Thread td;
    test (String m)
    {
        td = new Thread (this, m);
        td.start();
    }
    public void run()
    {
        for (int i=0; i<5; i++)
        {
            System.out.println (Thread.currentThread()
                .getName());
        }
    }
```

Prajwal Yadav

```
Class ThreadDemo
{
    public static void main(String[] Args)
    {
        test t1 = new test("Thread 1");
        test t2 = new test("Thread 2");
    }
}
```

Output

```
Thread 1
Thread 1
"      2
"      2
"      2
"      2

Thread 1
"      1
"      1
Thread 2
```

Prajwal Yadav

Thread Priority

minimum Priority — 1
normal priority — 5
Maximum priority — 10

Example

```
Class test extends Thread
{
    test (String m)
    {
        super (m);
    }
    public void run(){
        for (int i=0; i<3; i++)
        {
        System.out.println(Thread.
            currentThread().getName());
        }
    }
}
Class ThreadPriority
{
  Public static void main (String [] Args)
  {
    test t1 = new test ("Thread-1");
    test t2 = new test ("Thread-2");
    test t3 = new test ("Thread-3");
    system.out.println ("current Priority
        of t1 is : " + t1.getPriority());
    ~~t1.Start();~~
    t1.setpriority (3);
```

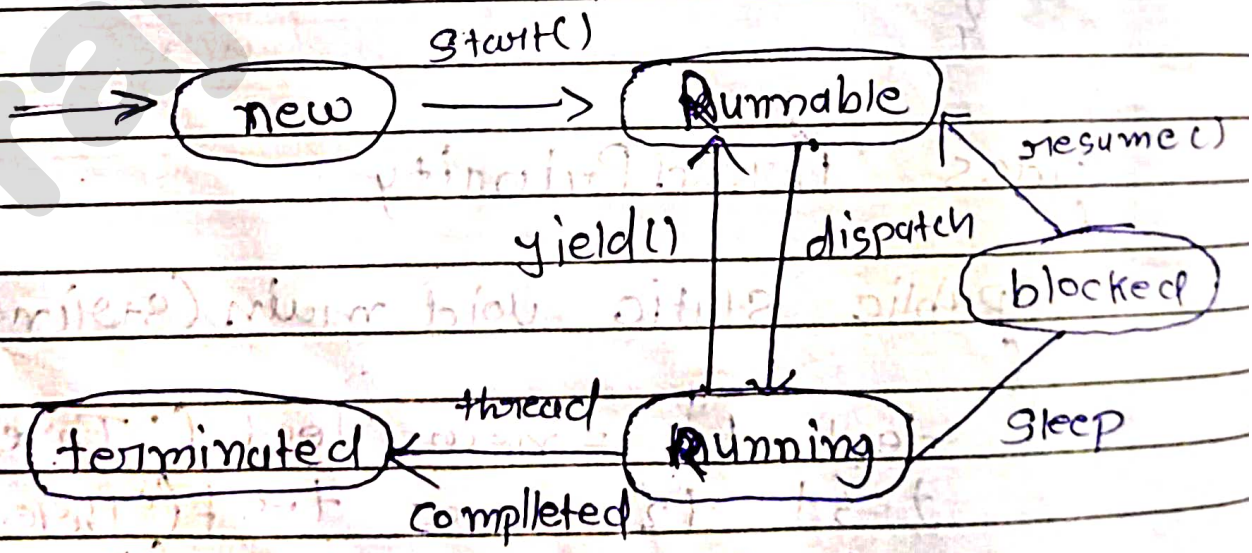
Prajwal Yadav

```
        t1.start();
        t2.start();
        t3.start();
    }
}
```

Output

Thread - 2
Thread - 2
Thread - 3
Thread - 3
Thread - 3
Thread - 2
Thread - 1
Thread - 1
Thread - 1

(*) Life Cycle of a Thread



(1) new :- A thread begins its life cycle in the new state. It remain in this state

until the start() method is called on it.

(2) Runnable : After invocation of start() method on new thread, the thread becomes runnable

(3) Running : A method is in running thread if the thread scheduler has selected it

(4) Waiting :- A thread is waiting for another thread to perform a task. In this stage, the thread is still alive

(5) Terminated :- A thread enters the terminated state when it complete it task.

*) Exception handling in Thread.

```
Class test extends Thread
{
    public void run()
    {
        System.out.println("Exception Generated"
        throw new RunTimeException();
    }
}
Class ThreadException
{
public static void main(String [] args)
{
    test t = new test();
    t.start();
```

Prajwal Yadav

```
try
{
    t.sleep (1000);
}
catch (Exception e)
{
    System.out.println ("Expection
                        Handled");
}
}
```

Output

Exception Generated
Exception Handled

Prajwal Yadav

Unit - 5 File handling and Collection Framework

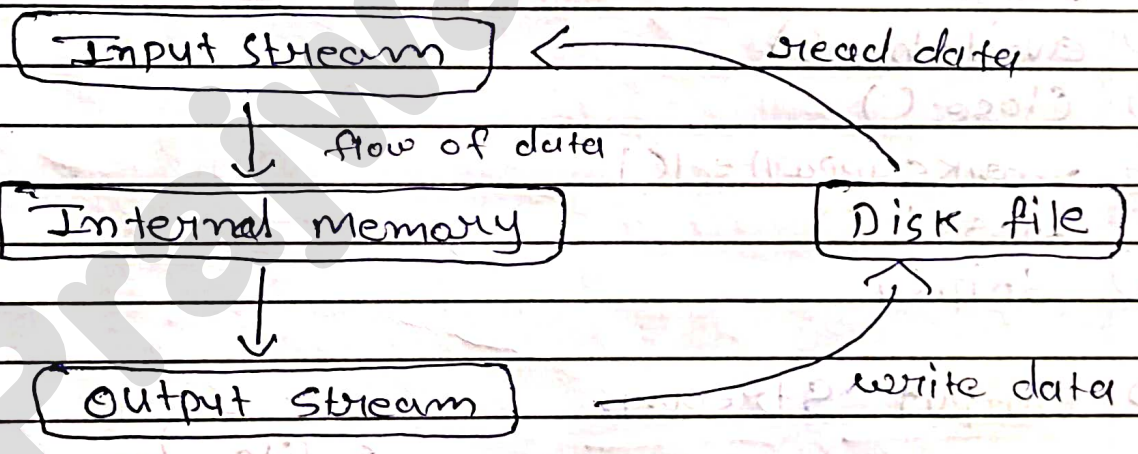
* **Stream** :- A stream is a sequence of data. In Java a stream is composed of bytes.

- Stream are pipelines for sending and receiving information in Java programs.

(*) Stream Types

(1) Character Stream → Reader and writer
(2) Byte Stream → Input & output

(*) How Files and Stream work



Prajwal Yadav

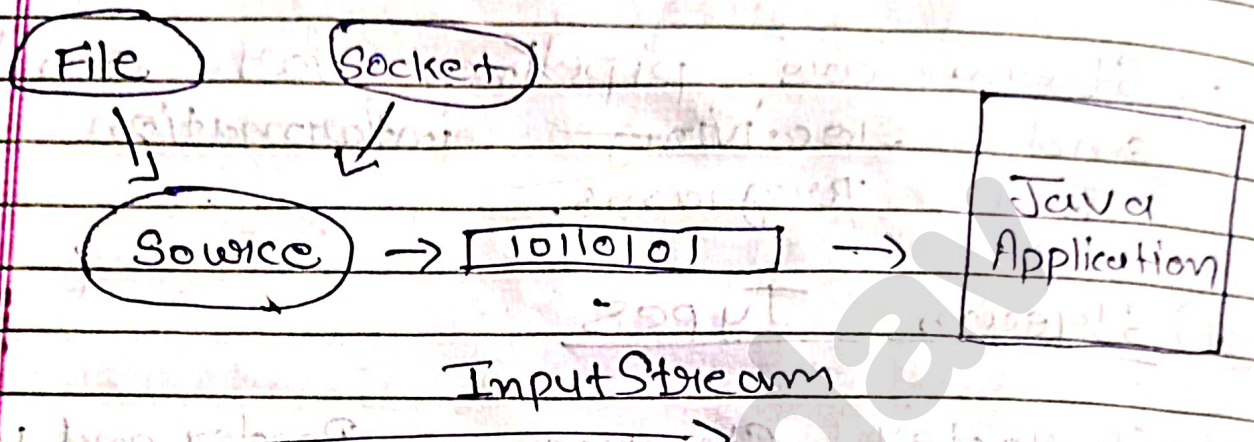
Notes

Diploma Sem - 4
AOOP

Prajwal Yadav

I/O Classes

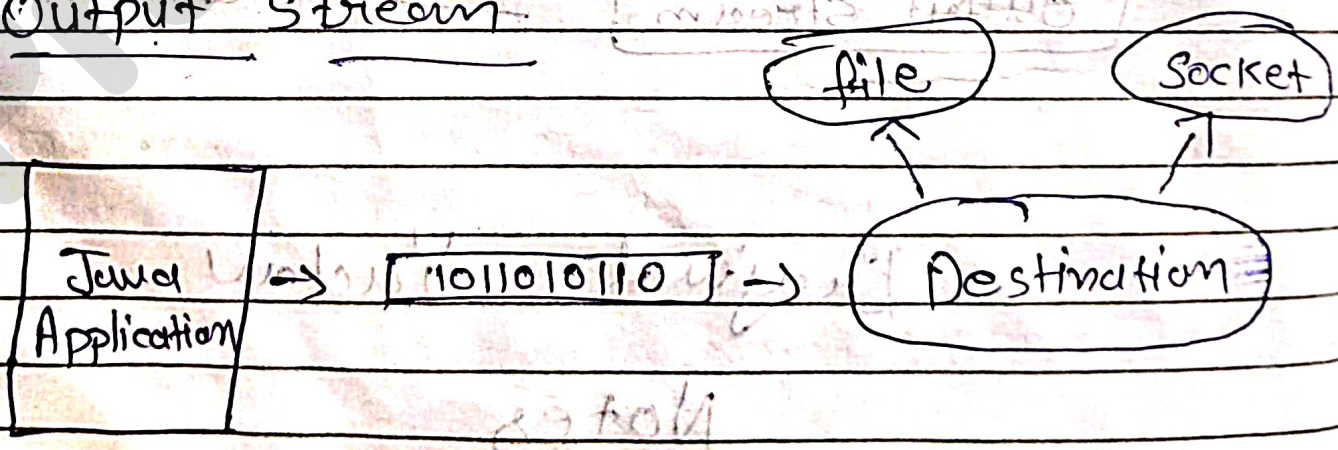
⇒ Input Stream



Method

1. read()
2. available()
3. close()
4. markSupported()
5. reset()
6. skip()

⇒ Output Stream



Method :- 1. write()
2. flush()
3. close()

Prajwal Yadav

(*) FileInputStream

(1) FileInputStream (String)
int read()
int read (byte[])
int read (byte[], int off, int len)
int available ()
void close ()

Example

```
import java.io.*;
public class FileInput
{
  public static void main (String[] args)
  {
    try
    {
      FileInputStream fp = new
                    FileInputStream("text.txt");
      int data;
      while ((data = fp.read()) != -1)
      {
        System.out.println ((char)data);
      }
      fp.close();
    }
    catch (IOException e)
    {
      e.printStackTrace();
    }
  }
}
```

Prajwal Yadav

Note : file already in directory

text.txt

Good Morning

Output

Good Morning

(4) File Output Stream

FileOutputStream (String)
void write (File)
void write (int b)
void write (byte [])
void write (byte [], int off, int len)
void flush ()
void close ()

Example

```
import java.io.*;
public class FileOutput
{
  public static void main(String [] args)
  {
    try
    {
      FileOutputStream fp = new
        FileOutputStream("text.txt")
```

Pranjal Yadav

```
        String msg = "Good Morning";
        byte [] msgBytes = msg.getBytes();
        fp.write (msgBytes);
        fp.close();
    }
    Catch (IO Exception e)
    {
        e.printStackTrace();
    }
}
}
```

Output

text.txt

Good Morning

* Creation of text file

```
import java.io.*;
Class student
{
  public static void main (string [] args)
  {
    File f = null;
    try
    {
      f = new File ("D://program/text.txt");
      System.out.println (" File Created :"
                    + f.CreateNewFile());
    }
```

Prajwal Yadav

```
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

output

File create : true

(*) Reading & writing text file

=> Reading from a file

```
import java.io.*;
Class Student
{
    public static void main (String [] args)
    {
        try
        {
        BufferedReader br = new
            BufferedReader(new FileReader(
                "input.txt"));
        BufferedWriter bw = new BufferedWrite
        String line;
        while((line = br.readLine()) != null)
        {
            bw.write(line);
            bw.newline();
        }
        br.close();
        bw.close();
    }
```

Prajwal Yadav

```
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
}
```

input.txt

one
two
three

← already created

=) output

output.txt

one
two
three

```
(new FileWriter("output.txt");
```

Prajwal Yadav

=> writing to a file

```
import java.io.*;
Class Student
{
    public static void main(String[] args)
    {
        try
        {
            FileWriter fp = new FileWriter("file.tx
            BufferedWrite bw = new
                BufferedWrite(fp);

            ~~Buffered Write~~
            bw.write("One");
            bw.newline();
            bw.write("Two");
            bw.newline();
            bw.write("Three");
            bw.close();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
}
```

output

file.txt

```
One
Two
Three
```

Prajwal Yadav

Collection Framework

- The collection Framework is a set of classes and interface in Java that provide a unified architecture for storing and manipulating group of object.

- Component of collection Framework
1. Interfaces
2. Classes
3. Algorithms

Collection Classes

- Java provides several collection classes in the java.util package that can be used to store and manipulate group of object.

1. ArrayList
2. LinkedList
3. HashSet
4. TreeSet
5. HashMap
6. ~~Set~~ TreeMap

Prajwal Yadav

① ArrayList

- ArrayList is a dynamic array

```
import java.Util.ArrayList;

Class ArrayList
{
    public static void main (String [] Ar
    {
        ArrayList <String> name = new
                                ArrayList <>();

        name.add("Alice");
        name.add("Bob");
        name.add("Charlie");

        System.out.println(name);
    }
}
```

output

[Alice, Bob, Charlie]

② LinkedList

```
LinkedList <String> names = new
                        LinkedList<>()
```

Prajwal Yadav

(3) Hash Set

- A hashset is an unordered set of elements with no duplicates

```
Hashset <String> names = new
                Hashset <> ();
```

- Output is unordered or no duplicates.

(4) Tree Set.

- Ordered set with no duplicates.

```
TreeSet <String> names = new
                TreeSet <> ();
```

(5) Hash Map.

- Unordered map of key-value pair with no duplicate key

```
import java.util.Hashmap;

Class Demo
{
  public static v. m (S [] args)
  {
```

Prajwal Yadav

```
Hashmap <String, Integer> ageMap
        = new Hashmap <>();

ageMap.put ("Alice", 25);
ageMap.put ("Bob", 30);

S.O.P ("Age of Alice : " +
                ageMap.get("Alice"));

for (String name : ageMap.keySet())
{

    S.O.P (ageMap.get(name));
}
}
}
```

Output

```
Age of Alice : 25
25
30
```

Prajwal Yadav

(8) TreeMap

- Sorted order key, value pair.

```
TreeMap <String, Integer> ageMap
        = new TreeMap <> ();
```

(*) For each loop

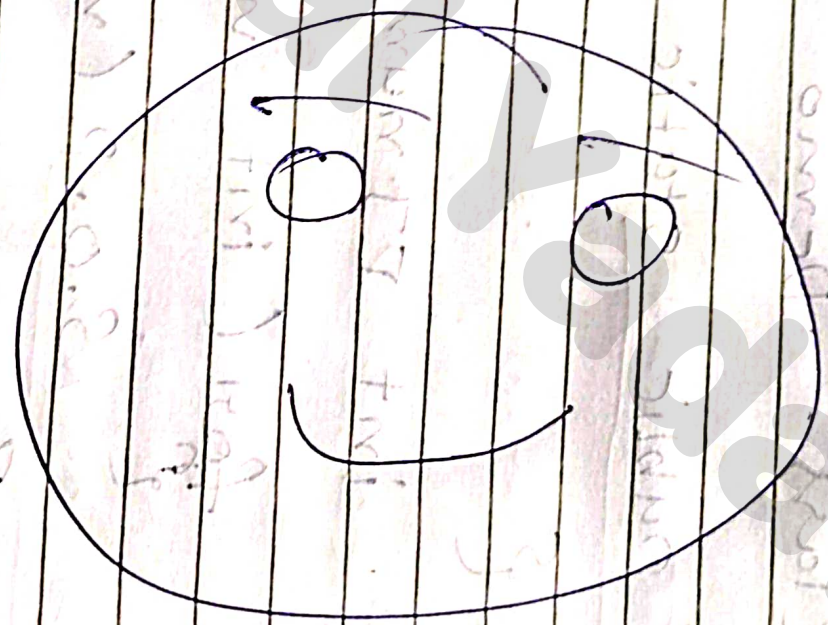
```
Class Demo
{
  public static void main (String
                          [] args)
  {

    int [] numbers = {1, 2, 3, 4, 5};

    for (int number : numbers)
    {
      S.O.P (number);
    }
  }
}
```

Output

1
2
3
4
5

Prajwal Yadav

Thank You



Pramod Yadav