⇒ JVM (Java Virtual Machine)

=

3 phases of a Program

1. Writing a Program (By Coders)
2. Compiling it by Javac (C in JDK)
3. Running the program, JVM executes the bytecode generated by compiler.

⇒ The Function of JVM is to execute the bytecode produce by javacompiler. Every OS has diff JVM but Output produced is after the execution remains same across all OS. Thus Java is Plat-form Independent.

⇒ Garbage Collector: JVM has a program called Garbage Collector. They can recollect the objects that are not referenced.

System.in → Standard Input Stream used to read characters from Keyboard.
System.out → Standard Output Stream used to produce result.

Q- Difference Between Java and C++

| | Java | C++ |
|---|---|---|
| 1 | Platform Independent | Platform Dependent |

| | |
|---|---|
| (ii) Memory Management System is Controlled | (ii) Management of Memory is Manual in C++. |
| (iii) It doesn't have Virtual keywords | (iii) It has Virtual keywords |
| (iv) Multiple Inheritance doesn't Supported | (iv) Multiple Inheritance Supported |
| (v) Operator Overloading X | (v) Operator Overloading - ✓ |
| (vi) MultiThreading - ✓ | (vi) MultiThreading - X |
| (vii) Pure OOP | (vii) Mixture of POP & OOP. |
| (viii) Structure & Union - X | (viii) Structure and Union - ✓ |

* Comments in Java

3 types
- → Single Line (// Comment)
- → Multi Line (/* Comment */)
- → Documentation Comment (/** Comment */)

* Access Modifiers.

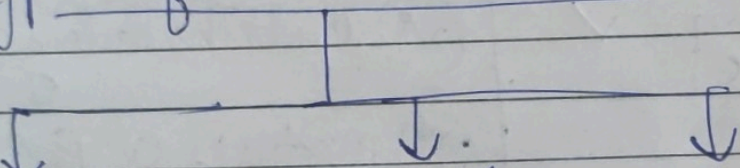
(i) default (ii) Public (iii) Protected (iv) Private

* Non-Access Modifiers

(i) final (ii) abstract (iii) strictfp.

| Access Modifier | Within class | Within Package | Out Package But in class | Out Package |
|---|---|---|---|---|
| Private | ✓ | × | × | × |
| Default | ✓ | ✓ | × | × |
| Public | ✓ | ✓ | ✓ | ✓ |
| Protected | ✓ | ✓ | ✓ | × |

* Types of Variable in Java




① Local Variables

→ A variable defined within a method, block or constructor is called Local Variable.
→ Created when Block is Entered
→ Destroyed when Block is Exited
→ Scope of Variable is within the block
→ Initialization is mandatory.

```
Eg class GFG {
        public static void main(String[] args)
        {
            int var = 10;
        }
}
```

Var is local

Q) Instance Variable : Instance Variables are the non-static variables and are declared in class outside of any method, constructor, or block.

→ Created when an object of Class is Created
→ Destroyed when an object of Class is deleted
→ Initialization is not necessary. Its default value is zero.
→ Instance variables can be accessed only by creating objects.

Eg
```
class GFG {

    public String Rishabh;

    public GFG()
    {
    this.Rishabh = "Soni";
    }

PSVM (String [] args)
  {
      GFG name = new GFG();
      S.O.P ("Name is" + name.Rishabh);
  }
}
```

O/P : Soni

∴ Hence, we declare Instance Variables inside a Class

## (3) Static Variables

→ Similar as Instance Variable.
→ In order to make any variable, Static we just need Static keyword.
→ Unlike Instance Variable, Static Variable has only One Copy.
→ Creation :- At the Execution of Program.
Deletion :- At the End of Program.
→ By Default Value → Zero.
→ It can be called directly, Compiler will automatically append the class name.

```
class GFG
{
    public static String geek = "Rishabh";

    public static void main (String [] args)
    {
        System.out.print ("Name :" + GFG.geek);
    }
}
```

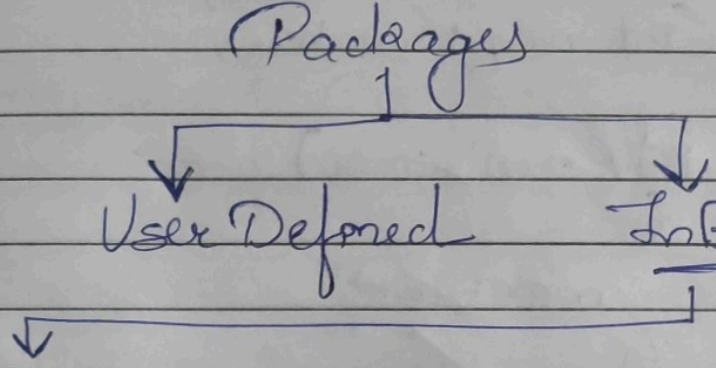
O/p : Rishabh.

## # Packages

→ Packages in Java is a mechanism to encapsulate classes, subpackages and Interfaces.
→ Uses of Packages :

(i) Data Encapsulation (Data Hiding)
(ii) Preventing Naming Conflicts.
(iii) Making Searching/locating & usage of classes, interface easier.

→ SubPackages: Packages that are inside another Package are the Subpackages. These aren't imported by default, they have to import explicitly.




(i) java.lang → Contains language support class. automatically imported.

(ii) java.io → Contains classes for input/output operations.

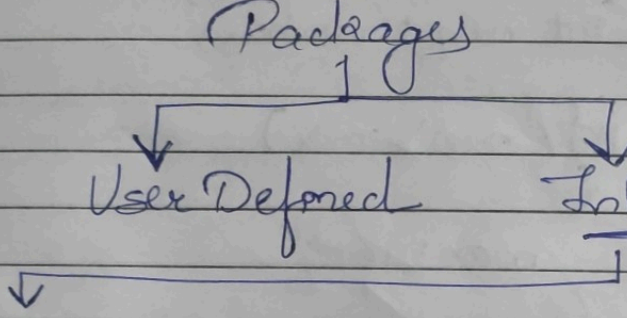
(iii) java.util → Contains utility classes which implements ds like LinkedList etc.

* For-Each Loop

Syntax: for (type var : array)
{
statements using var;
}

(i) Data Encapsulation (Data Hiding)
(ii) Preventing Naming Conflicts.
(iii) Making Searching / locating & usage of Classes, interface easier.

* SubPackages :- Packages that are inside another Package are the subpackages. These aren't imported by default, they have to import explicitly.




(i) java.lang → Contains language support class. automatically imported.

(ii) java.io → Contains classes for input/output operations.

(iii) java.util → Contains utility classes which implements ds like LinkedList etc.

* For-Each Loop

Syntax: for (type var : array)
{
statements using var;
}

Example

```
class For_Each {
public static void main (String [] args)
  {
    int [] marks = {125, 132, 95, 116, 110};
    int high_marks = maximum (marks);
    S.O.P ("" + high_marks");
  }
public static int maximum (int [] numbers)
  {
    int max = numbers [0];
    for (int num : numbers)
      {
        if (num > max)
          {
            max = num;
          }
        return max;
      }
  }
```

* Imp for MCQs

→ Diff. methods of String

(i) int length() - Returns number of characters in String.

Str = "GeeksforGeeks"

Str.length() = 13

(ii) char charAt(int i): Returns the character at $i^{th}$ index.

Str.charAt(3) = 'k' (Coz Strings start at index '0').

(iii) String substring (int i) : Return the substring from the $i^{th}$ index character to end.

Str.substring(3) = "ksforGeeks"

(iv) String substring (int i, int j) : Returns the substring from i to j-1 index

Str.substring (2,5) = "eks" (2,3,4)

(v) String concat (String str) : Concatenates specified string to the end of this string.

String s1 = "Rishabh"
String s2 = "Soni"
s1.concat(s2) = Rishabh Soni

(vi) int indexOf (String s) : Returns the index within the string of the first occurrence of the specified string.

String s1 = "CHARUSAT University"
0 1 2 3 4 5 6 7 8 9

s1.indexOf ("University") = 9.

(vii) int indexOf (String s, int i) : Returns the index within the string of the first occurrence of the specified string.

String s = "Rishabh Soni"
0 1 2 3
s.indexOf ("sh", 2) → (3).

(VIII) Int lastIndexOf(String s).

String s = "Learn Share Learn"
s.lastIndexOf("a") = (4)

(IX) boolean equals(Object otherObj)

Boolean out = "Geeks".equals("Geeks") ← True
Boolean out = "geeks".equals("Geeks") ← False

(X) boolean equalsIgnoreCase(String anotherString)

↳ "Aama small/big banera chale"

Geeks = Geeks ✓ true
geeks = Geeks ✓ true.

(XI) int compareTo(String anotherString)

↑
used to compare two Strings

int out = s1.compareTo(s2);

returns difference s1-s2 If:
out < 0 // s1 < s2
out = 0 // s1 = s2
out > 0 // s1 > s2.

(12) StringtoLowerCase() → Converts all characters to lower case

(13) StringtoUpperCase() → Converts all characters to Upper case.

(4) String replace (char oldchar, char newchar)

String s1 = "feelsforfeels";
String s2 = "feelsforfeels".replace ('f', 'g')

s1 = "feelsforfeels"
s2 = "geelsgorgeels"

* Methods in Java

Syntax :

<access modifier> <return type> <name of method> <parameter>

{
// body
}

Advantage : Code Reusability
Code Optimization

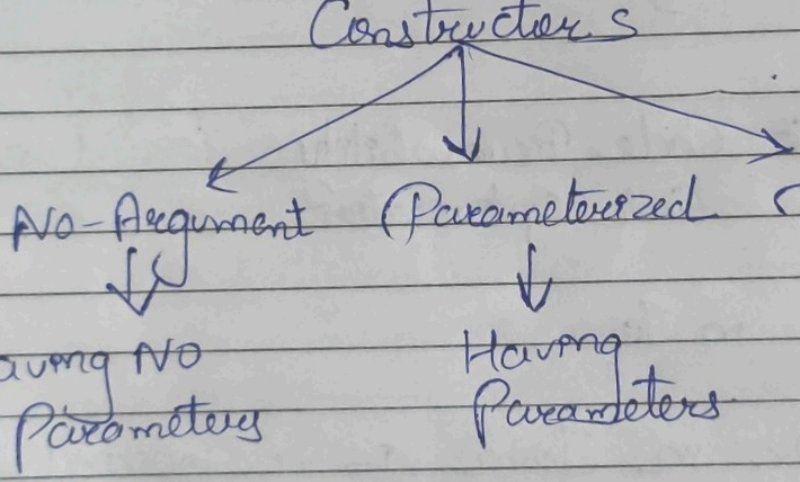
* Constructors in Java

→ A constructor in Java is a special method that is used to initialize objects.

→ Constructors have same name as that of class
→ Constructor don't have any return type.
→ Constructor can be called only once i.e. @ time of Object Creation.

E.g. class Reshabh
{
---
---
Reshabh() // Constructor
{
}
---
}
Reshabh r = new Reshabh(); // Constr called.

⇒ Constructor can't be abstract, final, static or synchronized.




* Constructor Overloading (Program).

Reshabh()
Reshabh(25)
Reshabh("hello", 5)
} If we use these three in a program it's called Constructor Overloading

⇒ Copy Constructor : Same as C++

* Constructor Chaining in Java

⇒ Constructor Chaining is the process of calling one Constructor from another constructor wrt current obj.

⇒ It can be done in 2 ways :-

(i) Within Same Class : Using this()

(ii) From Base Class : Using super().

Example

```
class Temp
{
    Temp()
    {
        this(5); // invokes C-2
        S.O.P("Default Called");
    }
    Temp(int x) // invokes C-3
    {
        this(5,15);
        S.O.P(x);
    }
    Temp(int x, int y)
    {
        S.O.P(x*y);
    }
    psvm(S Args)
    {
        Temp t = new Temp(); // invokes default
    }
}
```

O/P

d

75

5

Default Called

* Rules for Chaining

(i) The this() should be always used in first line of Constructor

(ii) There should be atleast one constructor without this() (for e.g. Constructor 3)

* Inheritance

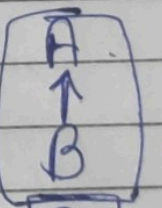
→ Need for Inheritance :- (i) Code Reusability
(ii) Method Overriding
(iii) Abstraction.

Syntax :

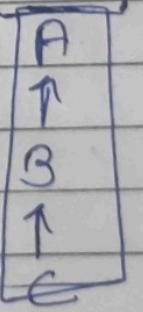
```
class derived_class extends base_class
{
  //----
}
```

Types

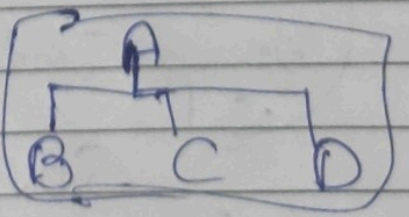
(i) Single Inheritance →




(ii) Multilevel Inheritance →




(iii) Hierarchical




(iv) Multiple - X Not Supported

Can be Achieved via Interfaces.

* (Polymorphism (Ability to take different forms)

Compile time Polymorphism → (Method Overloading)

Run time Polymorphism → (Method Overriding)

* Compile time

Method Overloading & Same Name different Parameters

Eg class Rishabh {

```
    static int Multiply (int a, int b)
    {
        return a*b;
    }
    static double Multiply (double a, double b)
    {
        return a*b;
    }
}
class RS {
    public static void main (String[] args)
    {
        S.O.P (Rishabh.Multiply(5,10));
        S.O.P (Rishabh.Multiply(2.5,2.0));
    }
```

O/P - 50
5.0

* Run Time (Method Over Riding)

↑

Same Name, Same Parameters

```
class Parent {
    void print()
    {
        S.O.P("parent class");
    }
}
class subclass1 extends Parent {
    void print()
    {
        S.O.P("subclass 1");
    }
}
class subclass2 extends Parent {
    void print()
    {
        S.O.P("subclass 2");
    }
}
class OFG {
    psvm(String[] args)
    {
        Parent a;
        a=new subclass1();
        a.Print();
        a = new Subclass2();
        a.Print();
    }
}
```

O/P

subclass 1

subclass 2

| Method Overloading | Method OverRiding |
|---|---|
| (i) Compile Time Polymorphism | (i) Run Time |
| (ii) Within the class | (ii) Not Within Class |
| (iii) Not Require Inheritance | (iii) Requires Inheritance |
| (iv) Same Name allowed but Same Parameter is not allowed | (iv) Same Name as well as same Parameter both allowed. |
| (v) Static Binding | (v) Dynamic Binding |
| (vi) Private and final can be Overloaded | (vi) Private and Final can't be Overloaded |

# Keywords in Java

(1) Super :- The super keyword in java is a reference variable used to refer to parent class objects.

1. Use of super with variables

It will call parent's class variable.
for e.g.

```
class Parent {
    x = 20;
}
```

```
class Child extends Parent {
        int x = 30;
    }
class Main {
        S.O.P(x); ← Child Class
        S.O.P(super.x); ← Parent class
    }
```

## (2) Use of Super with methods

⇒ This is used when we want to call the Parent class method.

⇒ So whenever a parent and child class have same named methods then to resolve ambiguity we use the super keyword.

```
class Person {
        void message()
        {
            S.O.P ("Person class");
        }
}
class Student extends Person {
        void message()
        {
            S.O.P("Student class");
        }
        void display(){
        {
            message(); ← Its own
            super.message(); ← Parent Method.
        }
```

## ③ Use of Super with Constructor

- To access Parent class Constructor

```
class Person {
        Person()
        {
            S.O.P("Person class");
        }
}
class Student extends Person {
            Student
            {
                Super();
                S.O.P("Student class");
            }
}
class Test {
        PSVM(SA)
        {
            Student s = new Student;
        }
}
```

O/P
- → Person class
- → Student class

Note: Call to Super() must be the first statement in derived class because Superclass has no knowledge about subclass, So any initialization it needs to perform is separate from & possibly prerequisite to

any initialization performed by the subclass.

* Final Keyword in Java (Non-Access Modifier)

Final Variable → To Create Constant Variable.
Final Method → Prevent Method Overriding
Final Class → Prevent Inheritance

* Abstract Keyword

→ Abstract is a non-access modifier applicable for classes, methods but not variables.
→ It's used to achieve abstraction.

→ Abstract Classes : The class having partial implementation (i.e. not all methods present in class have method definition).

```
abstract class class-name {
    // body of class
}
```

⇒ we can't create obj of abstract class.

Eg → java.lang.Number (Abstract Class)

⇒ Abstract Methods : The methods which just have declaration in super class & implementation in subclass.

Note : Final & Abstract aren't used simultaneously.

Note:- Any class that contains one or more abstract methods must be declared abstract.

* Static Keyword

(i) Blocks → executed first
(ii) Variables → Only one Copy Shared
(iii) methods → need no object to call
(iv) Classes → Only Nested Class can be Static

* This Keyword

(1) This keyword is used to refer current class instance variables

(2) To invoke current class constructor

(3) return this
↳ returns current class instance.

(4) display (this)
↳ used as parameter.

(5) this.display()
↑
use ref this to invoke current class method

# * Interfaces

=> An Interface in Java is defined as an abstract type used to specify the behaviour of class.

=> Interface = abstraction

Syntax: interface {
// methods (abstract)
}

=> A class that implements an interface must implement all the methods declared in Interface.

Uses of Interface: (i) Total Abstraction
(ii) Multiple Inheritance
(iii) Loose Coupling

Q- Why to use Interface when we can make abstract class?

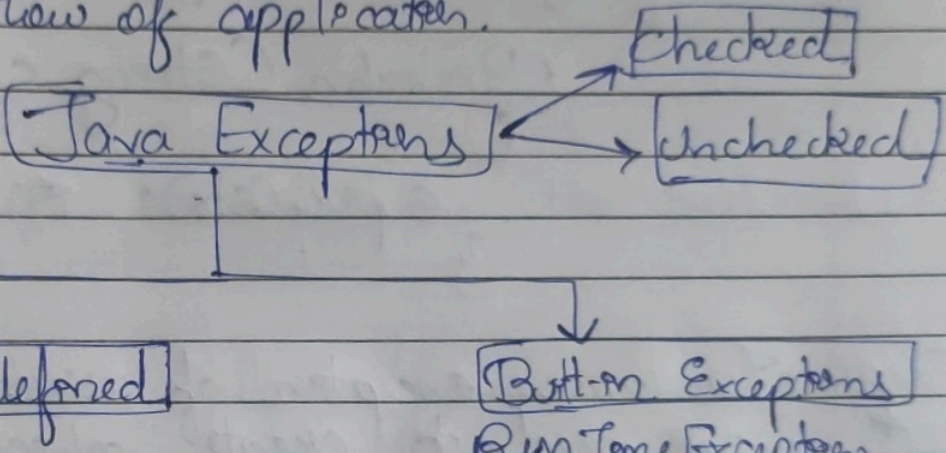
→ Abstract class may contain non-final variables, whereas variables in interface are final, public and static.

| Class | Interface |
|---|---|
| (i) You can instantiate variable & can create Object | (i) You can't |
| (ii) Normal/Concrete Methods | (ii) Abstract Methods |
| (iii) Maybe Public, Private, Protected | (iii) Only Public. |

# Exception Handling

⇒ Exception is an unwanted or unexpected event which occurs during the execution of program.

⇒ Exception Handling means to handle the run time errors in order to maintain regular flow of application.




→ Checked

- → ClassNotFoundException
- → InterruptedException
- → IOException
- → SQLException
- → FileNotFoundException

→ Also called Compile Time Exception

→ Arithmetic Exception
→ ClassCastException
→ NullPointerException
→ IllegalThreadStateE
→ ArrayIndexOutOfBound

Q- How to Create User-Defined Exception

Step 1: Create a subclass of Exception class

For e.g. class Rishabh extends Exception.

Step 2: Write a default constructor in your own class

```
Reshabh()          or    Reshabh(String str)
{                        {
}                        }
```

Step 3 If you create parameterized then

```
Reshabh(String str)
{
    super(str); ← Must write.
}
```

Step 4 To raise exception of user-defined type, we need to create object.

```
Reshabh obj = new Reshabh("details");
throw obj;
```

* Keywords (try, catch, throw, throws, finally)

① try: The try block contains a set of statements where an exception can occur.

```
try
{
    //Statement(s) that might cause exception
}
```

② catch: The catch block is used to handle the uncertain condition of a try block. A try block is always followed by a

catch block, which handles the exception.

```
catch
{
    // Statement(s) that handle exception
    // for e.g. exiting file
};
```

(3) throw :- The throw keyword is used to transfer control from the try block to catch block.

→ Throw Manually creates Exception Object
→ It's Useful on Userdefined Exceptions.

(4) throws :- The throws keyword is used for exception Handling without try and catch block. It specifies the exceptions that a method can throw and doesn't handles itself.

(5) finally :- It's executed after catch block. Finally is executed even if Exception Occurs or Not.

# Throw and Throws

→ Throw :- Used to explicitly throw exceptions

Syntax :- throw Instance

Example throw new Arithmetic Exception.
↑ to create Instance.

→ throws :

Syntax : type method_name (parameters) throws exception_name.

Eg public void Rishabh(int a) throws IOException

| Throw | Throws |
|---|---|
| (1) Used to Throw Exception | (1) Used to Declare Exception |
| (2) Can throw unchecked exception only | (2) Throws used with both. |
| (3) followed by instance name | (3) followed by class names |
| (4) within method | (4) With the method Signature |
| (5) Only Single Exception throw | (5) Throws can use Multiple Exception |
| (6) Eg throw new A.E | Eg throws IOExcep, A.E. |

* Basic Pgm of User Defined Exception

```
class Rishabh extends Exception {
        public Rishabh (String s)
        {
            super(s);
        }
}
```

```
public class Main {
    psvm (String args[])
    {
        try {
            throw new Reshabh("CSE");
        }
        catch (Reshabh r) {

            S.O.P ("Caught");
            S.O.P(r.getMessage());

        }
    }
}
```

O/P

Caught
CSE