① Variable :- Variable is a container used to store the data

- local → define inside block (or) function body
- member → define inside class body
  - Static
  - Non-Static

Ex:

```
Class A
{
    static int x;   NV -> Globally Every where
    int y;
    void display(int j)
    {
        int j   -> L.V. locally (block only)
    }
}
```

② Data type :- It is used specify type of data stored in the variable.

- Primitive
  - Keyword
    - 8
- Non primitive
  - Class / Interface
    - 'n' number of

Datatype	M.	Default value
byte	1 → 8b	0
short	2 → 16b	0
int	4 - 32b	0
long	8 - 64b	0
float	4 - 32b	0.0
double	8 - 64b	0.0
char	2 - 16b	unique value (or) non printable value.
boolean	1 - 8b	false

	Data type	Memory Storage		Default value
		byte	bit	
1)	Byte	1	8 bit	0
2)	Short	2	16 bit	0
3)	Int	4	32 bit	0
4)	long	8	64 bit	0
5)	float	4	32 bit	0.0
6)	double	8	64 bit	0.0
7)	char	2	16 bit	unique value / non printable value.
8)	boolean	1	8 bit	T/F

3) Default value:- If we not initialization value compiler through default value.

Default value is applicable for member variable not local variable.

```
class Pan
{
  static int cost;
  PVSTM (---) {
    S.O.P (cost); //0
    int x;
  } S.O.P (x); // CTE
```

4) This keyword:- If the local & instance variable is same in order to differentiate then we use this keyword.

-> point to current object

5) Diffrent static & non static.

	Accessing in Same class	Accessing in Diff class	memory location	No of copy
Static	Directly (or) Classname	Class name	Class Area (or) Static pool	Only 1 copy / class
non static	Object creation	Object creation	Heap Area	Multiple copies, Depend upon no of copies object

Example:- Accessing static and non-static variables in same class.

```
Package Spiders;
{
 class car
 {
   static int cost = 100;
   String brand = "Suzuki";
  Public string void main (string [] args)
  {
    S.O.P (cost + " " + car.cost);
    car C = new car();
  } S.O.P (C.brand);
```

O/P 100 100
Suzuki

2) Access Static & non Static member (variable & me
in different class.

```
Package com;
class Employee
{
  Static Int id = 101;
  String name = "TOM";

  Static void work(){
    S.O.P ("Working");
  }
  void eat()
  {
    S.O.P ("Eating");
  }
Package com;
class Solution
{
  Public Static void main(String[]args)
  {

    S.O.P (Employee.id)
    Employe.work();
    Employee emp = new Employee();

    S.O.P (emp.name);
    Emp.eat();
  }
}
```

O/P 101
Working
TOM
Eating.

1) Widenning
2) narrowing
3) up casting
4) down casting
5) Generalization
6) Specialization
7) Classcast Exception
8) instance of.

Type casting :- Converting one data type to another data type.

There are two type of typecasting

1) Primitive data type
2) nonprimitive data type

1) Primitive data type

i) Widening
ii) narrowing

i) Widening :- It is used to converting Smaller data type (or) Lower data type to Bigger data type (or) higher data type.

* Implicit

ii) ~~Narrowing :- It is used to converting Big~~

Ex :-

① int x = 10;
double y = x;
(10)

x 10
y 10.0

② double a = 25
S.O.P (a) o/p : 25.0

③ char x = 'A';
int x = x;
('A')

o/p x = A
y = 65

ii) Narrowing :- converting Bigger data type to smaller data type.

-> "Explicit"

Ex ① double a = 5.7;

int b = (int) a;

S.O.P(b)

O/P = b = 5

② int ? = (int) 67.8

S.O.P(?)

O/P = 67

③ int x = 98;

char y = (char) x;

98

S.O.P(y)

O/P Y = b

② Non-primitive data type

i) upcasting

ii) Down casting

i) up casting :- Super class reference and Sub class object

(or)

-> The process of create of an object of subclass & storing its address into a reference of type Superclass.

-> upcasting also be reffered super class reference and subclass object.

ii) -> With the upcasted reference we can access only superclass member (only superclass member are visible)

iii) -> upcasting happens "Implicitly"

iv) -> For achieve upcasting Inheritance is mandatory (IS-a Relationship)

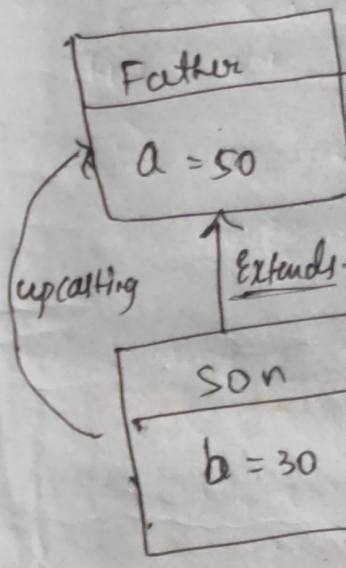
Down casting

1) -> The process back to

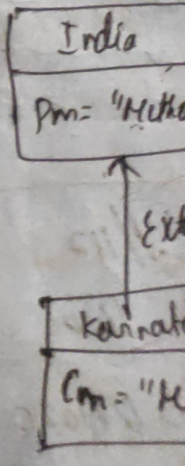
2) -> With the down we can bac

3) -> Down casting

4) -> For achi



Ex:- For upcast



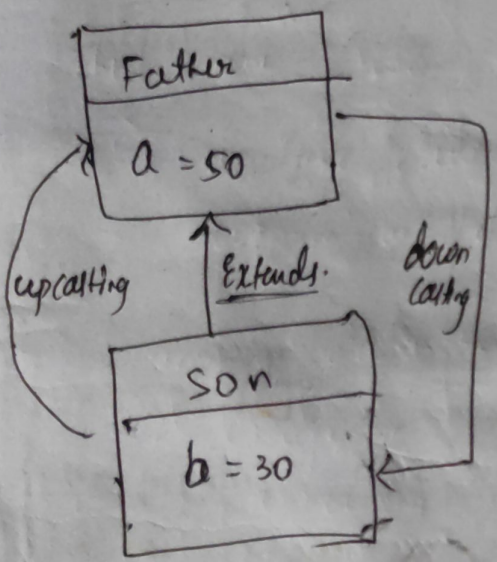
down casting

1) → The process of converting the upcasted reference to back to subclass type reference is called down casting.

2) → With the down casted reference (or) subclass reference we can access both super class members & sub class members. (Both super & subclass members are visible)

3) → Down casting happens Explicitly.

4) → For achieve down casting, "upcasting is Mandatory"



Ex:- upcasting → implicity.

Father f = new Son();
object

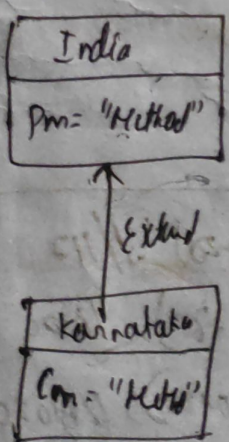
a ✓ b visibility. ✗

Ex: down casting → Explicity.

Son s = (Son) f

a ✓ b visibility ✓

Ex:- For upcasting & down casting.



Ex: upcasting → implicity.

India i = new karnataka();

(or)

karnataka k = new karnataka();

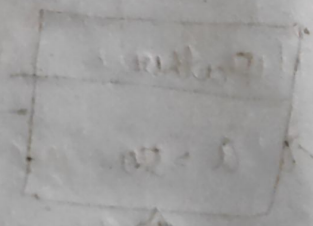
India i = k;

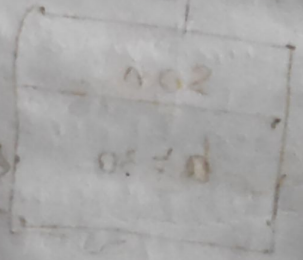
Ex:- down casting → explicity.

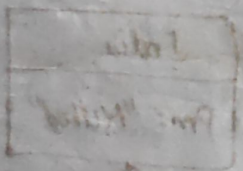
karnataka k = (karnataka) i

Eg:- For widening & narrowing

```
Package com;
class Test {
    Public static void main(String[] args)
    {
        // Widening
        int a = 10;
        double b = a;
        S.O.P (a+" "+b); // 10 10.0
        char c = 'A';
        int d = c;
        S.O.P (c+" "+d); // A 65
        // Narrowing
        double f = 56.7;
        int g = (int) f;
        S.O.P (f+" "+g); // 56.7 56
        int x = 101;
        Char y = (char) x;
        S.O.P (x+" "+y); // 101 e
        S.O.P ((int) 78.9); // 78
        S.O.P ((char) 68); // D
    }
}
```







S.O.P ("A"+"B"); AB

S.O.P ("A"+20); A20

S.O.P ('A'+'B'); 131

S.O.P (A+20); //117

S.O.P (10+20+"C"); //30C

S.O.P ("A"+10+20); //A1020

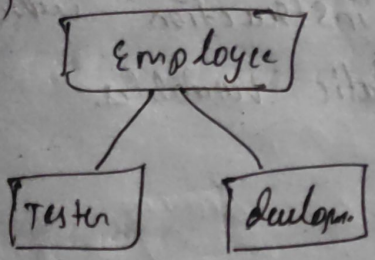
Specialization

If one reference variable storing only one specific type of object then called Specialization.

Generalization

If one reference variable storing different type of object then called Generalization

Ex!-(1)



Object creation

```
Developer d1 = new Developer }
Tester    t1 = new Tester    } Specialization
```

```
Employee e = new Developer }
             new Tester    } Generalization.
```

Method for Object

```
void work (Developer d)  } Specialization
          (Tester t)     }
{
}

void work (Employee e) Generalization.
{
}
```

work (new Developer())

work (new Test())

Blocks

Block are a set of instruction (or) a block of code which is used for initilization

Block are classified into 2 types

1) Static blocks
2) Non static blocks

Static Blocks

Static Block are a set of instruction which is used for initializing static variables.

Syntax :-

```
static
{
  ---
  ---
  ---
}
```

→ Static Block will get Executed before main method (class loading time)

→ When we have multiple static blocks's, the Execution happen in a sequential.

Ex:-1

```
Package com;
  class car
  {
   static {
     S.O.P ("In static Block-1");
   }
  Public static void main (String[] args)
      {
        System.out.println ("Hello");
      }
  static
  {
    S.O.P ("In static Block-2");
  }
```

```
Static
{
  S.O.P ("In Static Block-3");
}
}
```

Output

In Static Block - 1

In Static Block - 2

In Static Block - 3

Hello.

Ex: 2

```
Package com;
class Student
{
  Static int id;
  Static
  {
    Id = 100;
  }
Public Static void main (String[] args)
  {
    S.O.P ("Id: " +id);
  }
Static
  {
    Id = 200;
  }
```

Output

Id: 200

Non-Static Block

-> Non-Static Block are a set of instruc which used for initialization both static & non-static variable.

But majorly used for non-static vari

Syntax

```
{
  ---
  ---
  ---
}
```

-> Non static block get executed during object creation [Instantiation]

-> It is possible to having multiple non static Block and execution happen in a sequential order.

Ex:-1

```
class pen {
{
  S.O.P ("In non-static block-1");
}
Public Static void main (String [] args)
{
  S.O.P ("start");
  Pen = new pen();
  S.O.P ("End");
  new pen();
}
```

```
    {
      S.O.P("In non-static block-2");
    }
}
```

O/P

```
Start
In-non static block-1
In-non static block-2
End
In-non static block-1
In-non static block-2
```

Ex-2

```
Package com;
Class Employee
{
  int id;
  {
    id = 101;
  }
Public static void main (String[] args)
  {
    Employee Emp = new Employee();
    S.O.P ("Id:" + id);
  }
  {
    id = 301;
  }
}
```

O/P

Id: 301

NOTE: During object creation non-static Block will executed first and then constructor.

Ex:-3

```
Package com;
class Demo
{
 static {
  S.O.P ("static block"); //Exection 1
 }
Public static void main(String [] args)
  { new Demo();
  }
Demo()
 {
  S.O.P ("constructor"); // Exection 3
 }
{
 S.O.P ("non-static block"); // Exection 2
}
}
```

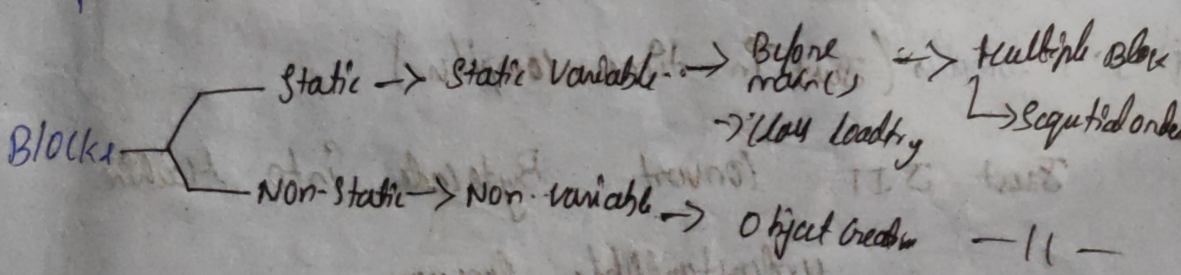
O/P

static Block
non-static Block
constructor.

Eg:-4

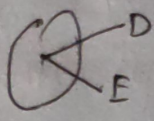
```
Package
  class Bike
  {
    static int cost;
      String brand;
  }
  static
  {
    cost = 100;
  }
Public static void main (String[] args)
  {
    S.O.P ("cost:" + Bike.cost); //100
    Bike b = new Bike ();
    S.O.P ("Brand" + b.brand); // Suzuki
    S.O.P ("cost" + Bike.cost); // 200
  }
  {
    cost = 200;
    brand = "Suzuki";
  }
```

Summary



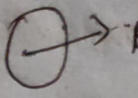
JDK :- (Java Development kit)

⇓

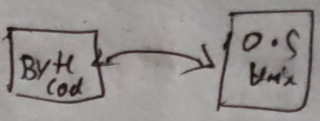
Softwar  → JP

JRE (Java Runtime Environment)

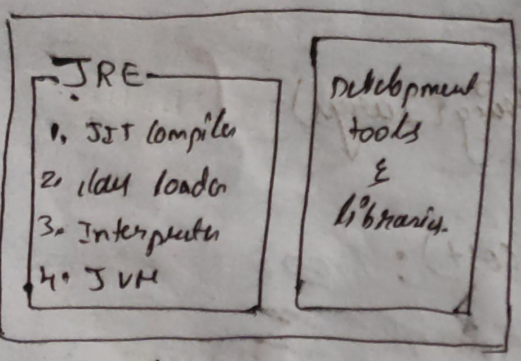
⇓

Softwar  → J.P

JIT (Just in time)



JDK



JDK :- (Java Development kit)

JDK is Software which contains all the resources used for Developing & Executing Java Program.

JRE :- (Java Runtime Environment)

JRE is Software which provides a platfor (or) Environment for Executing Java program.

i) JIT compiler (Just In time compiler)

~~Just~~ JIT convert Bytecode into Machine understandable language.

ii) Class loader

loades class from secondary storage to Executable area.

iii) Interpreter

Interpeter Execute the program line by line.

iv) JVM (Java Virtual machine)

~~Java~~ JVM is the manger of the JRE

⇒ It is responsible for converting source code to byte code.

ClassCastException :-

-> If object is upcasted, we have to downcast to the same type otherwise we get "class cast exception".

-> If one object is upcasted and downcasted some other type we get "ClassCastException"

-> We might also get ClassCastException when we downcast without upcasting.

-> ClassCastException can be avoided with the help of "instanceof" operator

instanceof :-

-> instanceof is used to check if an object is having the properties of a class (or) not

-> The return type of instanceof is boolean

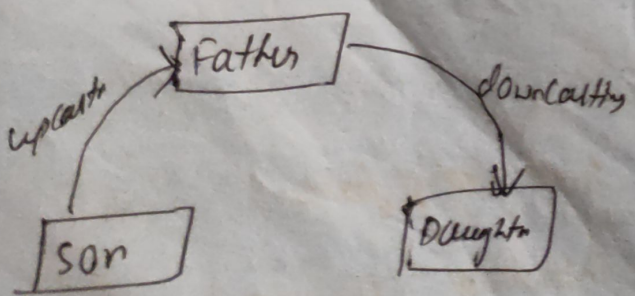
Syntax :-

Object instanceof ClassName

(or)

new ClassName() instanceof ClassName

ClassCastException



Father f = new Son();

Daughter d = (Daughter) f;

Program:-

```
Package .com;
Class Father
 {
  Int X=10; // Only 1 mentu
 }
Class Son Extends Father
{
 Int Y=30; // 2 mentu (X & Y)
}
Class Daughter Extends Father
 {
  Int Z=10; // 2 mentu (X and Z)
 }
Package .com;
Class Test
 {
  Public static void main (String[] args)
  {
   Father f = new Father();
   Son S = new Son();
   Daughter d = new Daughter();
S.O.P (S instanceof Son); // true
S.O.P (S instance of Father); // true
S.O.P (d instance of Daughter); // true
S.O.P (d instanceof Father); // true
S.O.P (f instanceof Daughter); // false
S.O.P (new Daughter() instanceof father); // true
S.O.P (new Father() instanceof Daughter); // false.
 }
}
```

Program 2

```
Package com;

class Father
{
  int x=10;
}

class Son Extends Father
{
  int y=20;
}

class Daughter Extends Father
{
  int z=30;
}

Package com;

{
  // Father obj = new Son(); (or) Father obj = new Daughter();
  Static void display (Father obj)
  {
    if (obj instanceof Son)
    {
      S.O.P ("Downcasting to Son");
      Son s = (Son) obj;
      S.O.P (s.x + " " + s.y);
    }
    else if (obj instance of Daughter)
    {
      S.O.P ("Downcasting to Daughter");
      Daughter d = (Daughter) obj;
      S.O.P (d.x + " " + d.z);
    }
  }
```

```
Public Static void main (String[] args)
{
    display ( new Son());
    display ( new Daughter());
}
```

o/p

```
Downcasting to Son
    10    20
Downcasting to Daughter
    10    30
```

Program 3

```
Package comP;
class vehicle
{
    String brand = "Audi"
}
class car Extend vehicle
{
    String colour = "Black"
}
class bike Extend vehicle
{
int cost = 50000;
}
```

class Solution

{

// Generalization

// vehicle obj = new Bike (or) vehicle obj = new car

Static void display (vehicle obj)

{

if (obj instanceof car)

{

car c = (car) obj;

S.O.P ("c.brand" " + c.colour)

}

else if (obj instance of bike)

{

bike b = (bike) obj;

S.O.P (b.brand + " " + b.colt)

}

}

public static void main (String[] args)

{

display (new car());

display (new bike());

}

}

O/P

Audi Black

Colt 50000;

Method

"A set of instruction perform some task is known method"

NOTE :- when method returning some value either store it (or) Print it"

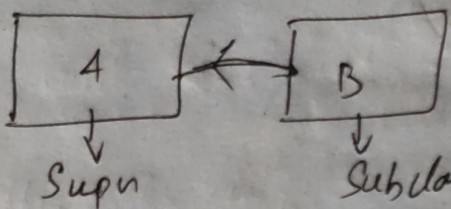
Ex:

```
class Demo
{
  static void m1()
  {
    S.O.P("Hai");
  }
  static int m2()
  {
    return 10
  }
}
class Solution
{ Public static void main(String[] args)
  {
      m1();

    int x = m2();
    S.O.P(x); //10
      (or)
    S.O.P(m2()); //10
```

Inheritance

-> "A class acquires the properties of another class is known Inheritance"



-> Super class is shares the properties

-> Sub class is acquires the properties

-> For achive Inheritance we use Extends keyword

-> Inheritance is also reffred as IS-A Relationship

-> Variable & Method are inherited whereas Constructor & Blocks are not inherited.

V & M ✓

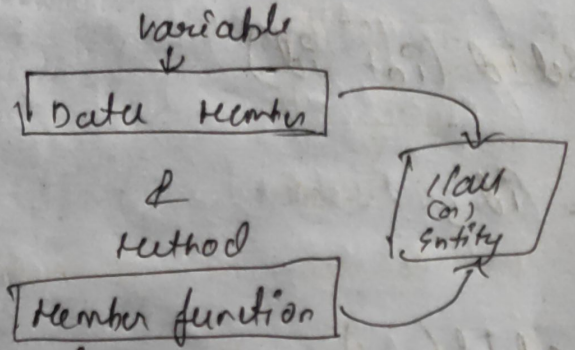
B & C ✗

Types of Inheritance

1) Single ✓
2) Multi value ✓
3) Hirarchi ✓
4) Multiple ✗
5) Hybrid ✓

Encapsulation

"The process of Binding/Grouping/Wrapping the Data member and member function in a Single Entity/class"



Eg:- Java bean class.

Class :- Blueprint of an object is known class

(or)

class is Defination block used to declare (or) define state & behavior of an object.

Object :- object is Realword Entity.

Anything (or) which is present in the realword, and Physical Existan.

Rules :-

1:- For Java bean class

1) ~~Public~~ class Name should be Public

2) Declare private Data Member

3) Public Setter method & public Getter method.

Ex:- For java bean class

```
Public class Emp
{
  Private int id;

Public void setId (int id)
  {
     this.id = id;
  }
Public int getId( )
    {
      return id
    }
}
class Solution {
 Public static void main (String[] arg)
  {
 Emp e = new Emp( );
    e.setId (100);
   int id = e.getId( );
   S.o.p (e.getId( ))
  }
}
```

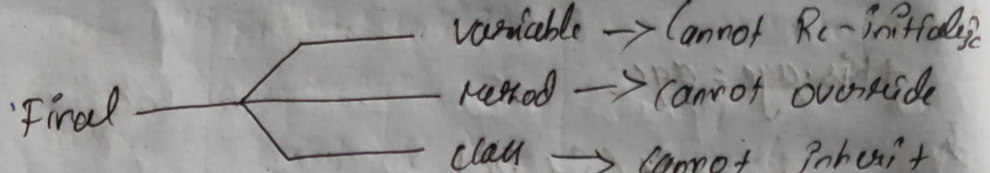
O/P

100

```
Public class Student{
    Private String name;
    Private int age;
    Private int id;
}
Public void setName(String name)
{
    this.name = name;
}
Public void setAge(int age)
{
    this.age = age;
}
Public void setId(int id)
{
    this.id = id;
}
Public String getName(){
    return name;
}
Public int getAge()
{
    return age;
}
Public int getId()
{
    return id;
}
class Solution
Public static void main(String[] args)
{
```

```
Student s = new Student();
    s.setName("Mahesh");
    s.setAge(24);
    s.setId(101);
s.o.p(s.getName());
s.o.p(s.getAge());
s.o.p(s.getId());
}
```

Final



Final : Final keyword which is used with a variable, method, class.

1. Final variables cannot Re-initialize (It acts as constant)
2. Final method can be inherited but cannot be overridden
3. Final class cannot be inherited at all

***

Coding Standards (or) Naming Conventions : (Oracle)

1) class → Car, AudiCar, SortEmpByAge
2) variable → age, studentAge, noofStudent
3) method → display(), showDetails(), setAge(), getAge()
4) package → lowercase → com, org, ...
5) Constant → final variable → Uppercase → PI

Method overloading :-

In a class contain multiple method & method name should be same but differ in arguments (or) parameters.

* Rules of overloading in arguments
  1) change in No of Args
  2) change in data type
  3) change in sequence
* Method overloading can done :
  1) Static ✓
  2) main method ✓
  3) Non Static ✓

Concrete class

```
class Car
{
    // Concrete method
}
// no abstract method.
```

→ Object creation ✓

Abstract class

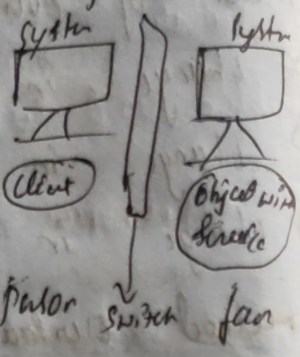
```
abstract class Pen
{
    // Concrete method & Abstract method
}
```

Object creation ✗

Interface

```
Interface Bike
{
}
```

Object creation ✗



Motor Switch fan

Abstract method

```
void test()
{
}
```

Both method declaration & method implementation

Abstract method

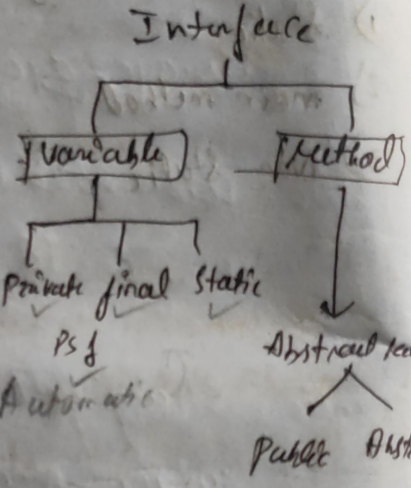
```
Abstract void test()
```

Abstract method cannot be

Private ✗ final ✗ static ✗

PSf ✗

JVC cannot override.



NOTE: Goal of Abstract method is **Override**.

Concrete class: A class which is not declared using 'abstract keyword' is called Concrete class

→ Concrete class can allow only **concrete method**

```
class A
{
    // Concrete methods
}
```

Abstract class:

→ A class which is declared using abstract keyword is called as "Abstract class"

→ Abstract class can allow both Abstract method & concrete methods.

Ex: abstract class car
{
// Both abstract & concrete method.
}

Concrete method:-

→ A method which has both declaration & implementation is called as concrete method

Ex: void test() → method declaration (head)
{ - - - - → method implementation (body)
}

Abstract method

→ Abstract method has to be delared using abstract keyword

→ A method which has only declaration & no implementation has to be suffixed as Abstract method

Syntax:- Access specifier abstract .Returntype method Name( );
↓
Arguments

Ex: 1) Public abstract void display( );

2) _____ abstract void test( );
default

Abstract method cannot be override.

private final static

so we cannot override.

Contract of abstract (Rules):

When a class Extends an abstract class we have to either

i) Override the inherited the abstract method
ii) Make the class has abstract

Ex:-

```
Package org;

abstract class person {
    abstract void eat();
}
class Tom Extends person {
    @override // annotation override.
    void eat()
    {
        System.out.println("Eating");
    }
}
Public Static void main (String[] args)
{
    Tom t = new Tom();
    t.eat();
}
```

O/P
Eating

Note:
"While overriding the method the access modifier should be same (or) higher visibility"

Is-A	Class	Interface
Class	Extends	Implements
Interface	X	Extends

Ex:

```
Package org;
Interface Amzon
  {
    int id=101; // Public static final int id=101;
    void purchase(); // Public abstract void purchase();
  }

Class Paytm implements Amazon{
  @overriding
  Public void purchase()
    {
      S.o.P("Purchashing Mobile");
    }

Public static void main(String[] args)
    {
      S.o.P("Id:"+Amazon.id);
      Paytm p=new Paytm();
      P.Purchase();
    }
```

O/P
Id : 101;
Purchasing Mobile

NOTE :

-> In Java we can create an object of only Concrete class but we can't create a object of Abstract and Interface.

-> In Java abstract class & Concrete class both can have constructor, But interface can't have constructor.

-> Abstract class can have constructors & those constructor, are invoked either implicitly (or) explicitly using super calling statement

Interface

* Interface is a Java type definition which has to be declared using Interface keyword.

(or)

* Interface can also be defined as "a medium b/w 2 system where in 1 system will behave as client & another system will behave as system with resource."

Syntax :- Interface InterfaceName
{

}

* It is possible to have a variable in an interface & those variable are automatically Public, Static, Final

→ Interface can allow only abstract method & those methods are automatically public & abstract

Ex:

```
Interface Test
{
Public static final int x = 50;
Public abstract void display();
}
```

Public, static, final → Default
Public, abstract → Default.

→ Interface doesn't contain any constructor therefore we cannot create an object of interface.

→ A class can achieve Is-a-relationship (Inheritance) with an other Interface using implements keyword

NOTE:

→ When a class implements an Interface it is mandatory to override abstract method

→ The Access specifier should be same (or) of Higher visibility.

→ A class can implement Any number of interfaces.

→ A class can extend another class & implements any number of Interfaces.

Ex: public class Demo extends Demo2 implements A, B

Interface.

	Object creation (or) Instantiation	variable	Constructor	Abstract method	Concrete method
Concrete class (or) Non abstract class	✓	✓	✓	✗	✓
Abstract class	✗	✓	✓ (Super)	✓	✓
Interface	✗	✓ Public, static, final (PSF)	✗	✓ PSA	✗ NO, from JDK Static & Default

Abstract class

Syntax

abstract class Tut
{
define static variable ✓
define non-static variable ✓
define constructor ✓
define static method ✓
define non-static method ✓
}

Interface

Syntax

interface Tut
{
Only static variable are allowed ✓
No constructor ✗
Only Abstract method ✓
No Concrete method ✗
Only Public Access is allowed ✗
}

Method overriding

→ "Method overriding is a process of inheriting the method & changing the implementation of the inherited method. (or) that

To achieve method overriding

1) Method Name should be same
2) Argument should be same
3) Return type should be same.

→ Access Specifier should be same (or) higher visibility.

→ In order to achieve method overriding, inheritance is necessary.

The following method cannot be override

1) Private method :- Because the private doesn't have acces in subclass

2) final method :- The final keyword doesn't allow to override method in a subclass

3) Static method :- Because the static method are not inherited to subclass

→ Only the non-static method can be overridden

→ method overriding is used to achieve runtime polymorphism

Method Binding

Method Binding is process of associating mapping the method call to its method implementation

Polymorphism

→ An object showing different behaviour at different stage of its life cycle is known as polymorphism.

→ Polymorphism means many form

→ In Java "the ability of a method to behave differently, when different object are acting upon it", is called polymorphism

(or)

→ "The smartness of method to exhibit different form when different object acting upon it," is called polymorphism

Polymorphism are 2 types:

1) Complie time polymorphism
2) Run time polymorphism.

1) Compile time polymorphism:

→ Compile time ploymorphism is achieved with help of method overloading.

→ Compile is also called as static binding (or) Early binding

→ In Compile time Polymorphism method binding happen during Compile time and Compiler decide

① Create a class called as Amazon under that achieve method overloading and method name should be purchase and create another class under main achieve Runtime time polymorphism.

F1:

```
Package ctp;
class Amazon
{
  void purchase (int cost)
  {
    S.O.P ("cost:" + cost);
  }
  void purchase (String brand)
  {
    S.O.P ("Brand:" + brand);
  }
  void purchase (int cost, String brand)
  {
    System.out.println ("cost:" + cost + "Brand:" + brand);
  }
  void purchase (String brand, int cost)
  {
    S.O.P ("Brand:" + brand + "cost:" + cost)
  }
}
Package ctp;
class Solution
  Public static void main (String[] args)
  {
    Amazon obj = new Amazon();
    Obj. purchase ("Apple");
    Obj. purchase (100, "Samsung");
    Obj. Purchase ("oneplus", 200);
    Obj. Purchase (500);
  }
}
```

O/P

Brand: Apple
cost: 100 Brand: Samsung
Brand: oneplus
cost: 200
cost: 500

2) Runtime Polymorphism :-

→ Runtime polymorphism with is achieved with the help of

Rule :- i) is-a Relationship
ii) method overriding
iii) upcasting

→ Runtime polymorphism also suffered as Dynamic Binding & late Binding

→ In Runtime polymorphism the method binding happen's during Runtime and J.V.M decides.

NOTE :- If a method is overriden always, the overriden meth implementation gets Executed, Even if it is upcasted (or) downcasted reference.

```
Package SP;
class Employee {
void Work() {
S.O.P ("Working");
}
}
class Developer Extends Employee { // Rule-1
void Work() // Rule-2
```

```
    {
      S.O.P("Developing App");
    }
}
class Tutor Extends Employee
  {
    @Override
   void Work()
    {
      S.O.P ("Tutting App");
    }
Public class Solution {

  public static void main (String[] args)
    {
        Employee e1 = new Developer(); // Rule-3
        e1. work();
        Employee e2 = new Tutor(); // Rule-3
        e2. work();
    }
}
```

(or)

```
Public class Solution
  {
    Public static void main (String[] args)
    {
      Static void display (Employee obj)
        { obj.work();
        }
    PVSM
      display (new Developer());
    } display (new Tutor());
```

Abstraction

→ The process of hiding the implementation and showing functionalities to the user is called Abstraction.

Rules for achieving Abstraction:-

1) Abstract class / interface with Abstract methods.
2) Is-A Realtionship (Inheritance)
3) Method overriding
4) Upcasting

Ex:-

```
Pakage abstraction;
interface Calculator { // Rule-1
    void add (int a, int b);
}
```

```
pakage abstraction;
class CalculatorImp implements Calculator { // Rule-2
    @Override
    Public void add (int a, int b) { // Rule-3
        S.O.P ("Sum of "+a+" & "+b+" is "+(a+b));
    }
}
```

```
Package abstraction;
class Solution
{
  public static void main (String[] args)
  {
    Calculator calc = new CalculatorImp(); // Rule 4 upcasting.
    calc.add(10, 40);
  }
}
```

O/P = Sum of 10 & 40 is 50

9/07/2

Java libraries

→ Java libraries is a collection of predefined

→ Each package is the collection of class and interface

→ Each class (or) interface is the collection of variable and method

→ All the predefined pakage are present in zip file called as src.zip (Source) or jarfile called as Rt.jar (runtime.jar)

A few predefined packages are as follows

java.lang
java.util
java.io
} → Scanner

java.sql
java.net
java.math
java.awt

NOTE:

java.lang → object, Thread, System, String...
java.util → Scanner, ArrayList, HashSet...
java.io → File, FileReader, FileWriter...

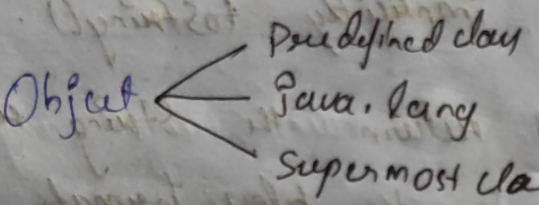
Java.lang Package

Java.lang package is automatically imported in all classes and interfaces

Object

->Object is a predefined class present in java.lang package.

->Object is the supermost class in entire java because every class in java will implicitly inherite Object class.




```
class Pen (extends Object) -> implicity inherite
{

    Object obj = new Pen(); //upcasting

}
```

Important method's present in method class :-

1. public String toString()
2. Public int hashCode()
3. public boolean equals(Object obj)
4. public void notify()
5. public void notifyAll()
6. public void wait()
7. public void wait(long a)
8. Public void wait(long a, int b)
9. Public class getClass()

10. Protected Object clone()
11. protected void finalize()

1) toString() : (toString method)

① → toString() will return the String representation of an object

② Syntax: public String toString(){

}

③ → when we print the reference variable (or) object reference implicitly toString() get's called

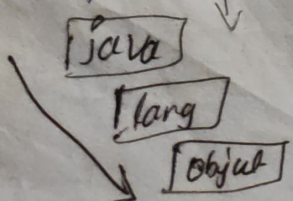
→ toString() will return the String representation of an object in the below format

④ ~~FullyQualifiedClassName~~ FullyQualifiedClassName@HexaDecimalvalue of HashCode

NOTE: FullyQualifiedClassName means "consider the class name from the package level"

→ in order to provide customize implementation we need to override toString();

Fully Qualified Classname for object class



java.lang.Object

Program without Overriding toString method.

```
class Car {
    PVS M (String [] args) {
        Car C = new Car();
        S.O.P (C); // Implicitly calling toString();
        S.O.P (C.toString()); // Explicitly calling toString();
    }
```

O/P

```
Com.Car@5c2c5ba
Com.Car@5c2c5ba
```

Program after overriding toString() @ method

```
class Car {
    @Override
    Public String toString() {
        return "Car@123";
    }
    Public Static void main (String [] args) {
        Car C = new Car(); Car C = new Car();
        S.O.P (C); // implicitly calling toString
        S.O.P (C.toString()); // Explicitly
```

O/P

```
Car@123
Car@123
```

2) hashCode() @ method

-> hashCode() method returns a unique id @ no for an object

-> hashCode() @ method is used to identify an object uniquely

Syntax:

```
Public int hashCode() {
}
```

Program without overloading hashCode();

```
class Employee {
    Public static void main(String[] args) {
        @Override
        Employee emp = new Employee();
        S.O.P (emp.hashCode());
    }
}
```

O/P

1579572170

Program after overriding hashCode();

```
class Employee {
x (Public static void main(String[] args) {)x.
    @Override
    Public int hashCode() {
        return 12345;
    Public static void main(String[] args) {
```

```
Employee emp = new Employee();
S.O.P (emp.hashCode());
}
}
```

O/P 12345;

Ex:- 1 20/9/22

Overriding toString method to return the attributes.

```
Package org;
class Student {
    int age;
    String name;
Student (int age, String name) {
    this.age = age;
    this.name = name;
}
@Override
public String toString() {
    return "Age of "+name+" is "+age;
}
}
class Test {
    Public static void main(String[] args) {
        Student s=new Student (22, "Age");
        S.O.P(s);
    }
```

O/P Age of Alex is 22

Ex: 2

Write a java program the below scenarios

i) Create a class called as Employee and declare two attributes called as name & Salary initilize the attributes with help of a constructor. Override ~~to~~ toString method to return the name and salary in the below format.

Salary of Tom is 15000.45. and create another class Solution under main method create 2 instances of Employee and invoke toString simplicity.

```
Package com;
class Employee {
    String name;
    int salary;

Employee (String name, int salary)
{
    this.name = name;
    this.salary = salary;
}

Public String toString ( ) {
    return "Salary of" + name + "is" + salary;
}
}

class Solution {
    Public static void main (String[] args) {

        Employee Emp1 = new Employee ("Tom", 15000.45)
        Employee Emp2 = new Employee ("Jerry", 17000.12)
```

```
S.O.P (emp1);
S.O.P (emp2);
}
}
```

O/P

```
salary of Tom is 20000.00
salary of Jerry is 30000.00
```

Array:-

→ "Array is a container in order to store a group of element (or) data."

→ In otherwords "Array is collection of object"

→ Array is homogeneous in nature (same type)

→ Array is fixed size

→ Array is indexed based & index position start from 0

① Array Declaration

Syntax:- datatype [] arrayName;

→ int[] a;

→ double[] x;

② Array Creation

Syntax:- arrayName = new datatype [size];

→ a = new int[3]; a [0|0|0] (0 1 2)

→ x = new double[4]

x [0.0|0.0|0.0|0.0]

③ Array Declaration & Creation

Syntax:- datatype [] arrayName = new datatype [size];

→ int [] a = new int [3];

a [0|0|0] (0 1 2)

→ String [] s = new String [2]; s [null|null] (0 1) (String)

4) Array Initialization.

Syntax: arrayName [index] = value;

```
a [i] = 10;
s [0] = "Java";
```

```
Package ary;
class Demo {
public static void main (String[] args ){
// Array declaration
int[] a;
// Array location
a = new int[2];
// printing Array Element
S.O.P(a[0]); // 0
S.O.P(a[1]); // 0
S.O.P ("------");
// Array Initialization
a[0] = 200;
a[1] = 100;
S.O.P (a[0]); // 200
S.O.P (a[1]); // 100
S.O.P ("------");
```

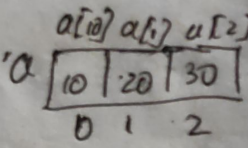
// Array Declaration and creation:

```
double [] x = new double [3];
S.O.P (x[0]); //0.0;
S.O.P (x[1]); //0.0
S.O.P (x[2]); //0.0
}
// Array intilization
S.O.P ("------");
x[0] = 1.1;
x[2] = 2.2;
S.O.P (x[0]); //1.1;
S.O.P (x[1]); //0.0
S.O.P (x[2]); //2.2
```

5) Array Declaration and Initalization:

Syntax:- datatype [] arrayName = { V_1, V_2 ... }

```
int[] a = {10,20,30}
```



array Name length

```
S.O.P(a.length)
O/P = 3
```

valid Array Declaration

1) int []a; ✓
2) int a[]; ✓
3) int []a; ✓

Program:- Basic Array

```
Package org;
class ArrayDemo {
public static void main(String[] args) {
    S.O.P(a[i]);
    int [] a = {10, 20, 30}; (or) int a[] = {10, 20, 30};
    for (int i=0; i<a.length; i++){
        S.O.P(a[i]);
    }
    S.O.P("-------");
//reverse
    for (int i=a.length-1; i>=0; i--){
        S.O.P(a[i]);
    }
    S.O.P("-------");
    String[] subjects = {"java", "sql", "apti", "css"};
    for (int i=0; i<subjects.length; i++){
        S.O.P(subjects[i]);
    }
    S.O.P("-------");
//reverse
    for (int i=subjects.length-1; i>=0; i--){
        S.O.P(subjects[i]);
    }
    S.O.P("-------");
```

3) Equals() method:

→ Equals method is used for content comparison of an object.

→ Syntax :-

Public boolean Equals (Object obj)

→ By default equals method will compare the Address (or) reference of 2 objects, therefore in order to compare contents of object, we have to Override the Equals() methods.

1) Program: without overriding Equals():

```
Packag ful;
Class Student {
    int age;
    Student (int age){
        this.age = age;
    }
Public static void main (String [] args) {
    Student s1 = new students (20);
    Student s2 = new students (20);

    S.O.P (s1 == s2);
    S.O.P (s1.equals (s2));
```

O/P

False
False

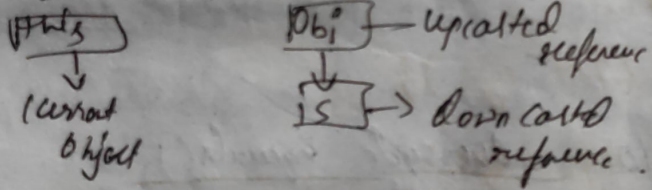
Rules for Overriding equals():

1) Upcasting
2) Downcasting
3) Comparision logic

```
@override
Public boolean equals (object obj)
{
    Student s = (Student) obj;   ② Downcasting
    return this.age == s.age;    ③ Comparision logic
    // s1.age == s2.age
    20 == 20
}
```

```
P V S M
{
S.O.P(s1.equal(s2))
```

new Student(20);



Program for override equals():

```
Package ti;
class Student {
    int age;
    Student (int age)
    {
        this.age = age;
    }
```

```
@Override
Public boolean Equals(Object obj)   // Rule 1
  {                                 Object obj = new Student(20);
                                    means upcasting
  Student s = (Student)obj;  // Rule 2 :- downcasting

  return this.age == s.age  // Rule 3 :- comparision logic
  }                                 s1.age == s2.age
Public static Void main(String[] args){
  Student s1 = new Student(100);
  Student s2 = new Student(100);
    S.O.P (S1.Equals(S2));
  }
}
```

O/P :
True

Program 2 :- for override Equals()

```
Class Employee {
  int id;
Employee (int id){
  this.id = id;
}
@Overide
Public boolean Equals(Object obj)   Rule 1.
  {                                 upcasting
                                    Object obj = new Employee();
  Employee e = (Employee) obj;  Rule 2 downcasting
```

return this.id == e.id ; // Rule 3 Comparison
e1.id == e2.id;

}

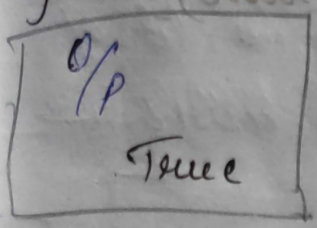
Public static void main (String[] args) {

Employee e1 = new Employee(20);
Employee e2 = new Employee(20);

S.O.P (e1.equals(e2));

}

}



Write a Java program following below Scenario
Create a class called as Car, and declare
2 attribute called as brand & cost, and initialize
the data member using Constructor, Overrid equal()
to Compare brand & cost of 2 car Object.

=> class Car {

String brand;
int cost;

Car (String brand, int cost)
{
this.brand = brand;
this.cost = cost;
}

@Override

```
Public boolean Equals (Object obj) {
    Car c = (Car) obj;
    return this.brand == c.brand && this.cost
        == c.cost ;
} // this.brand
Public static void main (String [] arg).
{
    Car c1 = new Car ("BMW", 2500000);
    Car c2 = new Car ("Audi" 2500000);
    S.O.P (c1.Equals (c2));
}
}
```

this.brand.Equals(c.brand)

O/P

False

or

```
if (c1.equals (c2)) {
    S.O.P ("Brand & cost are same");
}
else {
    S.O.P ("Brand & cost are different");
}
```

NOTE:-

// for string comparision follow below steps:

// kindly use --> this.brand.Equals (c.brand)

// Do not use --> this.brand == c.brand ---> not a good practise.

Method Name	Access modifier	Return type	Arguments
toString()	Public	String	X
hashCode()	Public	int	X
Equals()	Public	boolean	(object)

toString() → String representation of an object

hashCode() → unique id / no. for an object

equals() → content comparison.

String

→ String is a pre-defined final class present in java.lan package

→ String object are immutable in nature (can't change (or) modify)

→ String is class as well as a data type Therfore we refer String as non-primitive data type

→ The default value for String any class type is Null

→ String can also be reffred a set of character's which should be Enclosed with in double code's. Ex: " "

→ string object can be created in two ways.

(1) without new operator (literal objects):

Ex: String s = "java";

(2) with new operator:

ex: String s = new String ("java");

→ String object will get stored inside a memory location String pool

String pool further divided into

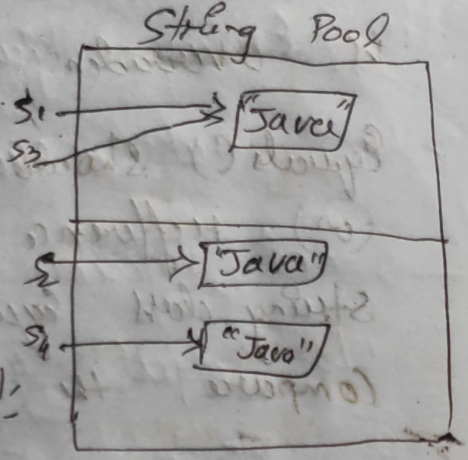
i) constant pool

ii) Non-constant pool

→ String literal objects will get stored inside constant pool and constant pool does not allow duplicates.

→ String object create using new operator will get stored inside non-constant pool and non constant pool allows duplicate.

① String s1 = "java";
② String s2 = new String("Java");
③ String s3 = "Java";
④ String s4 = new String("java");



① Constant pool
→ Literal object
→ NO Duplicates

② Non-Constant pool
→ new operator
→ Duplicate Allowed

NOTE:
→ String class implements the following interfaces:
1. Serializable
2. Comparable
3. CharSequence

→ String also implicitly inherits the Object class and has overridden three method
1) toString() 2) hashCode() 3) Equals()

→ toString() should have returned the String representation of an object but, In String class toString() is Overriden to return the Set of character

→ hashCode() should have returned a unique random number but, In String class hashcode() is overriden to return the ASCII Value

→ Equals() should have compare the address (or) reffernce of two objects but in string class Equals() is overriden to compare the content of two String object.

Ex:- (1)

```
Package com;

Class Student {
    public static void main (String[] args){
        Student s = new Student();

        S.O.P (s); //example.Student@5c268a4
        S.O.P (s.toString); //example.Student@5c268a4

        S.O.P (s.hashcode()); // 1579572132

        Student s1 = new student();
        Student s2 = new student();
        S.O.P (s1.Equals(s2)); // false
    }
}
```

Ex (2)

```
package com;

class Demo {
    Public Static void main (String[] args) {
        String s = new string ("a");
        S.O.P (s);
        S.O.P ( s.to string());
        S.O.P ("-----");
        S.O.P (s.hashCode());
        S.O.P ("-----");
        String s1 = new string ("java");
        String s2 = new string ("java");
        S.O.P (s1 == s2);
        S.O.P (s1.Equals(s2));
    }
}
```

O/P

a
a

97
---
false
true

NOTE: '==' -> will compare the address (or) Reference

Ex: s1 == s2

'Equals()' -> will compare the data (or) Contents

Ex: (s1.Equals(s2));

How String objects are Immutable in nature

(or)

Explain String immutability concept?

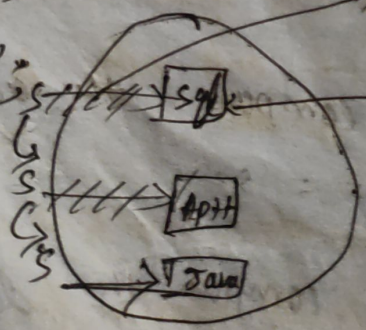
→ when we re-initialize a string object, the existing object is not modified, rather than that a new object gets created.

→ And reference variable pointing to the old object gets dereferenced & starts pointing to newly created object, therefore String object are immutable in nature.

The mutable version of String are:

1. String Buffer.
2. String Builder

String s = "sql";
String s = "Apti";
s = "Java";



① String Buffer:-

StringBuffer is a predefined final class present in java.lang package. StringBuffer was introduced from jdk 1.0, StringBuffer objects are mutable in nature.

② String Builder:-

StringBuilder is a predefined final class present in java.lang package. StringBuilder was introduced from jdk 1.5, StringBuilder is mutable in nature.

String	StringBuffer	StringBuilder
1) Predefined final class in java.lang package	→ Predefined final class in java.lang package	→ Predefined final class in java.lang package
2) Introduced from JDK 1.0.	→ Introduced from JDK 1.0.	→ Introduced from JDK 1.5
3) Immutable in nature	→ Mutable in nature	→ Mutable in nature.
4) Thread safe	→ Thread safe	→ Not thread safe.
5) Overriden toString() hashCode() & Equals()	→ only toString() overriden.	→ only toString() Overriden
6) String objects can be created with (or) without new operator.	→ StringBuffer objects can be created only using new operator	→ StringBuilder objects can be created only using new operator.
7) '+' operator can be used for concatination	→ '+' operator cannot be used for concatination	→ '+' cannot be used for concatination.

NOTE:

Both StringBuffer and StringBuilder implements the following interfaces

1) Serializable
2) Comparable
3) CharSequence

```
Package com
Class Solution
{
  Public static void main (String[] args)
  {
      String s1 = new String ("Java");
       S.O.P (s1); // Java
      s1. concat ("program");
       S.O.P(s1); // Java
S.O.P("-----------");
   String s2 = new String ("Java");
      S.O.P (s2); // Java
       s2 = s2. concat ("program");
      S.O.P (s2); // Java program
S.O.P("-----------");
   StringBuffer s3 = new StringBuffer ("Good");
       S.O.P (s3); // Good
```

```
S3.append("Afternoon");
    S.O.P(S3); //Good Afternoon
S.O.P("-------------------");
String Builder S4 = new StringBuilder("Dabba");
    S.O.P(S4); //Dabba
S4.append("Fallows");
    S.O.P(S4); //Dabba Follows
S.O.P("---------------------");
String S5 = new String("Wake")
    S.O.P(S5); //Wake
  S5 = S5.concat("up");
    S.O.P(S5); //Wake up
}
}
```

Program 2

```
class Runner
{ PVSM()
{
String S= "Software Developer";
S.O.P(S.length()); //18
S.O.P(S.startswith("Soft")); //false
S.O.P(S.Endswith("er")); //true
S.O.P(S.toUppercase()); //SOFTWARE DEVELOPER
S.O.P(S.to lowercase()); //software developer
S.O.P(S.charAt(1)); //O
S.O.P(S.indexof('w')); //4
S.O.P(S.Contains("Learn")); //True
```

```
String a= "Java";
String b= "Java";
String c = "Java";
S.O.P(a.Equals(b)); //True
S.O.P(a.Equals(c)); //false
S.O.P(a.EqualsIgnoreCase(c)); //True
```

Storing Object inside an array

(1) Write a Java program by following below Scenario

i) Create a class called as Employee

ii) Declare two attribute called as id and name

iii) Initialize the attribute with help of a Constructor

iv) Override toString method to return the id and name in the below format

o/p Employee id of Tom is 100

(v) Create another class called as Solution and in main method create three instance of Employee store it inside an array and traversed it using for loop.

```
Package StoringObjects;
Class Employee
{
    int id;
    String name;
Employee (int id, String name)
  {
  this.id = id;
  this.String name = name;
  }
@ Override
Public String toString()
  {
    return "Employee id of" + name + "is" + id;
  }
}
```

```
class Solution
{
    public static void main(String[] args)
    {
        Employee emp1 = new Employee(100, "Tom");
        Employee emp2 = new Employee(200, "Jerry");
        Employee emp3 = new Employee(300, "Dog");
```

② Create a

```
        Employee [] emp = {e1, e2, e3};
        for(int i=0; i<emp.length; i++)
        {
            S.O.P (emp[i]); //printing e1, e2, e3 (reference variable)
        }
    // if toString() is not Overridden use the below line
    // S.O.P (emp[i].id + " " + emp[i].name);
    }
}
```

② Write a java program to store car object inside an array and traverse it using for loop. Every car will have attribute (brand, cost):
Package storing objects

```
class car
{
    String brand;
    int cost;

car (String brand, int cost)
{
    this.brand = brand;
    this.cost = cost;
```

```
Public String toString()
{
    return "car brand is" + brand + "and cost is" +
                                    cost;
}
}
Class Solution
{
    Public static void main (String [] args)
    {
        Car C1 = new Car ("Audi", 100);
        Car C2 = new Car ("BENZ", 200);
        Car C3 = new Car ("BMW", 300);

        Car [] C = {C1, C2, C3};
// Traversing car objects when toString() is
                                    overrided
        for(int i=0; i<C.length; i++){
            S.O.P("Car brand is" + C[i].brand + "And
        S.O.P("-----------");  C[

// Array of type Car created of size 3
        Car [] car = new Car[3];

// Storing car objects directly in the array
        car[0] = new Car("Audi", 100);
        car[1] = new Car("BENZ", 300);
        car[2] = new Car("BMW", 300);
```

// Traversing Car objects where toString() is overridden.

```
for (int i=0; i<car.length; i++)
  {
    S.O.P (car[i]);
  }
}
```

O/P

Car brand is Audi and cost is 100
Car brand is Benz and cost is 200
Car brand is BMW and cost is 300

1) Write a Java Program to reverse a String
2) Write a Java program to check if string is Palindrome (or) not.

(1)

```
Package com;
Public class ReverseString {
  Public static void main (String[] args) {
    String s = "JAVA";
    char[] ch = s.toCharArray();
    for(int i=ch.length-1; i>=0; i--)
    {
      S.O.P (ch[i]);
    }
  }
```

O/P AVAJ

```
class SumOfArray {
    P V S M {
    int[] a = {10, 20, 30, 40, 50};
    Sum = 0; // sum // a local variable as
                    local variable are not
                    have default
                    value
    for (i=0; i<a.length; i++)
    {                          // And we have to
        S.O.P {                   initialize sum = 0;
        sum = sum + a[i];
    }
     S.O.P ("SUM:" + sum);
     S.O.P ("AVG:" + (sum/a.length));
    }
}
```

O/P

SUM: 150
AVG: 30

Exception handling

Error :- "Error is a mistake (or) a problem which generally occur during Execution of a Program"

→ Error can occur during ~~first~~ compile time ~~2nd Run time~~

1) Compile time
2) Run time

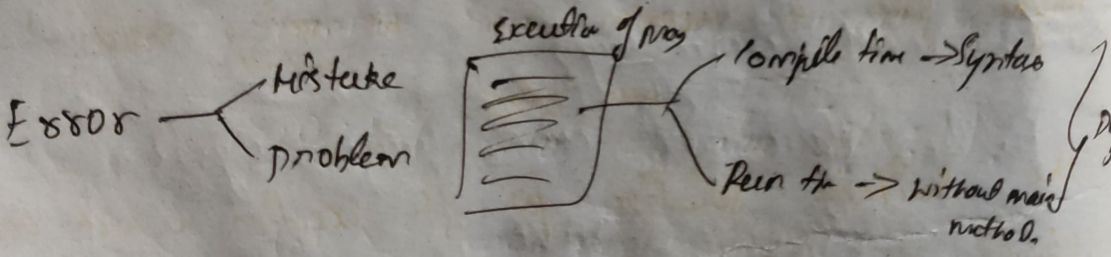
→) In Java we get compile time Error due to Syntax mistake

→ 2) We might also get Run time Error, when we execute a class without main method and also when stack is full

↓

(StackOverflowError)

→ Error should always be debugg



Exception:-

→ Exception is a event (or) interruption which stops the execution of program

(or)

In otherwords Exception is an Runtime interruption which terminate (or) stop the execution of Java program

→ Exception will always occur during runtime

→ Exception should always be handle and the handling an Exception is called has exception handling

→ In Java we generally handle an Exception try block and catch block.

Try block and catch block:-

→ In Java the critical line of code which might give a Exception should be written inside try block.

→ The suitable solution for the exception which occur should be written inside catch block

→ It is mandatory to have try and catch block together

→ The catch block we get executed only when Exception occur.

Syntax

try block

```
try {
}
```

catch block

```
Catch ( ExceptionName referenceVariable )
{
}
```

Ex 1 :- ArithmaticException

```
Package day1;

import java.util.Scanner;

class Solution {

  public static void main (String [] args) {

      S.O.P("start");

      Scanner Scan = new Scanner(System.in

      S.O.P ("Enter the value of x:");

      int x = Scan.nextInt(); // int x=0;

      S.O.P("Enter the value of y:");

      int y = Scan.nextInt(); int y=0;

      Scan.close();
```

```
try {
    S.O.P (x/y); // 10/0 -> Arithmetic Exception
    }
catch (ArithmeticException obj)
    {
      S.O.P ("kindly do not divide by 0");
    }
    S.O.P ("End");
  }
}
```

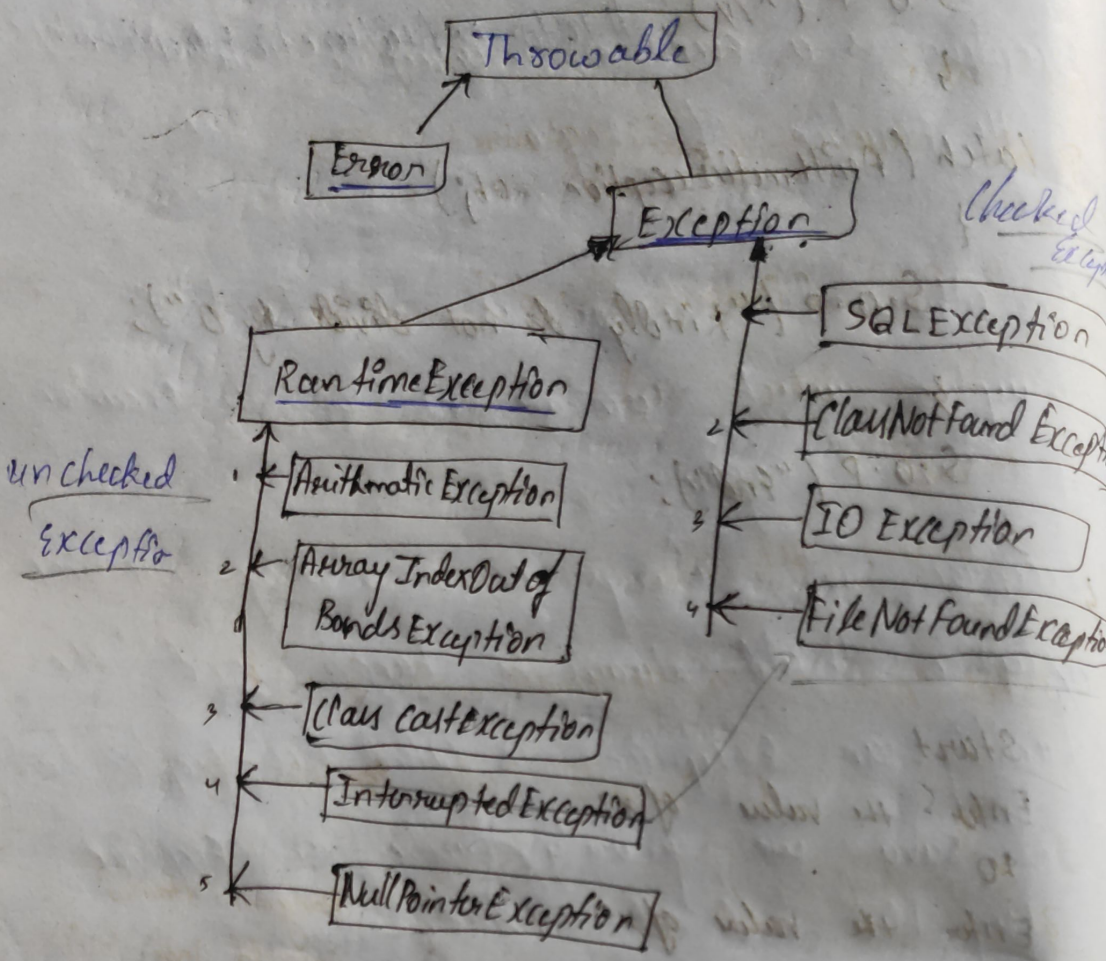
O/P

```
Start
Enter the value of x:
10
Enter the value of x:
0
kindly do not divide by 0
End
```

NOTE

In Java all the Exception are either pre-defined (or) user defined classes

Exception Hierarchy



NOTE

1) It is always important to handle the Exception using suitable try and catch block

2) We can't have any Executable line of the code b/w try block and catch block

3) Comments are allowed b/w try and catch block because comment is non executable code.

Ex:-2 ArrayIndexOutOfBoundException

```
Package dars1;
class Test {
  Public static void main (String [] args)
  {
    S.O.P ("Program started");
  int[] a = {20, 24, 30};
    try {
      S.O.P (a[100]);
    }
  catch (ArrayIndexOutOfBoundsException e) {
        S.O.P ("Invalid Index Position");
    }
    S.O.P ("End");
  }
}
```

o/p
Program started
Invaild Index position
End

NOTE

-> when an Exception occur we can handle it using same type (or) superclass type.

-> when we handle it using same type, we refer it has Specialization

-> When we handle it using ~~Exception~~ superclass type, we refer it has Generalization

Ex:- 3 Why we are using reference variable for catch

```
Package day3;

class Demo{
  Public static void main(String[]args)
  {
    //Specialization --> specific Exception Handler
    try{
      S.O.P (10/0); // throw new ArithmeticException()
    }
    catch (ArithmeticException e){ // ArithmeticException e=
                                   new ArithmeticException();
      S.O.P ("Invalid Denominator");
    }

    // Generalization -> Superclass Exception Handler
    try{
      S.O.P (10/0); // throw new ArithmeticException(
    }
    catch (Exception e) // Exception e = new Arithmetic
    {                                   Exception();
      S.O.P ("Invalid Denominator");
    }
  }
}
```

Internal working when we access an invalid Index Position

→ In Java, when we try to access an index position which is not available, internally, an object of Array IndexOutOfBoundsException(); is throw.

→ The object throw can be handled (or) caught using the same type [ArrayIndexOutOfBoundsException()]

Specialization

(or)

using super class type [Exception]

Generalization

→ It always a good practices to handle an Exception with suitable try and catch block.

→ One try block can have any number of catch blocks but suitable catch block will get Executed

→ When their is subclass catch blocks and superclass catch blocks, we should always have superclass catch block at the last.

→ It is always a good practice to handle the superclass catch blocks has the last catch blocks

Ex6

```
class Monday{
  Public static void main (String[] args){
    try{
      S.o.p (10/0); // throws new ArithmeticExcep
    }
  Catch (NullpointerException e){
    S.o.p ("Nullpointer Exception handled");
  }
  Catch (ArithmeticException e){
    S.o.p ("Arithmetic Exception handled");
    }
S.o.p ("-----");
  try {
      S.o.p (10/0);
    }
  Catch (Exception e) { // Exception e = new ArithmeticException()
  Generalization
    S.o.p (e.getMessage());
    String msg = e.getMessage();
    S.o.p (msg);
}}
    S.o.p ("All the Exceptions
    are Handled");
}}
```

O/P

```
Index 100 out of bounds for length 3
Index 100 out of bounds for length 3
---
by zero
by zero
```

O/P

```
Catch (ArithmeticExcep
```

Types of Exception

i) Checked exception
ii) unchecked exception

i) Checked Exception :-

→ Checked Exception are Compiler known Exceptions and the compiler will force you to handle immediately

→ All checked exception will inherit Exception class

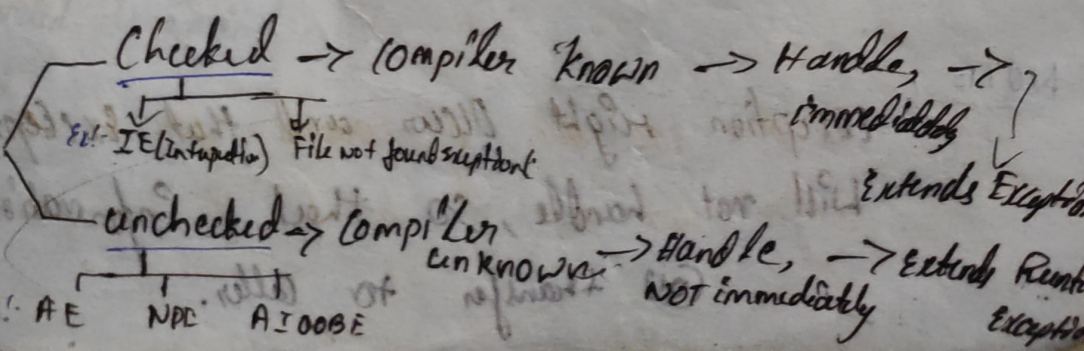
Ex: Interrupted exception, File not Found Exception

ii) unchecked Exception

→ unchecked exception are unkown to the compiler and the compiler will not force you to handle immidiately.

→ All unchecked Exception will inherite Runtime Exception class.

Ex: Arithemetic Exception, Array index out of bounds Exception, Nullpointer Exception



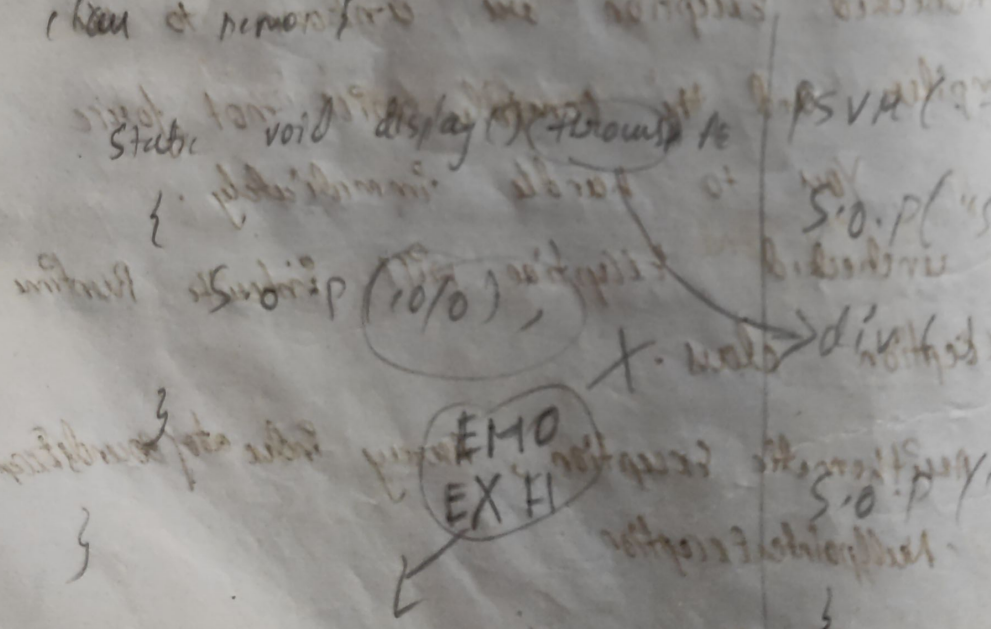
throws:

→ "throws is a keyword which is used to indicate the caller about the possibility of the Exception"

→ throws can be used with both checked and unchecked exception, but majorly used with checked exception.

→ throws can be used with both method and constructor but majorly used with method declaration.

→ throws is also used to transfer the exception to the caller side



NOTE: Exception might occur and that exception will not handle, so that information (or) transfer to caller.

that ... to caller.
give

Ex:-

```
class ThrowDemo1 {
  static void disp() throws ArithmeticException
    S.o.P ("0/0");
  }
Public static void main (String[] args) {
  System.out.println ("start");
  try {
    disp();
  }
  Catch (ArithmeticException e) {
      S.o.P ("Invalid Denominator");
    }
    S.o.P("End");
  }
}
```

O/P

```
Start
Invalid Denominator
End
```

Ex1.2

```
class ThrowsDemo{
    static void display() throws Exception{
        for(int i=1; i<=5; i++){
            S.O.P(i);
            Thread.sleep(1000);
        }
    }
public static void main(String[] args){
    try{
        display();
    }
    Catch(Exception e){
        S.O.P("Handled");
    }
}
```

if we create non-static method this will create an object.

O/P

1 → time delay 1000 milli sec → means 1 second.
2
3
4
5

Ex 3

```
package org;
import java.io.*;
class ThrowsDemo3 {
    static void readData() throws FileNotFoundException {
        FileReader f = new FileReader("dinga.txt");
    }
    public static void main(String[] args) {
        try {
            readData();
        }
        catch (FileNotFoundException e) {
            S.O.P("File not present");
        }
    }
}
```

O/P

File Not present.

NOTE: It is possible to indicate the caller about the possibility of multiple exception. With a single [we can declare multiple exceptions using throws]

Syntax Ex:-

```
static void readData() throws FileNotFoundException,
                              ArithmeticException)
```

NOTE Advatages

→ throws is to increase the efficiency of a program.

→ we should handle an exception only when it is necessary, therefore with the help of throws we can achieve it.

→ It is possible to use throw in main method But it is not good practice.

Why means.

Because main method is called by JVM we can not handle at the JVM side.

Important method present in throwable class:-

1) getMessage():-

"This method is used to return a small message about the Exception."

Syntax: Public String getMessage()

Ex:-

```
class Test {
  Public Static void main (String[] args) {
    int[] a = {20, 24, 30};
  try {
    S.O.P(a[100]);
    }
  catch (Exception e){ //Exception e = new ArrayIndexOutOfBounds
                                          Exception();
   S.O.P (e.getMessage());
   String message = e.getMessage();
       S.O.P (message);
   }
S.O.P ("-----");
  try {
     S.O.P(100/0);
    }
  catch (Exception e){ // Exception e = new ArithmeticException();
   S.O.P(e.getMessage());
   String msg = e.getMessage();
} }  S.O.P (msg);
```

O/P

Index 100 out of bounds for length 3
Index 100 out of bounds for length 3
by zero
by zero

(2) printStackTrace():-

→ This method is use to print the complete information about the Exception

→ printStackTrace():- print * Exception name
* Message about Exception
* Line numbers.

Ex:-

```
class Test {
    Public static void main(String[] args){
        S.O.P("START");
    try {
        S.O.P(10/0);
    }
    catch (Exception e){
        e.printStackTrace();
    }
    S.O.P("End");
}
}
```

O/P START
java.lang.ArithmeticException: / by zero at day1.Test.main(Test.java:7)
End

NOTE:- when an Exception occurs we need to avoid it, only when we fail to avoid the Exception we need to handle it

Program to avoid null pointer Exception

```
class student {
int id=100;
Public static void main (String[] args) {
S.O.P ("start");
Student s=new student();
// s= null  ——> Null pointer Exception.
if (s!=null){
S.O.P (s.id);
}
S.O.P ("End");
}
}
```

O/P

```
start
100
End
```

Custom Exception / user-Defined Exception

→ Based on the project it is some time necessary to create user defined exception.

→ Any Exception which the user (or) programmer will create explicitly is called custom Exception / user-Defined Exception

Rules for creating custom Exception

i) Create a class with the exception name.

ii) The class should inherit Runtime Exception class to create unchecked exception (or) inherit exception class to create checked exception.

iii) opitionaly override getmessage().

iv) throw: Invoke the exception using throw keyword, handle it using try & catch block and provide suitable solution.

Ex:-

```
// Rule 1                                    // Rule 2
class InvalidPinException extends RuntimeException {

    Private String message;

    InvalidPinException(String message){
        this.message = message;
    }
```

@Override // Rule 3
public String getMessage() {
    return message;
}

throw :-

→ throw is a keyword which is used to invoke (or) throw an object of Exception type.

→ throw is majorly used with user defined Exception

Syntax:- throw ObjectOfExceptionType;

(or)

throw new ExceptionName(); → throw new ClassName();

Ex:-2

Package customeException;

import java.util.Scanner;

class ATM {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter pin NO:");
        int pin = scan.nextInt();

        if(pin == 123){
            S.O.P("valid Pin");
        }
    }

```
else{
    try{
        throw new InvalidPinException("please check the
                Pin NO");
    }
    catch(InvaidPinException e){
            S.O.P(e.getMessage());
    }
}
```

// InvaildPinException e: new InvaildPinException("Please check the pin NO");

O/P

Enter pin NO:

456

Please check the pin NO

Ex(3) Amount withdrawing Scenario:

Create an unchecked exception called as InsufficientBalanceException and Override getMessage() and invoke it during a suitable Scenario.

Sol:-

```
// Rule 1                                    // Rule 2
Class InsufficientBalanceException Extends RuntimeExc
    private String message;
```

```
InsufficientBalanceException (String message){
    this.message = message;
}
@Override          // Rule 3
Public String getMessage()
{
    return message;
}
Public static void main (String [] args) {
Scanner sc = new scanner (system.in);
S.O.P ("Enter amount");
int amount = scan.nextInt();
int balance = 5,000;
if (amount <= balance)
{
    S.O.P("transaction Successful");
}
else {
try{   // Rule 4
    throw new InsufficientBalanceException
                ("Insufficient Balance.");
};
catch (InsufficientBalanceException e){
        S.O.P(e.getMessage());
}
}}
```

O/P

Enter amount

10,000

Insufficient Balance

(Ex. 9)

Create a checked exception called as InvalidPassword Exception and override getMessage() and invoke it during a suitable scenario.

```
class InvalidPasswordException extends Exception {
    Private String message;
    InvalidPasswordException (String message) {
        this.message = message;
    }
    Public String getMessage () {
        return message;
    }
Public static void main (String[] args) {
    Scanner sc = new Scanner (System.in);
    S.O.P ("Enter PatternNo");
    int PatternNo = sc.nextInt();
```

```
if (Password == 9845) {
    S.O.P ("valid pin");
}
else {
    try {                                   Exception
        throw new InvalidPasswordE("please check your
                                    password");
    }
    Catch ( InvalidPasswordException e) {
        S.O.P ( e.getMessage ());
    }
}
```

O/P

Enter password

1234

please check your password

Example 5

Create a unchecked exception called as AgeInvalidException and Override getMessage() & Invoke it during Scenarios.

Sol:

```
class AgeInvalidException extends RuntimeException{
    Private String message;
    AgeInvalidException (String message){
        this.message = message;
    }
    Public String getMessage(){
        return message;
    }
}

Public static void main (String[] args){
    Scanner scan = new Scanner (System.in);
    S.O.P ("Enter age");
    int age = scan.nextInt();
```

```
if (age > 21)
 {
   S.O.P ("Get married");
 }
else {
    try {
        throw new AgeInvaildException ("Have patience!");
        }
    Catch (Exception e){
        S.O.P (e.getMessage());
    }
}

// AgeInvaildExeption obj = new AgeInvalidException("Have
                            patience");
// throw obj;
```

Different b/w throws and throw

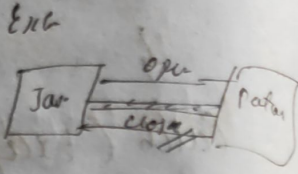
throws	throw
throws is used to indicate the caller about possibility of an exception	throw is used to invoke an object of Exception
Error	Exception

finally block

finally block is a set of instruction (or) a block of codes which get executed always irrespective of an exception occur (or) not.

Syntax

```
finally
{
  ---
  - -
  -
}
```



Ex:-1

```
class FinallyBlockDemo {
  Public static void main (String[] args) {
    S.O.P ("Start");
    try {
      S.O.P (10/0);
    }
    Catch (Exception e) {
      S.O.P ("Invalid");
    }
    finally {
      S.O.P ("Inside finally Block");
    }
    S.O.P ("End");
  }
}
```

Package

→ Package are noting but folder.

→ Package are used to store classes and Interface

→ By creating package searching becomes easy.

→ Better Maintenance of the Program.

Example :-

```
Package com.google.gmail;
class Inbox {
}
```

Scanner

→ Scanner is a pre-defined class present in java.util package.

→ Scanner class is used to accept input from the user.

Rules to achieve Scanner

1) Create an object of Scanner class
2) pass system.in to the Constructor call
3) Import Scanner class from java.util package
4) Make use of pre-defined methods to accept input.

Important method used while Scanner class

1) byte - nextByte()
2) short - nextShort()
3) int - nextInt()
4) long - nextLong()
5) float - nextFloat()
6) double - nextDouble()
7) boolean - nextBoolean()
8) String - next(). (or) nextLine()
9) char - next().charAt(0)