JAVA

PAGE NO.
DATE / /

* Java ⇒
  - Java is a programming language & platform.
  - It is high level, secured & object oriented language.
  - Any hardware or software environment in which a program runs is known as platform.
  - Java has its own JRE (Java Runtime Environment)

* Application of Java ⇒
  - Desktop applications such as media player, antivirus, etc.
  - Web applications such as IRCTC, etc
  - Enterprise applications such as banking app's.
  - Mobile
  - Embedded system
  - Smart card
  - Robotics
  - Games, etc.

* Types of Java applications -

i) Stand alone application →
Desktop or window based applications

2) Web application → JSP, struts, servlet

3) Enterprise appln → Banking related.

4) Mobile application → Android & Java

* Java ~~platf~~ Editions / Platforms -

1) Java SE (Standard edition) -
- It includes core topics like OOP's, string, exception, multithreading, collections,

2) Java EE (Enterprise edition) -
- Used to develop web and enterprise applications. includes JSP, web services EJB, JPA, servlets, etc.

3) Java ME (Micro edition) -
- Used to develop mobile applications.

4) Java fx -

-Used to develop rich internet applications It uses lightweight user interface API.

* History of Java -

- James Gosling, mike sheridan & patrick Naughton initiated Java language project in 1991 (Sun Engg team) and designed for small, embedded system (set top box)
- Firstly called "Greentalk" by James Gosling & extension was .gt
- Then called as Oak and developed as a part of the green project. As Oak is symbol of strength and choosen as National tree of many countries like USA, france, Germany, Romania, etc.
- In 1995, renamed as "Java". Java is an Island of Indonesia where first coffee was produced (called Java coffee)
- In 1995, Time magazine called Java - one of the best product of 1995.
- JDK 1.0 released on 23 Jan. 1996.

DATE / /

* Java version History -

1. JDK Alpha & Beta → 1995
2. JDK 1.0 → 23 Jan 1996
3. JDK 1.1 → 1997
4. JDK 1.2 → 1998
5. JDK 1.3 → 2000
6. JDK 1.4 → 2002
7. Java 5.0 → 2004
8. Java SE 6 → 2006
9. Java 7 → 2011
10. Java SE 8 → 18 March 2014

* Features of Java -

- Simple (ease with c++)
- Object oriented (OOP's)
- Portable (Java bytecode)
- Platform independent
- Secured
- Robust (Strong)
- Dynamic
- Interpreted
- High performance
- Multithreaded
- Distributed (RMI & EJB)

* Difference between C++ and Java.

	C++	Java
i)	Platform dependent	i) Platform independent
ii)	Supports multiple inheritance	ii) Doesnot support multiple inheritance. Achieved by interface.
iii)	Supports operator overloading	iii) Doesnot supports operator overloading
iv)	Supports goto statement	iv) Doesn't support go to statement.
v)	Uses compiler only	v) Uses compiler + interpreter.
vi)	Supports call by value & call by reference.	vi) Support only call by valu and no call by reference
vii)	Supports structures & unions.	vii) Doeshot supports structure & unions.

viii) Doesnot have build in support for threads. It	viii) Java has built in thread support
ix) Supports virtual ~~keyboard~~ Keyword	ix) It has no virtual keywords.
x) Doesnot support >>> operator.	x) Supports unsigned right shift >>> operator.
xi) Creates a new inheritance tree always.	xi) It uses single inheritance tree always because all classes are child of object class in Java.

Simple program ⇒

Ex.
```
Public class myclass{
    public static void main (string [ ] args){

        system.out.println (" My first program");
    }
}
```

O/P ⇒ My first program

Public - Access modifier (Accessed by anyone)

class - Used for declaring a class in Java

static - Keyword used for declaring any method as static. There is no need to create object to invoke the static method.
& main method is executed by JVM & doesnot require to create object.

void ⇒ Return type of method (doesnot return anything)

main ⇒ ~~Entry~~ Entry point of program

string [ ] args ⇒ command line argument string array.

DATE / /

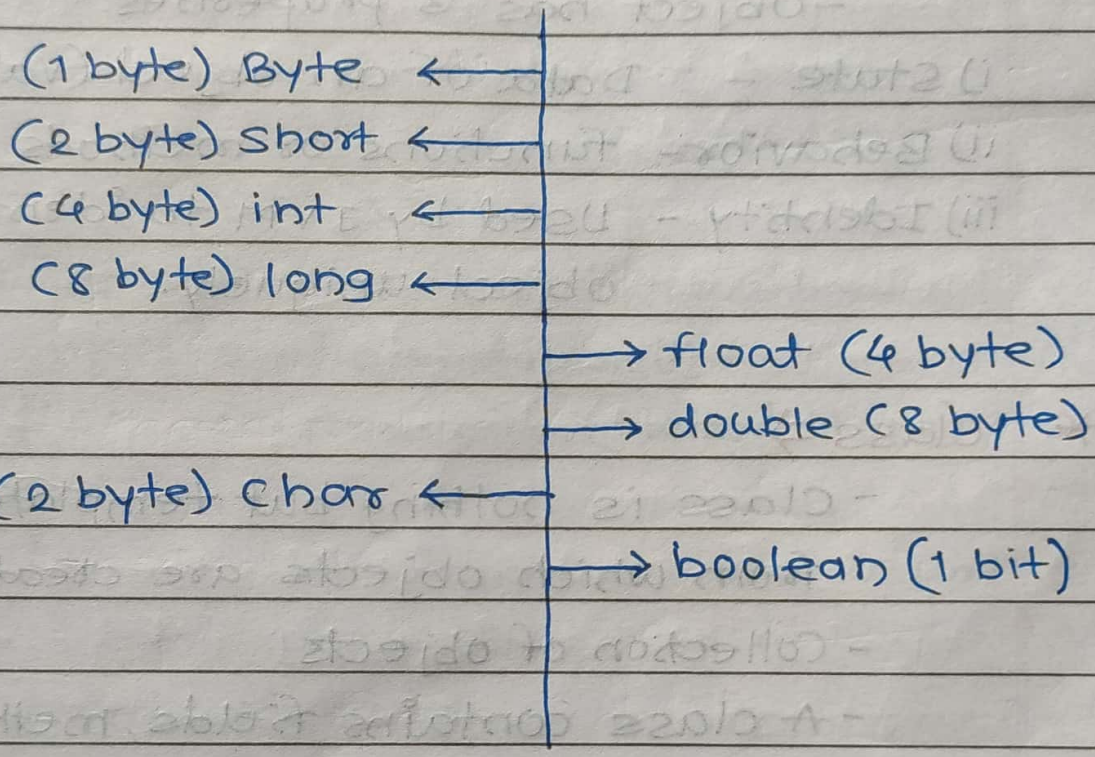
* Variable ⇉
- Variable is nothing but a piece of memory used to store the information.
- One variable can store one information at a time.
- To utilize variable in Java, we need to follow below steps ⇉
(Declaration, Initialization & Usage)

* Data types in Java ⇉
- Data types are used to represent type of data or information which we are going to use in Java program.
- It is mandatory to declare datatype before declaration of variable.
- Data types are of two types - Primitive & Non-Primitive

Non-primitive datatype ⇉
- These data types start with uppercase.
- Memory size is not fixed.
- These data types are also called identifiers
1) String
2) Class

Primitive datatypes ⇒

- Memory size is fixed.
- Primitive datatypes start with lowercase.
- Primitive datatypes are keywords.



PAGE No.
DATE / /

* Object ⇒

- An object is an instance of class which state & behaviour.
eg. chair, pen, etc.
- Object has 3 properties

i) State - Data of an object (value)
ii) Behavior - functions or methods
iii) Identity - Used by JVM to identify each object uniquely.

* Class ⇒

- Class is nothing but template or blueprint from which objects are created.
- Collection of objects
- A class contains fields, methods, blocks, constructors, etc.

* Method ⇒

- Methods are declared inside class body.
- Methods are used to perform code reusability operation.
- Inside class body programmer can declare no. of methods.

Methods are of two type-

1) Main ⇒

```
Public staic void main (string[] args)
    {

    }
```

2) Regular Method ⇒

A) Static Method ⇒

```
public static void test()
{                    (Method Name)

}
```

B) Non-static Method ⇒

```
public void test1()
{              (Method name)

}
```

- At the time of program execution, main method is going to execute automatically but regular methods are not executed automatically.
- To execute regular method, programmer need to make method call from main method.

* Method call from same class ⇒

1) To call static method,

methodname(); eg. test();

2) To call non-static method, we need to create object of class, by using

classname refname = new classname();
refname.methodname();

eg. demo d = new demo();
d.test();

* Method from diff. class ⇒

i) To call static method → classname.methodname()

ii) To call non static method →

classname refvariable = new classname();

Regular r = new regular
r.test1();

* Constructor ⇒

- constructor are special member of class.
- constructors are used to

i) Initialize data members

ii) To load non-static members of class into object.

- Constructor name is same as class name.
- We can declare any no. of constructor in class but constructor name should be same as class name & argument should be different.
- Constructors are two type

1) Default constructor
2) User defined

1) Default -

If constructor is not declared in class, then at the time of compilation, compiler will provide constructor for the class, that is known as default constructor.

2) User defined -

If programmer is declaring constructor in class, then it is called as user defined Constructor.

User defined constructors are of two type

i) Parameterized

ii) Without parameter

i) Parameterized constructor ⇒

- If you are passing some values while creation of object, it will call parameterized constructor.

- It is used to provide diff. values to diff. objects. eg. myclass (int a, int b) { }

ii) Without parameter -

When we are not passing any values then it is called as without parameter constructor.

```
eg. myclass ( ) {
    }
```

Ex.

```
Public class myclass {
    int id;
    String name;
myclass()
  {
    sop(id + "  " + name);
  }

myclass(int i, String n)
  {
    id = i;
    name = n;
    sop(id + "  " + name);
  }

Public static void main(String[] args){
    myclass obj1 = new myclass();
    myclass obj2 = new myclass(100, "Ram");
  }
}
```

o/p ⇒

```
0   NULL
100 Ram
```

DATE / /

* Control statements -

1) If statement ⇒

```
int marks = 50;
if (marks >= 35)
    { SOP ("Pass");
    }
```

2) If else ⇒

```
int marks = 50;
if (marks >= 35)
    { SOP ("Pass"); }
else
    { SOP ("Fail"); }
```

3) else if ⇒

```
int marks = 50;
if (marks >= 65)
    { SOP ("Distinction"); }
else if (marks >= 60 && marks < 65)
    { SOP ("first class"); }
else if (marks >= 40 && marks < 60)
    { SOP ("Pass"); }
else
    { SOP ("Fail"); }
```

4) Nested if ⇒

```
String username = "abc";
String password = "xyz";
if (username == "abc")
    { SOP(
        if (password== "xyz")
            { SOP ("Login successful"); }
        else
            { SOP ("Wrong password"); }
    }
  else
    { SOP ("Invalid username"); }
```

5) switch ⇒

```
int day = 2;
switch (day)
case 1 : SOP ("Monday");
case 2 : SOP ("Tuesday");
case 3 : SOP ("Wedensday");
case 4 : SOP ("Thursday");
case 5 : SOP ("friday");
case 6 : SOP ("saturday");
case 7 : SOP ("Sunday");
default : SOP ("Wrong input");
```

PAGE No.
DATE / /

* Loops ⇒

To perform same task multiple times.

1) For loop
2) While loop
3) Do loop
4) For each

1) For loop ⇒

```
# for (i=1 ; i<=10 ; i++)
    {
      sop(" value of i"+i);
    }
```

```
# for (i=1 ; i<=10 ; i=i+2)
    {
      sop(" value of i"+i);
    }
```

2) While loop ⇒

```
# int i=1;
While (i<=10)
    { SOP(i);
      i++;
    }
```

```
# int i=1;
While (i<=10)
    { SOP(i);
      i=i+2;
    }
```

PAGE No.
DATE / /

* Block ⇒

- Anywhere inside java program, open & closed brace { } represent the block.

- Blocks are used to initialize data members, declaring printing statement and also to declare executable statements.

	Static Block	Non-static Block
1)	Declaring a block with keyword static is considered as static block.	1) Declaring a block without any keyword is considered as Non-static block.
2)	Static blocks are used to initialize static variables only.	2) Used to initialize static as well as non-static variable.
3)	To access static block, we need to make class call i.e. classname.methodname();	3) To access non-static block, object creation is mandatory.
4)	Static blocks are going to get executed only once throughout lifetime of program.	4) It is going to get executed multiple times based on object creation.
5)	Static block are going to get executed before execution of main method.	5) Executed after execution of static block.

Ex.

```
Class block
 {
    {
      SOP (" Non-static Block");
    }
   static
    {
      SOP ("static Block");
    }
   PSVM (String[] args)
    { SOP ("Running main"); }
 }  block obj = New block ();
```

⇒ static Block
Non-static
Running Main

Ex.

```
Class demo2
 {
   static
      { SOP(" RSB"); }
      { SOP(" RNSB"); }
   demo2 ()
      { SOP (" constructor"); }

   Public void test ()
      { SOP (" Non-static method"); }
 }
```

```
class demo
{
  PSVM (string[] arg)
    {
      SOP("Main method");
      demo2 d2=new demo2();
        d2.test();
      SOP("_____");
      demo2 d2 = new demo2();
    }
  static
    {SOP("static block main class");}
}
```

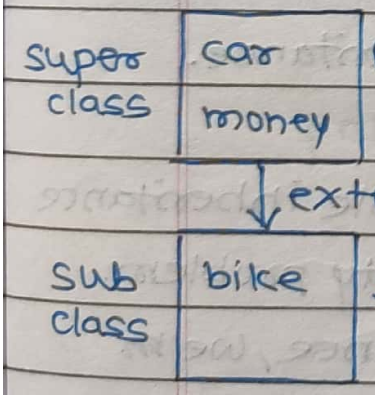
⇒ Static block main class
Main method
RSB
RNSB
constructor
Nonstatic method

RNSB
constructor

* Object is supermost class in Java.

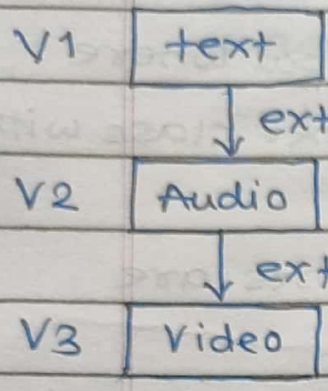
* OOPS ⇒
* Inheritance ⇒
  - It is one of the OOPS principal where one class acquire property of another class with the help of extends keyword.
  - The class from where properties are acquiring is known as super class.
  - The class where properties are inherited or delivered is known as sub class.
  - Inheritance takes place between two or more than two classes.
  - It is classified into 4 types ⇒

1) Single level Inheritance ⇒



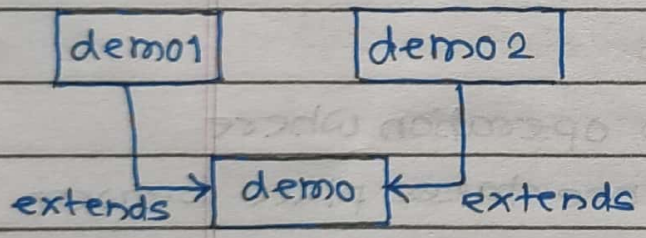
- It is an operation where inheritance takes place between two classes.
- To perform single level inheritance two classes are mandatory.
- Here sub class (son) acquires property of super class (father).

2) Multilevel Inheritance ⇒



- Multiple inheritance takes place between 3 or more than 3 classes.
- In multiple inheritance, one sub class acquire property of another super class & that class acquire property of another super class & phenomenon continues is known as multilevel inheritance.

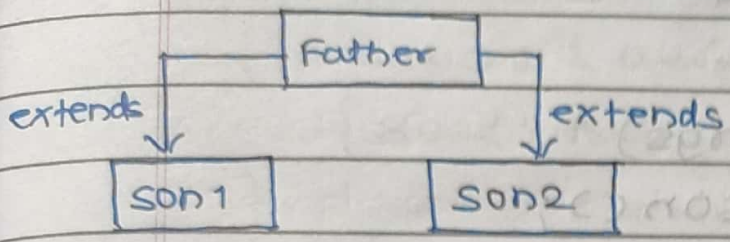
3) Multiple Inheritance ⇒



- One sub class acquires the property of two super classes at the same time is known as multiple inheritance.

- Java doesnot support multiple inheritance because of diamond ambiguty problem.
- To acheive multiple inheritance, we need to make use of interface.

4) Hybrid Inheritance ⇒



- Multiple subclasses can acquire property of one superclass is known as hybrid inheritance.

Ex. Single level Inheritance ⇒

```
package inheritance;
public class father
 {
    public void home()
       {SOP("home");}
    public void money()
       {SOP("money");}
    public void car()
       {SOP("car");}
 }

package inheritance;
public class son extends father
 { public void bike()
     {SOP("bike");}
 }
```

```
package inheritance;
public class_singlelevel
{
   psvm (string[] args)
   { son s = new son();
      s.bike();
      s.money();
      s.car();
      s.home();
   }
}
```

Result ⇒
bike
money
car
home

Ex. Multilevel inheritance ⇒

```
package inheritance;
public class v1
 { public void textmsg()
    { sop("text msg"); }
 }
```

```
package inheritance;
public class V2 extends V1
 { public void audioCalling()
    { SOP("Audio calling"); }
 }
```

```
package inheritance;
public class V3 extends V2
 { public void videoCalling()
    { SOP("Video calling"); }
 }
```

```
package inheritance;
public class multiLevel()
 {
   PSVM (String[] args)
   { V3 v = new V3();
     v.textMsg();
     v.audioCalling();
     v.videoCalling();
   }
 }
```

Result ⇒
text Msg
Audio Calling
Video Calling

Ex. Hybrid inheritance =>

```
package inheritance;
public class father ()
{ public void car()
    { SOP ("car"); }
  public void home()
    { SOP ("home"); }
}

public class son1 extends father
  { public void bike()
      { SOP ("bike"); }
  }

public class son2 extends father
  { public void mobile()
      { SOP ("mobile"); }
  }

public class hybrid ()
{ PSVM (String[] args)
  { son2 s2 = new son2 ();
    s2.car();
    s2.home();
    s2.mobile();
  }
}
```

Result =>
car
home
mobile

* Types of variables ⇒

Global variable ⇒

- Declaring variable outside the method or block is known as global variable.
- scope of global variable remains throughout the program.

Local variable ⇒

- Declaring variable inside the method or block is known as local variable.
- scope of variable remains within the method or block.

Class/Static ⇒

- Declaring variable with static keyword is known as static variable.
- To access static variable, class name is used.

Non-static / Instance variable ⇒

- All non-static variables are known as instance variables.
- To access non-static variable, object or instance need to be created.

Ex.

```
class sample
{
  int a = 10;
  int e = 50;
  static int d = 40;

  public void test()
  { int b = 20;
    sop(b);
    sop(a);
  }

  public void test1()
  { int c = 30;
    sop(c);
    sop(e);
  }

class sampletest
{ psvm(String[]arg)
  { sample s = new sample();
    s.test();
    s.test1();
    sop(sample.d);
    sop(s.e);
} }
```

* This Keyword ⇒

- This keyword is used to access global variable of same class.

Ex.

```
Class sample
{ int a = 30;
  public void test ()
  { int a = 20;
    SOP(a);
    SOP(this.a);
  }
}

class testsample
 {
   PSVM (String [] args)
    {
      Sample s = new sample ();
      s.test ();
    }
 }
```

Result ⇒

20

30

PAGE No.
DATE / /

* Super Keyword ⇒

-To access global variable from super class

ex.

```
class sample
{
  int a=30;
}

class sample1 extends sample
{ int a=20;
  public void test()
  { int a=10;
    SOP("local variable a" +a);
    SOP (this.a);
    SOP (super.a);
  }
}

class testSample
{PSVM (String[] args)
  {
    Sample s = new sample();
    s.test();
  }
}
```

Result ⇒

10
20
30

* Polymorphism ⇒

- One object showing different behaviour at different stages of life cycle is known as polymorphism.
- Polymorphism is classified into 2 types.

Compile Time polymorphism	Runtime polymorphism
1) Declaration & definations are binded during compilation based on arguments, is known as compile time polymorphism.	1) Declaration & definations are binded during run time/execution time based on object creation is known as run time polymorphism.
2) Also known as early binding	2) Also known as late binding
3) Method overloading is an example of compile time polymorphism.	3) Method overriding is an example of run time polymorphism.

Polymorphism ⇒ (Many forms)

(Poly - Many, Morphs - forms)

* Method overloading ⇒

- Declaring multiple methods with same method name but with different parameters (arguments) in a same class is known as method overloading.
- Different parameters in the sense of
  i) Either by changing no. of arguments.
  ii) By changing data type.

eg.
```
Class demo
{ public void addition (int a, int b)
    { int c = a+b ;
      SOP (c);
    }
  public void addition (int a, int b, int c)
    { int d = a+b+c;
      SOP (d);
    }
}

Class overloading
{ PSVM (String [] args)
    { demo d = new demo ();
      d.addition (5, 6, 7);
    }
}
```

Result
⇒ 5+6+7
⇒ 18

Def^{n} - Acquiring superclass property into sub class & changing implementation according to sub class specification is known as method overriding.

* Method over-riding ⇒

- If sub class has same method as declared in parent class, known as method overriding.
- Two classes which are in parent-child relationship (inheritance).
- Used to archieve run time polymorphism.

eg.

```
class father
{ public void money()
    { SOP ("100"); }
}

Class son extends father
{ public void money()
    { SOP ("50"); }
}

Class overriding
{
  PSVM (String [] args)
    { son s = new son();
      s.money();
    }
}
```

Result ⇒

50

PAGE No.
DATE / /

* Abstract ⇒

- A class declared with abstract keyword is known as abstract class, where programmer can declare complete as well as incomplete method.
- Incomplete method is a method where method declaration will be present & defn will be absent.
- We cant create object of abstract class.
- Programmer can declare incomplete methods by declaring abstract keyword in front of method.

eg.

```
abstract class demo
  {
    public void test()
      {SOP("complete method");}
    abstract public void test1();
  }
```

PAGE NO.
DATE / /

* Concrete class ⇒

- Programmer can't create object of abstract class. To create object of abstract class, there is approach called concrete class.

- A class which provides defination for all incomplete methods which are present inside abstract class is known as concrete class.

eg.

```
class demo1 extends demo
 { public void test1()
    { SOP("Incomplete method"); }
 }

class sample
{
  PSVM(String[] args)
    { demo1 d1 = new demo1();
      d1.test();
      d1.test1();
    }
```

PAGE No.
DATE / /

* Access specifier ⇒

- Access specifier are used to represent scope of member of class.
- In Java, access specifier are classified into 4 types.

Private, default, protected & public.

1) Private - If we declare any member of class as private, then scope of that member remains within the class. It can't be access from other classes.

2) Default - scope of member remains only within package. It cannot be access from other package. There is no keyword to ~~access~~ represent default access specifier.

3) Protected - scope of member remains within the package. The class present outside package can also access it by inheritance operation.

4) Public - Scope remains throughout the project.

* Interface ⇒

- Interface is one of the OOPS priciple.
- It is 100% abstract in nature.

* Features of interface -

1) Data members declared inside interface are by default static & final.
2) Method declared inside interface are public & abstract.
3) Constructor concept is not present inside interface.
4) We can't create object of interface.
   - To create object of interface, we need to make use of implementation class.
   - Interface supports multiple inheritance

* Implementation class ⇒

- Class which proride implementation or defination for all incomplete methods which are present inside interface with the help of implement keywords is known as implemenation class.
- At the time of implementation, declare method with public access specifier.

PAGE No.
DATE / /

Ex. Interface sample

```
{
    int a;
    void test();
    void test1();
}

Class sample1 implements sample
{
    public void test()
    { SOP("test"); }

    public void test1()
    { SOP("test1"); }
}

Class interfaceTest
{
    PSVM(String[] args)
    { sample1 s1 = new sample1();
        s1.test();
        s1.test1();
    }
}
```

* Casting ⇒

- Converting one type of information into another type is known as casting.

- In Java, casting is classified into two types - Primitive & Non-primitive casting.

1) Primitive casting -

- Converting one datatype information into another datatype is known as primitive casting

- It is classified into 3 types

a) Implicit casting - Converting lower datatype info into higher datatype info is called implicit casting

eg. int a = 10; → 4 byte

double b = a; → 8 byte

SOP(a); ⇒ 10.0

- It is also called as widening casting where memory size goes on increasing.

b) Explicit casting - Converting higher datatype info into lower datatype info is called explicit casting.

eg. double a = 2.5; → 8 byte

int b = (int)a; → 4 byte

SOP(b); ⇒ 2

- It is also called as narrowing casting where memory size goes on decreasing.

- In explicit casting, data loss takes place.

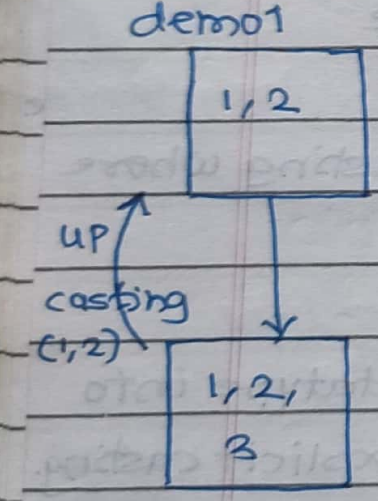
c) Boolean casting - It is considered to be as incompatible casting type because boolean datatype are unique type where information is already predeclared inside it.

2) Non-primitive casting -

- Converting one type of class into another type of class is known as non-primitive or class classing.
- It is classing into 2 types.

a) Up casting -

- Assigning sub class property to super class is known as up casting. Before up casting, we need to perform inheritance operation.
- After inheritance, properties of super class will be delivered to sub class.
- At the time of up casting, properties which are inherited from super class are only eligible for up casting operation.
- The new property declared in sub class are not eligible for up casting.



⊛ demo1 d = new demo2 ();

eg.
```
public class father
{ public void car()
    { SOP("car"); }
  public void home()
    { SOP("home"); }
}

class son extends father
{ public void bike()
    { SOP("bike"); }
}

class upcasting
{ psvm(String[] args)
  {
    father s = new son();
        s.car();
        s.home();
  }
}
```

b) Down-casting ⇒ Inherface
upcasting
down-casting

PAGE No.
DATE / /

* Generalization ⇒
  - Extracting all common properties from sub class & declaring it in super class and providing implementation according to sub class specification.
  - Generalization file can be java class or abstract class or interface. Recommonded is interface.

eg.
```
public interface simcard
  {
    void sms ();
    void audioCalling ();
  }

class idea implements simCard
  { public void sms ()
      { SOP ("100"); }
    public void audioCalling ()
      { SOP ("100"); }
  }

Class jio implements simCard
  { public void sms ()
      { SOP ("100"); }
```

```
    public void audiocalling()
        {SOP("150");}
}

class testInterface
{ psvm(String[] args)
    { idea i = new idea();
      i.sms();
      i.audiocalling();

    jio j = new jio();
     j.sms();
     j.audiocalling();
    }
}
```

* **Abstraction ⇒**

- Hiding implementation code & providing only functionality to end user is known as abstraction.
- The scenario of abstraction is if any customer is using any application then he should utilize functionality only. & he should not feel any backend code processing.

DATE / /

* String class ⇒
  - String is not primitive datatype.
  - Memory size of string is not fixed.
  - String is used to store collection of characters.
  - Object creation of string can be done in two ways.

a) Without using new keyword
b) Using new keyword
  - String objects are store inside string pool area which is present inside heap area.

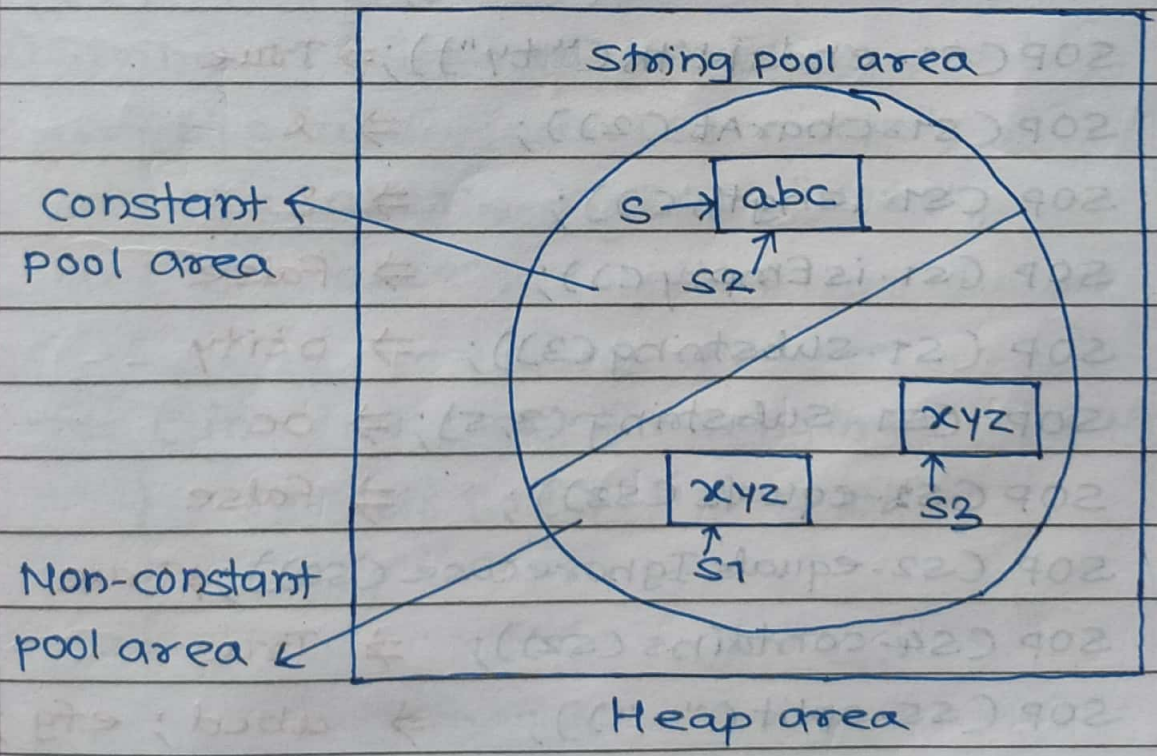
* String pool area ⇒
  - It is used to store string object.
  - String pool area is classified into two parts.

1) Constant pool area -
  - During object creation time, if we don't make use of new keyword, then object creation takes place inside constant pool area.
  - Duplicates are not allowed inside constant pool area.

2) Non-constant pool area -

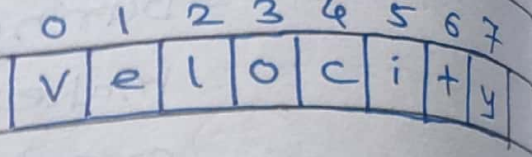
- During object creation time if we make use of new keyword then object creation takes inside non-constant pool area.
- Duplicates are allowed inside non-constant pool area.

eg. String s = "abc"; String s2 = "abc";
String s1 = new String ("xyz");
String s3 = new String ("xyz");



PAGE No.
DATE / /

* Methods of string class ⇒




```
String s1 = "Velocity";
String s2 = "abcd";
String s3 = "ABCD";
String s4 = "abcdefgh";
String s5 = "abcd efg hi";
```

1) SOP(s1.toUpperCase()); ⇒ VELOCITY
2) SOP(s1.toLowerCase()); ⇒ velocity
3) SOP(s1.indexOf("o")); ⇒ 3
4) SOP(s1.lastIndexOf("o")); ⇒ 3
5) SOP(s1.startWith("vel")); ⇒ True
6) SOP(s1.endsWith("ty")); ⇒ True
7) SOP(s1.charAt(2)); ⇒ l
8) SOP(s1.length()); ⇒ 8
9) SOP(s1.isEmpty()); ⇒ False
10) SOP(s1.substring(3)); ⇒ ocity
11) SOP(s1.substring(3,5); ⇒ oc
12) SOP(s2.equals(s3)); ⇒ False
13) SOP(s2.equalsIgnoreCase(s3)); ⇒ True
14) SOP(s4.contains(s2)); ⇒ True
15) SOP(s5.split(" ")); ⇒ abcd ; efg ; hi
16) SOP(s3.replace("C","A"); ⇒ ABAD
17) SOP(s2.concat(s1)); ⇒ abcdvelocity

* Array ⇒

- Array is data structure used to store collection of information.
- Array is homogeneous in nature (similar type of information can be stored.)
- Array declaration need to be done with capacity (size should be declared)
- Array is not growable in nature (size can't be increased.)
- Array is nothing but an object, inside object array index will be present.
- Array index starts from zero.

eg ① String ar[] = new ar[5];

```
ar[0] = "abc" ;  ar[3] = "xyz";
ar[1] = "abc1";  ar[4] = "xyz1";
ar[2] = "abc2";

for (int i=0 ; i<=ar.length-1 ; i++)
   { SOP (ar[i]);
   }
```

eg ② int ar[] = new ar[];

* Array sorting ⇒

```
int ar[] = new ar[5];
ar[0] = 10 ; ar[1] = 20;
ar[2] = 15 ; ar[3] = 30;
ar[4] = 25;
```

OR

```
int ar[] = {10,20,15,30,25};
```

```
for(int i=0 ; i<=ar.length-1 ; i++)
  { SOP(ar[i]); }

SOP("Ascending order ——");
Arrays.sort(ar);
 for(int i=0 ; i<=ar.length-1 ; i++)
  {SOP(ar[i]);}

SOP("Deseending order ——");
Arrays.sort(ar);
 for(int i=ar.length-1 ; i>=0 ; i--)
  {
    SOP(ar[i]);
  }
```

* Multidimensional array ⇒

```
int ar[][] = new ar[2][2];
ar[0][0] = 10;
ar[0][1] = 20;
ar[1][0] = 30;
ar[1][1] = 40;
```

```
int ar[][] = {{10,20},{30,40}};
```

```
SOP(ar[0][1]); ⇒ 20

for(int i=0; i<=1; i++)
 { for(int j=0; j<=1; j++)
    {
      SOP(ar[i][j]);
    }
    SOP();
 }
```

PAGE No.
DATE / /

* Exception Handling ⇒

* Exception -

During the exception of execution of Java program JVM faces abnormal situation based on the code declaration.

If JVM faces abnormal situation then JVM triggers an event, this event is known as exception.

If exception event got generated in Java program then it results in termination of Java program.

* Exception Handling -

Handling of the event generated by JVM during the program execution is known as exception handling.

We can handle the exception by using try & catch block.

```
try {// Risky code}
catch (exceptionName RefVariable)
    {
        // Event handled msg
    }
```

* Try Block -
  - It is used to declare risky code only.
  - Controller visits inside the try block only once throughout the lifetime of program.
  - Try block should be followed by either catch block or finally block.
  - Multiple try blocks are not allowed.

* Catch Block -
  - It is used to handle event generated from try block.
  - Catch block will get executed only if event generated in try block.
  - Catch block should be declared after try block.
  - Any no. of catch block can be declared for the single try block.

* finally block -
  - It is used to close the costly resources of the current program.
  - finally block should be followed by catch block.
  - We can declare finally block after try block but it is not recommended.

PAGE No.
DATE / /

* Throws keyword -
  - It is used to show or declare type of exception generated inside class or method.

* Throw keyword -
  - It is used to throw new custom exception.

* Collection ⇒

- Collection is readymade framework used to store collection of data of unique type as a single entity.
- Collection framework make use of container to store unique data.
- By using collection, data can be stored, modified and manipulated.
- There are some drawbacks in array, thats why collection framework was introduced.
- Collection interface is extended by 3 sub interfaces.

a) List b) set c) Queue

- All these sub interfaces are implemented by structured class i.e. ~~eacy~~ each & every implementation class has its own data structures.

a) List Interface ⇒

- It is sub interface of collection interface.
- List is used to maintain information in index basis.
- List allows duplicate values & any no. of null values.
- List is also used to maintain insertion order.

- List interface is implemented by 3 classes.

I) Arraylist II) Vector III) LinkedList

I) Arraylist ⇒

```
ArrayList Al = new ArrayList();
SOP(Al.isEmpty());
SOP(Al.size());
Al.add(20); Al.add("ABC");
Al.add('c'); Al.add("ABC");
Al.add(null);
SOP(Al); SOP(Al.size()); // Not recommended
SOP(Al.size());
SOP(Al.indexOf("ABC"));
SOP(Al.lastIndexOf("ABC"));
SOP(Al.get(2));
SOP(Al.contains('c'));
Al.add(2,"XYZ");
SOP(Al.get(2));
Al.set(2,"PQR");
SOP(Al.get(2));
Al.remove(2);
Iterator itr = Al.iterator();
While(itr.hasNext())
{ SOP(itr.next());
}
```

II) Vector ⇒

```
Vector V = new Vector();
SOP(V.size());
SOP(V.capacity());
SOP(V.isEmpty());
- - -
Enumeration En = V.elements();
while(En.hasMoreElements())
{ SOP(En.nextElement());
}
```

III) LinkedList ⇒

```
LinkedList ll = new LinkedList();
- - -
Iterator itr = ll.iterator();
while(itr.hasNext())
  { SOP(itr.Next());
  }
```

ArrayList	Vector
1) Arraylist is not a legacy class.	1) vector is a legacy class.
2) Arraylist uses resizable data structure	2) vector uses doubly data structure.
3) Incremental capacity = current capacity $\times \frac{3}{2} + 1$	3) Incremental capacity = current capacity $\times 2$
4) Arraylist is not synchronized & not thread safe.	4) vector is synchronized & thread safe.
5) Performance is high	5) Performance of vector is low.
6) Arraylist uses iterator cursor to traverse/ fetch/get elements. (Iterator & List iterator)	6) Enumeration cursor to traverse/fetch/get elements. (iterator & list iterator also)

Similarities ⇒

- Best choice for retrival because it is implemented by Random Access interface
- Worst choice for manipulation (insert & delete) because needs to perform several shifting operation either left/right.

ArrayList	LinkedList
1) Best choice for data retrival operation.	1) Worst choice for data retrival operation
2) Worst choice for data manipulation	2) Best choice for data manipulation.
3) Default capacity for arraylist is 10	3) No default capacity for linkedlist.
4) Resizable data structure.	4) Linear Data structure.

PAGE No.
DATE / /

* SET ⇒
- set is sub interface of collection interface.
- set dont allow duplicate values.
- set allow only 1 ~~value~~ null value insertion.
- No default capacity.
- set don't deals with array index operation, it deals with hashtable operation.

A) Hashset ⇒
- Hashset is implementation of set interface.
- Null insertion allowed (only one)
- Dont allow duplicate values.
- Order of insertion - Random insertion
- Storage type - hashtable
- No default capacity.
- Data structure - hashtable

When hashset is used?
1) To remove duplicate values.
2) If order of insertion is not mandatory.

eg.

```
Hashset sh = new Hashset();
    sh.add("Manual");
    sh.add("A");
    sh.add("B");
    sh.add("C");
    sh.add("E");
    sh.add("A");        → Duplicate "A" is removed.
    sh.add(null);

    SOP(sh);
    SOP(sh.size());
    sh.remove(null);
    SOP(sh.size());
    SOP(sh.isEmpty());
    SOP(sh.contains("M"));

    SOP("---Hashset Elements---");
    Iterator it = sh.iterator();
    while(it.hasNext())
    {
        SOP(it.next());
    }
    sh.clear();
    SOP(sh.size());
```

* LinkedHashset ⇒

- Linked hashset is an implementation class of set interface.
- Null insertion allowed (only one)
- Don't allows duplicate values.
- Order of insertion is maintained.
- Storage type - hashtable
- There is no default capacity.
- Data structure - Hybrid.

When LinkedHashset is to be used ?

- If duplicate not allowed
- Insertion order need to be maintained.

eg.
```
LinkedHashset lsh = new LinkedHashset();
lsh.add("B"); lsh.add("C"); lsh.add("A");
lsh.add("A"); lsh.add(null);
SOP(lsh);
SOP(lsh.size()); lsh.remove(null);
SOP(lsh.size()); SOP(lsh.contains("A");
SOP("---LinkedHashset Elements---");
Ide Iterator it = lsh.iterator();
    while(it.hasNext())
        { SOP(it.next()); }
    lsh.clear();
    SOP(lsh.size());
```

* TreeSet ⇒

- Treeset is an implementation class of shortedset interface.
- Null insertion not allowed
- Dont allows duplicate values.
- Order of insertion is Ascending order.
- Storage type - Hashtable
- No default capacity
- Data structure - Balanced tree.

	HashSet	LinkedHashSet	Treeset
1) Implementation class of	set	set	set
2) Null insertion	only 1	only 1	Not allowed
3) Duplicate values	Random	order maintained	Ascending order
4) Order of insertion	Not allowed	Not allowed	Not allowed
5) Data structure	Hashtable	Hybrid	Balanced tree
6) Storage type	Hashtable	Hashtable	Hashtable
7) Default capacity	No default capacity	No default capacity	No default capacity.

Java Programs

1) To draw pattern

```
* * *
* * *
* * *
```

⇒

```
public static void pattern1 ()
{
  for(int i=1 ; i<=3 ; i++)
    { for(int j=1 ; j<=3 ; j++)
      {
        system.out.print(" *");
      }
      system.out.println();
    }
}
```

2) To draw pattern

```
*
* *
* * *
```

⇒

```
public static void pattern2()
{
  int str=1;
  for(int i=1 ; i<=3 ; i++)
   {for(int j=1; j<=str; j++)
    {
      system.out.print("*");
    }
    str++;
    system.out.println();
   }
}
```

3) To draw pattern

```
    *
  * *
* * *
  * *
    *
```

⇒

```
public static void pattern3()
{
   int str=1;
   for(int i=1 ; i<=5 ; i++)
   {for(int j=1 ; j<=str ; j++)
     {
       system.out.print("*");
     }
      system.out.println();
     if( i <= 2)
          { str++;}
     else
          { str--;}
```

4) To draw pattern

```
*
*
*
*
```

⇒
```
public static void pattern4()
{ int str=1;
  int sp=0;
 for(int i=0;  i<=4 ; i++)
  {
   for(int j=1 ; j<=sp ; j++)
   {
     system.out.print(" ");
   }
  for(int j=1 ; j<=str; j++)
   {
     system.out.print("*");
   }
   sp++;
   System.out.println();
}
```

5) To draw pattern

```
    *
   *
  *
 *
```

⇒

```
public static void pattern5()
{
  int str=1;
  int sp=3;
  for(i=1; i<=4; i++)
  { for(int j=1; j<=sp; j++)
    {
      System.out.print(" ");
    }
   for(int j=1; j<=str; j++)
    {
      System.out.print("*");
    }
   sp--;
   System.out.println();
  }
}
```

6) To draw pattern

```
*
* *
* * *
* *
*
```

⇒ public static void pattern6 ()

```
{
  int str=1;
  for(int i=1; i<=5; i++)
  {
    for(int j=1; j<=str; j++)
    {
      system.out.print("*");
    }
    system.out.println();
    if(i<=2)
    {str++;}
    else
    {str--;}
  }
}
```

7) To reverse the string

⇒
```
public static void reverseString()
{
   string ori = "abcd";
   string rev = " ";
  for(int i = ori.length()-1 ; i >=0 ; i--)
   {
     reverse = reverse + ori.charAt(i);
   }
     System.out.println(reverse);

  if(ori.equals(reverse))
       { SOP(" string is pallindrome"); }
  else
       { SOP(" string is not pallindrome"); }
```

8) To check the number is Armstrong No. or Not.

Armstrong No. ⇒ 153 = $1^3 + 5^3 + 3^3$

⇒
```
public static void armstrong()
{ int num = 153;
  int sum = 0;
  int rem;
  for(int i = num; i > 0; i = i/10)
  {
    rem = i % 10;
    sum = sum + rem * rem * rem;
  }
  if (sum == num)
    { SOP (" Given No. is armstrong No.");}
  else
    { SOP (" Given No. is not armstrong No.");}
```

9) To reverse number oriNo = 12345;
revNo = 54321;

⇒
```
public static void reverseNumber()
{
  int oriNo = 12345;
  String ori = Integer.toString(oriNo);

  for(int i = ori.length()-1; i>=0; i--)
  {
    rev = rev + ori.charAt(i);
  }

  int revNo = Integer.parseInt(rev);
  System.out.println("Reverse No = "+revNo);
}
```

10) Fibonnaci series ⇒ 0, 1, 1, 2, 3, 5, 8, 13, 21, ...

⇒
```
public static void fibonnaciseries()
{
    int firstNo = 0;
    int secondNO = 1;
    SOP(firstNo+" ");
    SOP(secondNo+" ");
    for(int i=0; i<10; i++)
    {
        int thirdNo = firstNo + secondNo;
        SOP(thirdNo+" ");

        firstNo = secondNo;
        secondNo = thirdNo;
    }
}
```

⇒ 0 1 1 2 3 5 8 13 21 34 55

11) To check No. is prime or Not.

⇒
```
public static void primeNo ()
{
  int num = 7;
  int temp = 0;
for ( int i = 2 ; i < num ; i++)
{
    if(num % i == 0)
        { temp = temp + 1 ; }
}
if(temp == 0)
      { SOP (" Given No is prime No");}
else
      { SOP (" Given No is Not prime");}
```