Attributes + Behaviours ()

Class

A class is a named group of properties and function.

Class = logical Construct / Template

Object = physical reality // Occupies space in memory

~~address~~ = Instance of Class

What is Object Oriented Programming?

It revolves around the object = Code + Data.

→ Types of Lang
- → Procedural
- → Functional
- → Object Oriented.

Static vs Dynamic Language

Static

- Performs type Checking at ~~Runtime~~ Compile Time
  - Before the code transformed to M/c Lang
- → Errors will show at Compile Time.
- → Declare datatype before use it.
- → More Control.

int a = 10

a=10 ✓
a="manual" ✓

classmate
Date
Page

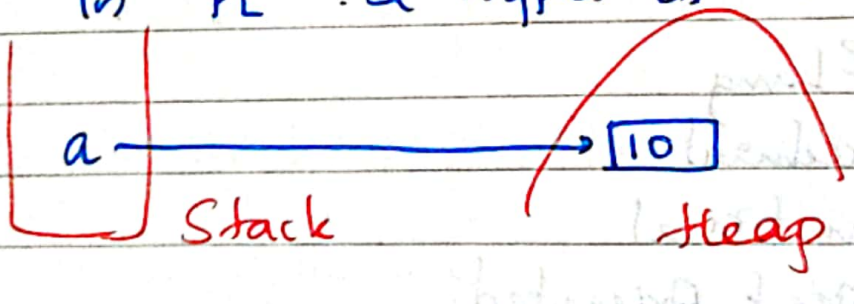
Dynamic — eg: python.

- Performs type checking at runtime.
- Error might not show till program is run
- No Need to declare datatype of var:
- Saves time in writing code but might give error at runtime.

How memory is managed:

Memory Management

In PL . 2 types of Memory.



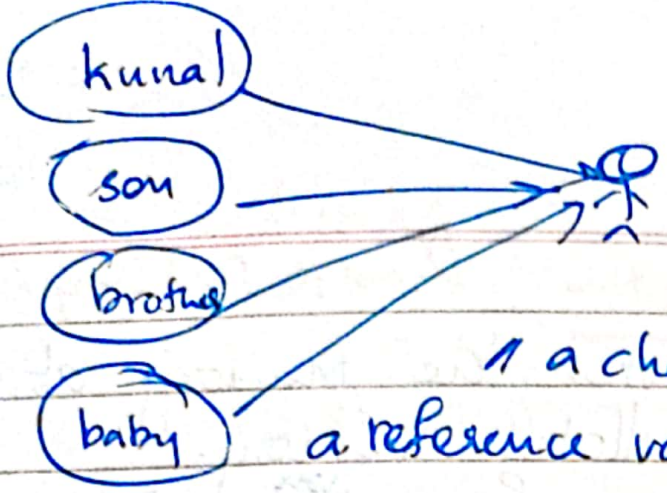
When you write a=10 ← object

↑ ref variable

Variables Stored in Stack.

Object / Value Stored in the Heap Memory

a is the pointer to the obj in heap memory



↑ a change made through a reference variable to an object
- orginal object will change
- It will be changed for all.

eg: Haircut.

Garbage Collection

Obj. doesn't have a reference variable then garbage coll^n will remove it.

a ⟶ 35 a = 35
a = 10.

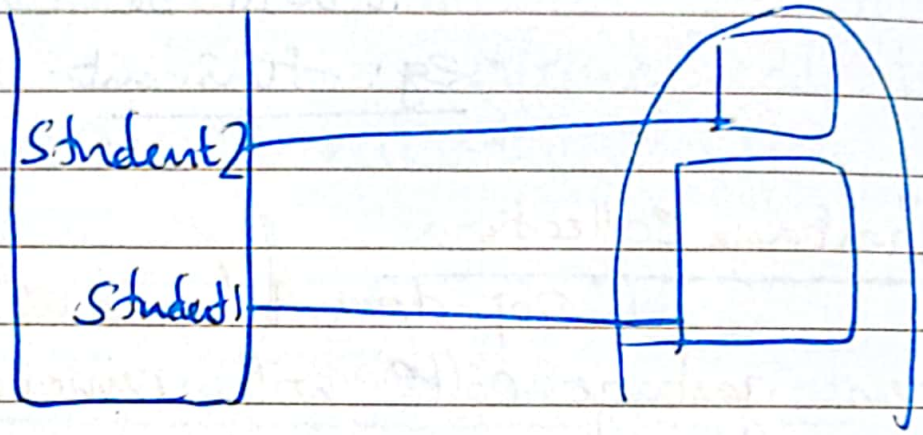
a ⟶ 10 So 35 kept alone in memory. It is removed from memory by GC.

Dot Operator - Seperator in Java
link Object reference Variable — Obj
to Instance variable — Variables inside the object.

Student1.rollnumber

new Operator - Dynamic Memory Alloc

Allocates the memory at runtime
Dynamically allotes memory.
& returns a reference var to it.



All Class Objects in Java allocated dynamic

```
Student obj = new Student();
```

Compile time
- Pgm checks error
- converted to byte code
- Byte Code to M/c code
- M/c using JVM

Run time
- After Code Conver
- Now Code Runs
→ This time Mem allocated.

In Java You can't access memory addresses using Pointers because
⇒ Java Security.

Student = Class

```
rno : = 0 // default
name = null
marks = 0.0
```

new S1

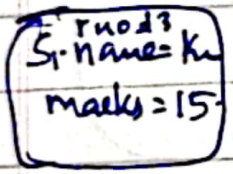
print (S1.rno) = will be default.

S1.rnow = Show me the rno data in the S1 Object

then S1.rnoow = 13

S1.name = kunal

S1. Marks = 15



If we say S1. salary // not mentioned in class, so it can't do in Java

Java is a Static type language.

You can do it in Python.

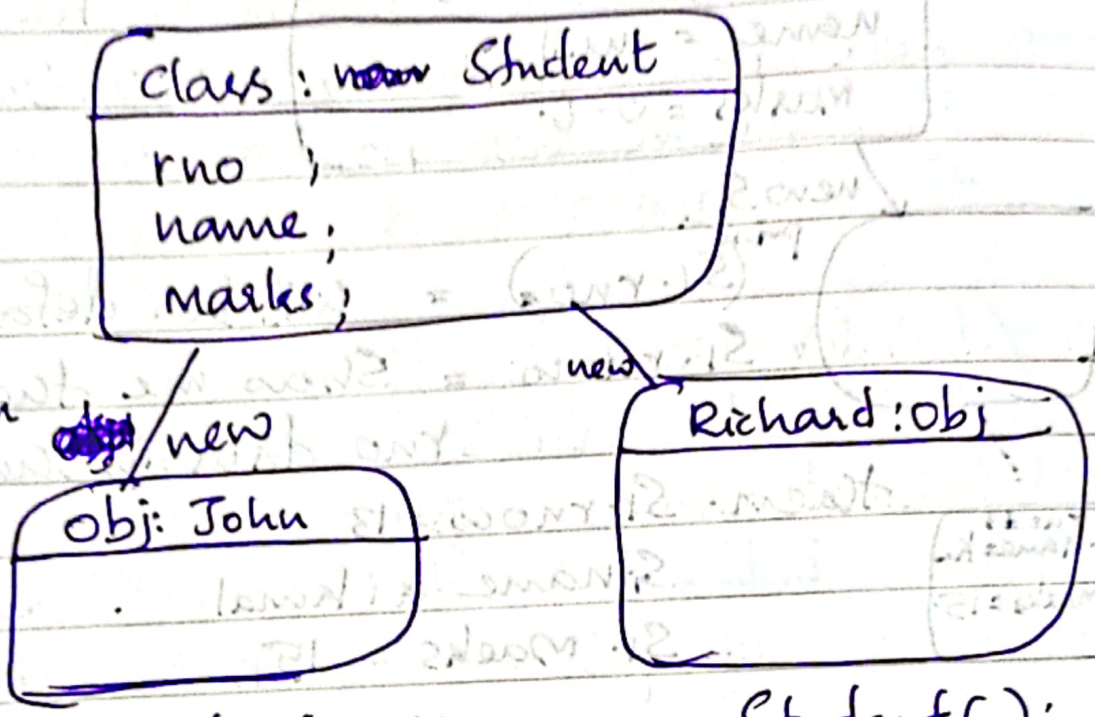
CONSTRUCTOR

Defines what happens when an obj is been created.

Return type is the Class itself.

Bcoz it creates the ~~obj~~ type obj of this class.

Use of Constructor




```
student. John = new Student();
Student . Richard = new Student();
```

print = default values.

```
John. rno = 13   ;
john . name = Jhon ;
john . marks = 88  ;
```

```
Richard. rno = 14
richard. name = Ric
richard . marks = 9
```

Q. Why we don't use new keyword in Primitive Datatype double, char, short, Boolean, byte, int, long, float,

In Java Primitive Datatypes are not implemented as Objects.

Objects are stored in Heap Memory.

So in Stack Memory.

new is used for Objects. Others are Variables.

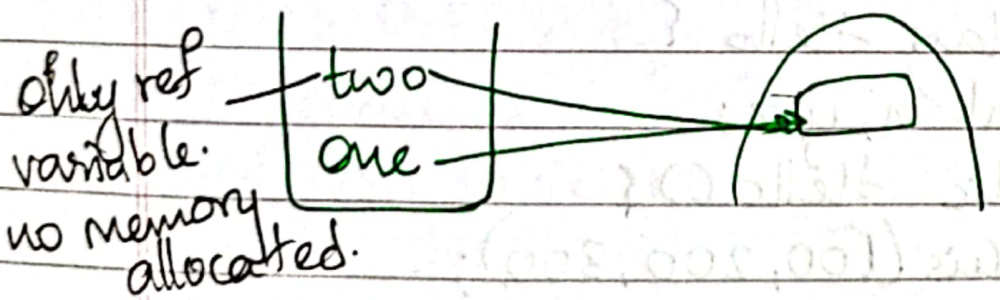
Why Java doing this?

Efficiency, To make it fast

Putting in Heap Memory its all make it slow.

Python is a Slow Language.

```
Student one = new Student ();
Student two = one;
```



this keyword

```
public Class Hello {
    int x = 5;
    public Hello () {
    }
    public static void Main (String [] args)
    {
        Hello obj = new Hello ();
        S.O.P (obj.x);
    }
}
```

Output

x = 5.

Default value

x = 0;

```
public Class Hello {
    int x, y, z;
    public Hello () {
        this (100, 200, 300);
    }
    public Hello (int x, int y, int z)
    {
        this.x = x;
        this.y = y;
        this.z = z;
    }
```

```
public static void main (String[]args)
{
    Hello obj1 = new Hello();
    Hello obj2 = new Hello (500, 600, 700)

  S.O.P ("x = " + obj1.x + " y= " obj1.y + "z=" obj.z)
  S.O.P("x= " + obj2.x + "y" obj2.y + "z=" obj.z);
}.
```

Java 2 types of Constructor

Non argument Constructor.

- Default — gives default value — by compiler.
- Parameterised.

- No Return type.
- Same Name
- Can have more than one Constructor.

Copy Constructor .

```
int name;
int age;
public Student (String name, int age)
{   this.name = name;
    this.age = age;
}
```

Para Const

```
public Student (Student std){
    this.name = std.name;
    this.age = std.age;
}

Student std = new Student ("Rahul", "25");
Student cStd = new Student (std);
```

Copy Constructor use Current class object to initialize new object.

cStd. obj initialised using Std. object.

Constructors cannot be inherited in Java
In Inheritance subclass inherits the members of Super class except Constructors.

Break & Continue

Break = Loop or Switch Case.

OOP Objects are primary source to implement.

Objects - Real world entities that has their own properties and behaviour.

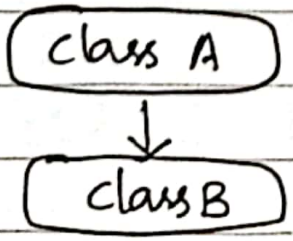
Classes - Blueprint from which an objects properties and behaviours are decided.

ABSTRACTION, INHERITENCE, ~~ABSORPTION~~ ENCAPSULATION, POLYMORPHISM

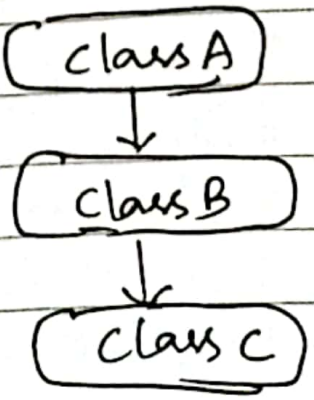
Inheritence - Is the property of an object to acquire all the properties and behaviour of its parent object.

Types of Inheritence

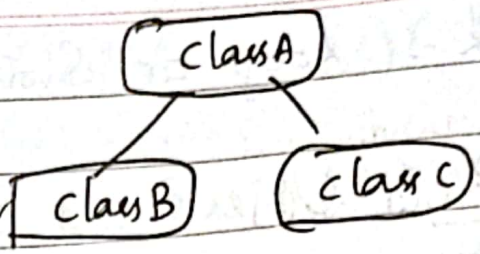
Single Inheritence



Multilevel Inheritence



Hierarchical -



Advantages

- Code Reusability
- Method Overriding.
- Data Hiding - Base class can keep data private, such that the derived class will not alter.

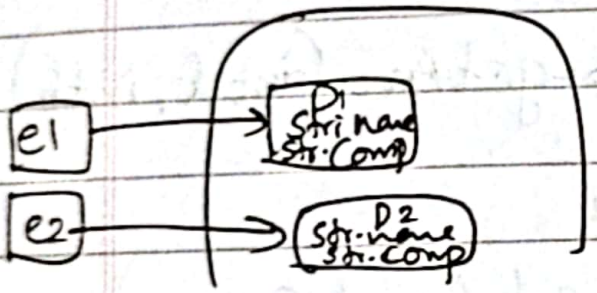
Disadvantage

- Tightly bounded.
  - Can't use parent & child seperately.

Static keyword

```
class Employee{
    String name;
    String company="Smart herd";
}
Employee e1=new Employee(); // D1
```

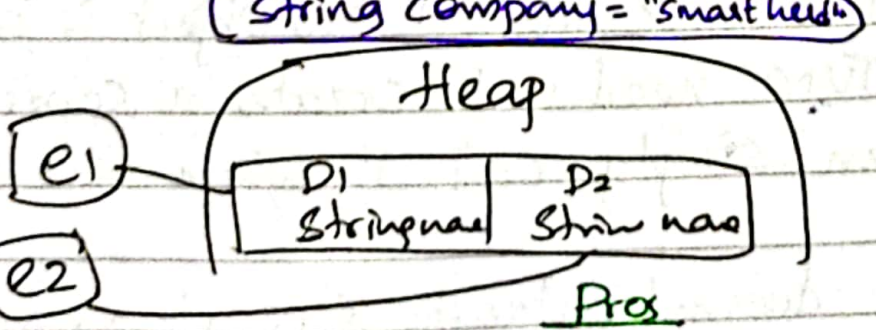
non static var



Cons
- String Company read twice
- Memory waste.

```
Class Employee {
    string name;
    static String Company = "Smart herd";
}
  Employee e1 = new Employee (); // D1
  Employee e2 = new Employee (); // D2
}
```

Static var.



Pros
→ String Company gets memory for just once.
→ Memory Efficient.

Employee.company

main is very I thing running. How can you create an obj, How can you run the program without creating obj.

static - Static direct call;

- Static variables are not associated with objects.
- It belongs to Class
  Syntax ⇒ className. variableName

Adv ⇒ Memory Efficient.

Static Method

```
class Rectangle {
    public static int getArea(int l, int b)
    {   return lxb;
    }
int area = Rectangle.getArea(20,10);
```

(area: variable; Rectangle: className; getArea(20,10): static fn call)

→ Static Methods do not require object.
→ Static Methods are just a Part of Class not Object

You can't access non-static member in a static fn → doesn't depend obj
static depends obj

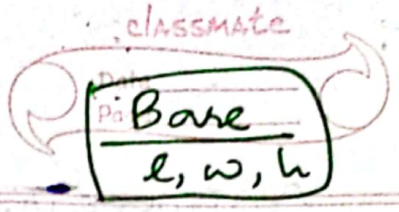
Why main() is Static ?

- To call a static method No Object is required.
- So JVM need not create a seperate obj
- It can simply call it when the class loads.

JVM doesnot allocate seperate memory for Main method.

Can't use this is static → bcoz it refers obj

Inheritence




```
child class extends Base Class {
        int weight;
}

Child child = new Child ()
        child.l ;
        child.w ;
        child.h ;
Super child.weight ;
```

first we declare super constructor var then child var.

[private members of Base class cannot be inherited by Child Class]

You can do Private whatever only in that file.

ABSTRACTION

ABSTRACT CLASS

- It is A class is declared as abstract class.
- You cannot create object of abstract class.
- It may or may not contain abstract method

Abstract Method

- ~~A class that is declared abstract.~~
- No Method Body.
  You can write code in abstract method body.
- It is mandatory to override the abstract method in child class.

Abstraction

Hiding Implementation details thereby showing only essential ones.

Ways to Implement Abstraction

- Abstract Classes
  to achieve 0-100% abstraction
- Interfaces
  (to achieve full or 100% abs")

Abstraction
- no object
- if method Ab then class
- have static
- no abstract Constructor.

No Multiple Inheritence.

In java, abstraction is achieved by abstract methods.

Abstract methods do not have an implement it only has method signature. // declaration

```
public abstract void Method1();
```

→ If a class is using an abstract method (class) they must be declared as abstract.

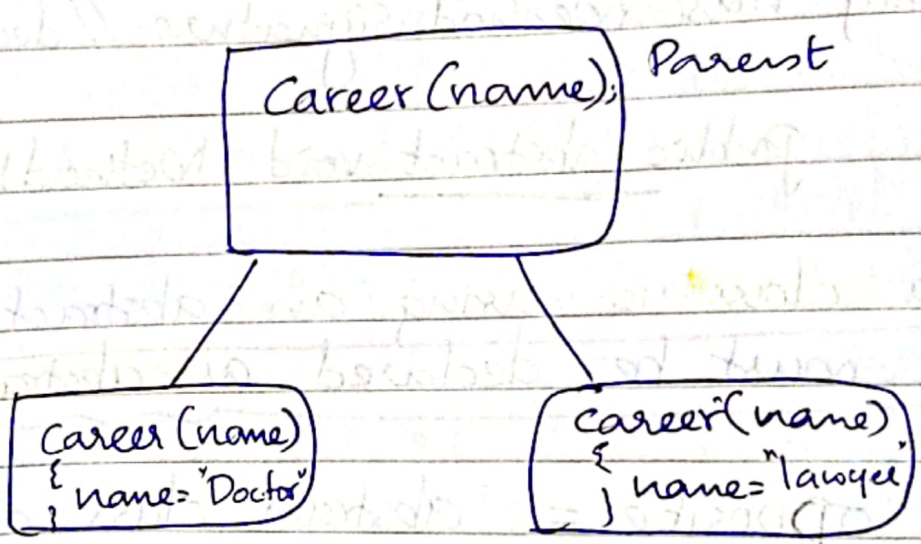
→ But opposite = abstract class doesnot necessarily have an abstract method

```
public abstract class Player {
    public abstract void Method();
}
```

Points to remember

- An abstract class must be declared with an abstract keyword
- It can have abstract and non abstract methods
- It cannot be instantiated. (with new keyword)
- It can have Constructors and Static Methods also. and final

- It can have final methods which will force the subclass not to change the body of the method.



Abstract method - No body
So it totally depend on child class to provide body.
So inorder to use it, it has to override

```
abstract void career (string name);
```

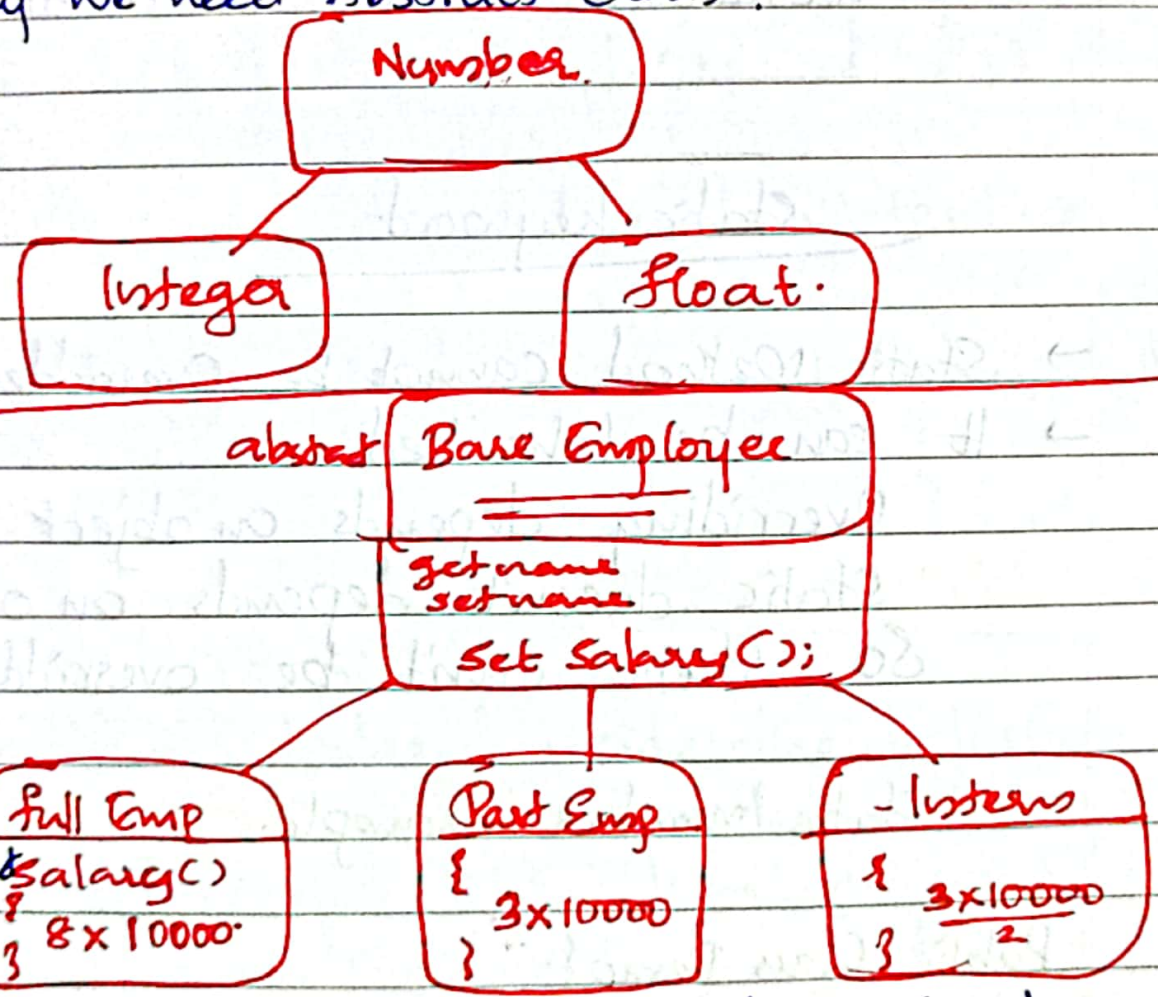
The Child Class Must Override them.
It cannot use undefined if Parent class bcoz there is no body.

If you definitely know, functions in the Parent Class needs to be overridden then make it Abstract and give declaration only.

→ The Code looks clean

If we call Concrete Constructor instead of Abstract Office user can create Base Employee object and create issues.

Why we need Abstract Class?



- All the basic data will be stored and entered.
- Abstract method Salary compels user to define that function (which is different for different classes). childclass

final keyword

→ Used to create Constants.
→ Can be used to prevent Overriding.
→ Doesn't Support Inheritence.
↳ 'final' class { } can't be inherited.
→ If class is final, then implicitly methods are declared as final.

Static keyword.

→ Static Method cannot be Overridden.
→ It can be Inherited.
Overriding depends on object
Static doesn't depends on object.
So Static can't be overridden.

Static member example.

```
Public Class Demo{
    [illegible]
static int Student Strength=1
  public Demo()
  {
     S.OP("You created. Student:"+ Strength);
  } Strength++
```

every time I create a student using Constructor the variable Strength get called and Counted.

public static final constants;
abstract methods ();

INTERFACE

not a class.

→ We write Interface A. not Class A

→ Interface is Basically contains Abstract functions.

```
interface interfaceName {
    variables;
    Methods();
}
```

By default abstrac

- All methods declared inside interface are by default abstract // public too.
  Compiler add abstract in front of Method name in Interface

- We can't create Concrete Methods in Interfa

- Variables declared inside the Interface are Public static final by default.
  ↳ (bcoz can't have constructor, so no inherit)

- Use implements keyword

```
interface B {
}
class A implements B {
}
```

Abstract Class vs Interface.

Types of method:

- Interface can have only abstract metho
- Abstract Class can have - Abstract, Concrete, private, default, static meth

Variables

I - By default final, static & public

A - Can have nonfinal & non static, final & static

Implementation

Abstract class can provide the implementn of interface.

Interface can't provide the implemn of an abstract class.

Type of variables

I - Can only have Static & final

A - final, nonfinal, static, non static

keyword

I - Implements
A - extends.

Multiple Inheritence

I - Supports

Access Modifiers for Members

I - Public by default
A - can have private, protected.

- I/f cannot be instantiated just like abstract Class. // no constructor. for i/f but no obj for abstract
  So constant fields // That's why final

- An abstract class can be extended using extends and implements multiple i/f
  An interface can be ~~implemented using~~ 'implements' extend another i/f

When to use abstract classes?

→ There are some related classes sharing some lines of code

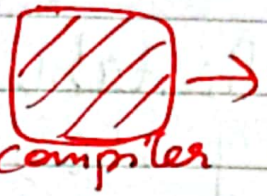
→ Interface?

→ Total Abstraction. All methods declared within an interface must be implemented by the classes that implements this interface.

→ A class can implement more than one interface. = Multiple Inheritance.

→ You want to specify the behaviours of a particular datatype. but are not concerned who implements its behaviour.

```
interface Print {
    int Min = 5;
    void print();
}
```




```
interface Print{
 public static final
        int min=5;
 public abstract print();
}
```

eg:

```
interface Printable {
    void print();
}

class Office implements Printable {
 public void print()
 {    S.O.P ("Hello");
 }

P.S.VM {
 Office office = new Office();
  office.print();
}
```

class won't support Multiple Inheritence
Only interface supports it.

Multiple Inheritence.

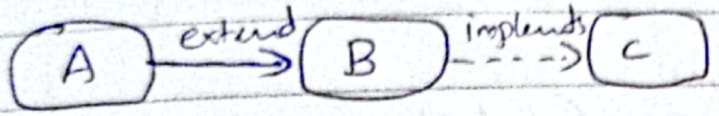
```
interface Printable {
        void print();
}

interface Showable {
        void show();        void print();
}

class Test implements Printable, Showable{
  public void print() {
      S.O.P ("Hello");
    }
  // public void show()
  // {
  //      S.O.P ("Hai");
  }//}

  P.S.V.M {
   Test obj = new Test();
   obj. print();
   //obj. show();
}
```

O/p

Hello

In Interface :-

→ Interface Inheritence ✓




```
interface A { }
interface B extends A { }
class C implements B { }
```

→ Static Method ✓ also default

also default: So Method Body Needed.

→ Marker / Tagged I/f.

eg:- Serializable, Cloneable, Remote.

used to provide some essential info" to JVM so JVM may perform useable info".

```
public interface Serializable { }.
```

→ Nested I/f ✓

Abstract Class example:

```
abstract class Bike {

    Bike ()
    {   S.O.P ("bike is created");
    }
    abstract void run ();
     void change gear(){
          S.O.P. ("gear changed");
     }
}

class Honda extends Bike {
    void run()
    {   S.O.P ("running safely");
    }

class Test {
     P.S.V.M
  Bike obj = new Honda();

     obj.run();
     obj. changeGear();
}
}.
```

OOP

OOP

Is a methodology or paradigm to design a program using classes and objects.

Object - Any entity that has state and behavior.
- Instance of a class.
- Contains an address and takes up some space in memory.
- Objs can communicate without knowing the details of eachother's code.

Class - Collection of objects.
- logical entity.
- Blueprint / template.
- doesn't consume any space.

OOP Concepts

→ Object → class
- Inheritence
- Polymorphism
- Abstraction
- Encapsulation.

Advantages

→ OOPs - develop^nt & maintenance easier. easy to manage as project size increas

→ Data hiding

→ Provides the ability to simulate real-wor event much more efficiently.

OBJECT

Object 3 Characteristics

→ State - data of an object variables
→ Behaviour - functionality method.
→ Identity - by an ID - used by JVM interv

Definitions

- real world entity
- runtime entity
- entity which has state & behaviour
- instance of a class.

Instance variable

Variables inside class, outside the function body.
doesn't get memory at Compile Time.
gets memory at Runtime when an obj created.

3 ways to Initialize Object

(1) → Reference Variable

```
Student S1=new Student
S1.id = 101
S1.name = "Sona"
```

(2) → Initialization through Method

```
class Student {
    int rollno;
    String name;
  void setRecord (int r, String n)
  {
      rollno = r;
      name = n;
    }
}
```

```
Student S1 = new Student()
S1.SetRecord (111, "Aryan");
```

→ Initialization through Constructor.

Different methods to create Object

→ new keyword
→ newInstance() method
→ Clone()
→ deserialization
→ factory method

Anonymous Object

```
new Student(); // anonymous obj
new Student().id(5); // calling it.
```

→ Constructor returns value?
Yes, current class instance (but cannot use return type)

→ Constructor can perform other tasks such as obj creation, starting a thread, calling a method etc.
- perform as a method.

Static

Static keyword

- mainly for memory management.
- Saves memory.

1. **Static Variable**

used to refer common property of all objs.
ies. which is not unique for each object
eg- Population, Company name.
(counter)

- Static variable gets memory only once in the class area at the time of class loading

2. **Java Static Method**

- Static method belongs to class rather than the object of class.
- Static method invoked without obj
- Static method can access static variable and change the value.

2 restrictions

→ Cannot use nonstatic data or method directly.
→ Cannot use super and this.

this

this keyword

- Ref variable that refers to current object.

1. to refer current class instance variable

```
{
    this.rollno = rollno;
    this.name = name;
}
```

2. to invoke current class method

```
{
    void m() { }
    void n() {
        m();   ⇒ this.m();
    }          // compr autoly adds
```

3. to invoke Current Class Constructor.

- constructor chaining

```
class A {
    A() {
        this.(100, Jack);
    }
    A (int num, String name).
    {
    }
```

4. to pass as an arguement in the method.

```
class S{
    void m(S obj)
    {    S.O.P("method invoked");
    }
    void p(){
        m(this)
    }
P.S.V.M {
        S obj = new S();
        obj.p();
```

It is mainly used in event handling. In event handling / in a situation where we have to provide reference of a class to another one. It is used to reuse one object in many methods.

5 to pass as an arguement in Constructor Call

```
A(B obj)
{   this.obj = obj;
}
```

6 Return Current Class Instance.

```
return type method(){
        return this;
```

INHERITENCE

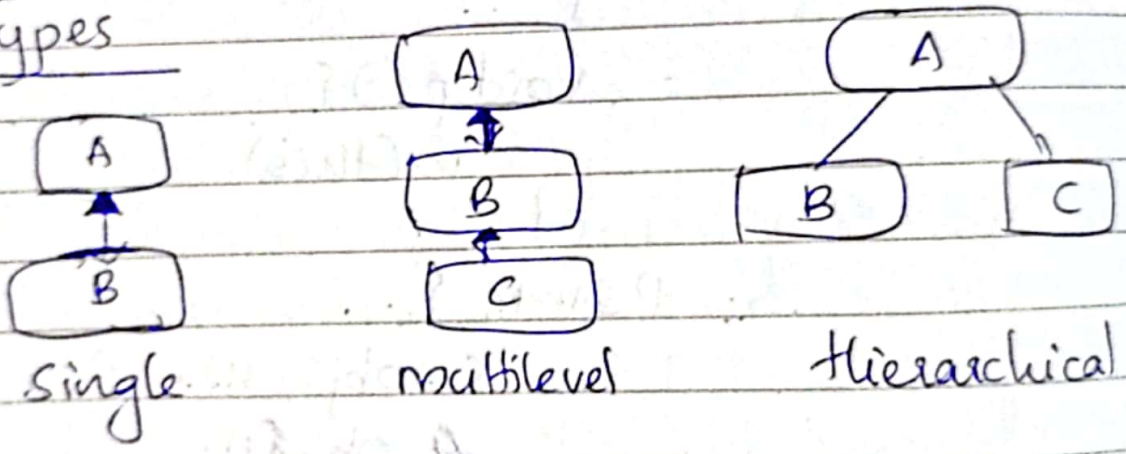
Inheritence - "extends"

Why use Inheritence?

→ for Method Overriding.

→ Code Reusability.

Types



single multilevel Hierarchical

eg:

```
class Animal {
    void eat()
    { S.O.P("Animal is eating;");
    }
class Dog extends Animal {
    void bark()
    { S.O.P("Dog is barking;");
    }
class Cat extends Animal {
    void meow()
    { S.O.P("Cat is meowing");
    }
class Test {
    P.S.V.M {
        Cat c = new Cat();
        c.meow();   c.eat();
```

AGGREGATION

Aggregation

If a Class have an entity reference.

Address.java

```
public class Address {
    String city, State, country; }

public Address (String city, String State, String Cou
{
    this.city = city;
    this.State = state;
    this.Country = country;
}
```

Emp.java

```
public class Emp {
    int id;
    String name;
    Address address;
public Emp (int id, String name, Address addres
{   this.id = id
    this.name = name;
    this.address = address;
}
void display()
    { ___ ___ ___ }
}
```

```
Psvm(){
    Address address1 = new Address("clt","KL",
Emp e = new (111, "vasan", address1);
}
```

Why use ?

- Code Reusability.
- Here no Inheritence Relation.
- Inheritence, only if relationship is maintained throughout the lifetime of the objects involved.

Otherise aggregation is the best.

METHOD OVERLOADING

Method Overloading

If a class have multiple methods having same name but different in parameters.

Advantages:-
Increases readability of the Programs.

Different ways to overload:-
- By changing no: of arguements
- By changing the data type.

- In Java, Method Overloading is not possible by changing the return type of the method only. - double Add (int) & int add (int, int)
Adder.add(11, 11) Compiler ???

1. class Adder {
int add (int a, int b) { return a+b; }
int add (int a, int b, int c) { return a+b+c; }

2. class Adder {
int add(int a, int b){ return a+b; }
double add(double a, double b) {return a+b; }

- Can we overload Java main() method?

Yes, Jvm calls main() method which receives string array as arguements only.

```
class Test {
    P.S.V.M (String[] args) { }
    P.S.V.M (String args) { }
    P.S.V.M() { }
}
```

✓

Type Promotion

Type is promoted if no match found.

char → int, long, float, double
float → double
long → float, double
int → long, float, double.

METHOD OVERRIDING

Method Overriding

If a subclass provides the specific impl^{n} of the method that has been declared by one of the Parent class.

Usage

- Provide Specific implementat^{n} of method which is already provided in supee class.
- is used for runtime Polymorphism.

Rules

- Same name as Parent Class
- Same Parameter as Parent Class.
- Inheritence Relationship.

Can we override Static method?

No, it is bound with class not object.

Static ⟶ Class Area

Obj ⟶ Heap Area.

Main method?

No, Bcoz it is Static.

SUPER.

Super

- Reference keyword for immediate parent Class Objec

Usage

- Refer immediate Parent Class variable.
- Invoke immediate Parent Class method
- Invoke immediate Parent Class Constructo

Super() - is added in each class constructor automatically by compiler if there is no super() or this();

Super() - first stmt in Constructor

FINAL

final

- used to restrict the user.

Final can be
- variable // initialize constant
- Method
- class.

Final keyword
variable —— stop Value Change
method —— stop Method Overriding
class ~~Overriding~~ —— stop Inheritence.

Q final method inherited? ✓
- Yes inherited but cannot override.

Q Blank / Uninitialized final var?
↳ Not initialized at the time of declaration.
- Used if we want runtime initial^{n}.
- Initialized through Constructor only.

- final Parameter - int cube(final int n) {n=n+2;} error

- No Constructor – Final
Bcoz Constructor never inherited.

POLYMORPHISM

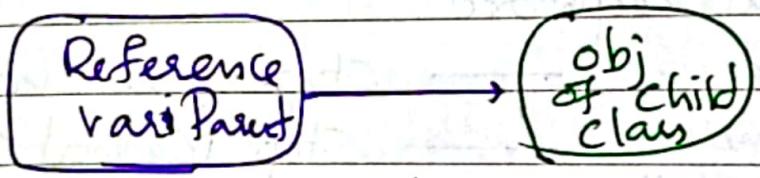
- RUNTIME

Polymorphism

- Single action in different ways.

Runtime Polymorphism = Dynamic Method Dispatch is a process in which a call to an overriden method is resolved at runtime.

upcasting.




```
Animals animal = new atHorse();
└parent┘        └child┘
```

Since it refers to subclass obj and subclass method overrides the Parent Class method, the subclass method is invoked at runtime.

```
public class Animal {   int weight = 100;
    void eat(){
        S.O.P(" Animal eating"); }

public class Horse extends Animal {   int weight = 300
    void eat(){
        S.O.P(" Horse is eating");
        }
```

```
class Test {
    P.S.V.M {
        Animal obj = new Horse();
        obj.eat();
    S.op("obj.weight")}
```

O/p

Horse is eating. 100 // superclass var.

- Since method invocation is determined by the JVM not compiler = Runtime Polym.
- Data members can't overriden so no polym.
- Overriding not, immediate Superclass fn invoked.

PACKAGE

Package

Group of similar type of classes, i/f, sub-packages.

- Built in
- User defined

Usage

- Similar category — So easy maintance
- Access Protection
- Removes naming collision

How to use Other Pkg

```
import package.*
import package.classname;
fully qualified name;
```

If you import a package - all classes & i/f not subpackages

Access Modifier

Access Modifiers

- Specifies the accessibility or scope of field, method, constructor, or class.

4 Types

— Private. — only within the class not outside the class.

- Default — only within the Package not outside the Package.

- Protected — within the Package and outside the Package through child class. only.

- Public — everywhere.

Private Constructor

- You cannot create obj of that class from outside the class.

A class cannot be Pvt/Protected except Nested

ENCAPSULATION

Encapsulation

- process of wrapping code and data together into a single unit.

How we make

- All data members private. (or protected)
- No we use getter and getter for modifying & viewing var

Adv

By providing only a setter or a getter you can make it

read only - getter method only
write only - setter method only

- Control over data.
  = Can give if (condition) in setter method
- Data hiding
- Encapsulate class is easy to test.
- Easy & fast to create // IDE in built getter/setter

```
package com.JavaSalma;
    public Class Student {

        private String name;

        public String getName()
        {
            return name;
        }

        public void SetName
                (String name)
        {
            this.name = name;
        }
    }

Class Test {
    P.S.VM
        Student s = new Student();
        s.setname ("Vijay");
        S.O.P (S.getName());
```

output

input

Abstraction

The process of hiding the implementation details and showing only functionality to the user.

eg:- Sending msg // You don't know the internal processing

Abstraction lets you focus on what the object does instead of how it does it.

There are two ways to achieve Abstraction

1. Abstract Class (0-100%)
2. Interfaces.

→ No instantiation
→ Constructors ✓
→ Static Method ✓
→ final Method ✓ - But no change in body

Abstract Class

- A class which declared as Abstract Class
- Can have Abstract & Non Abstract Methods
- Must Extended and its Method Implemented
- Cannot be Instantiated.
- It can have Constructors and Static Method also
- It can have final methods which will force the subclass not to change the body of the method.

abstract Class A { }

- No Object for Abstract Class.

- Can have a data member
- Can have a abstract method
- Can have a Concrete Method
- Can have a Constructor
- Can have a main method.

If you are extending an abstract class to Child class, Child Class must either provide implementation of the method or Make this class Abstract.

Interface

public static final Vars
public abstract methods

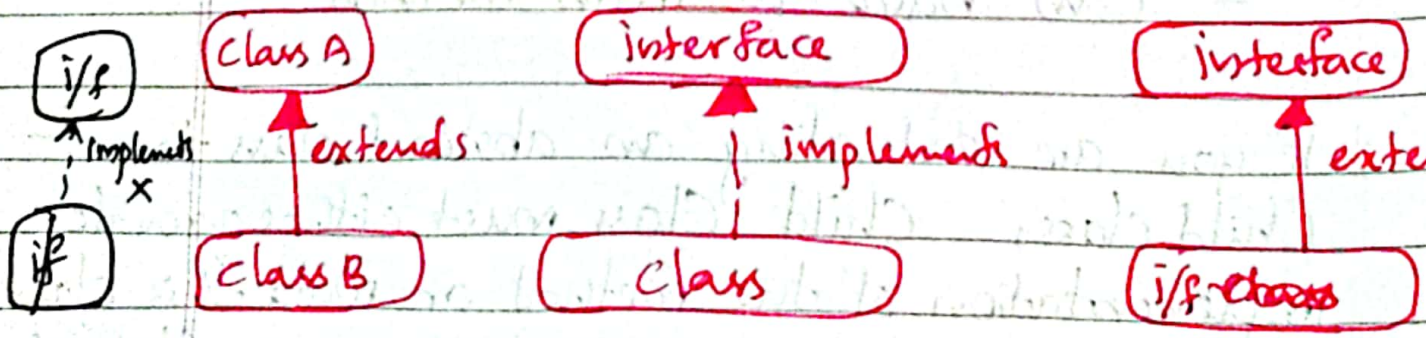
Interface in Java is blueprint of a Class.

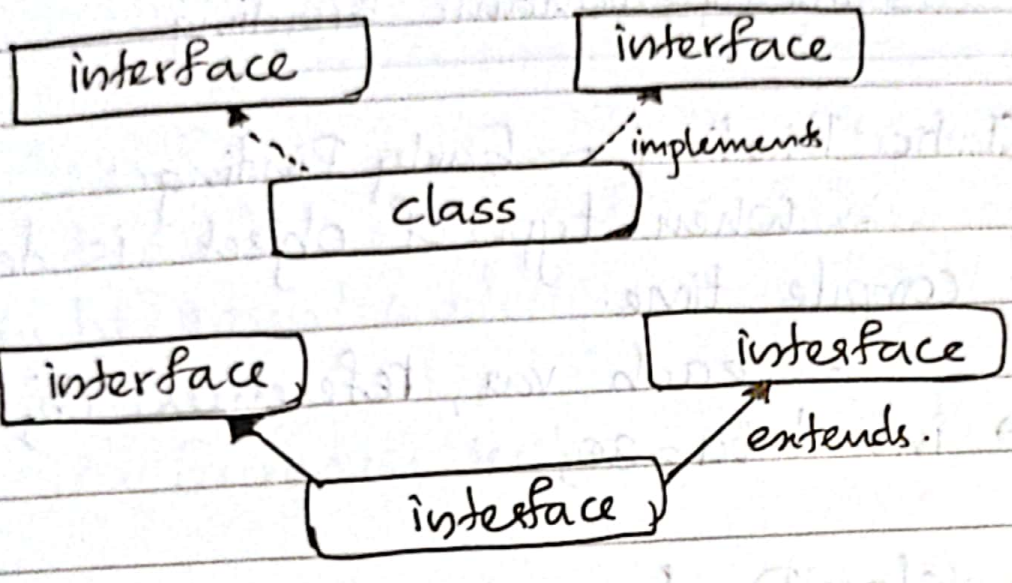
It has :-
- Static Constants
- Abstract Methods.

- No Constructor. // Because Not a Class.
- var = public static final (default)
- method = abstract (default)
- " = private & public (default)
= default and static method.

Why use Java Interface?

- To Achieve Abstraction.
- Support Multiple Inheritence
- Achieve loose Coupling





Why Multiple Inheritence through I/F?

Because in case of Inheritence there is no ambiguity. It is because its implement is provided by the implementation class.

Static & Dynamic Binding

Static Binding - Early Binding.
- when type of object is determined at compile time.
- each var, references, obj have type

→ int data = 30;

→
```
class Dog{
    PSVM (String[] args)
    {   Dog d1;
    }
```

→
```
class Animal{}
class Dog extends Animal
{
  PSVM
      Dog d1 = new Dog();
}
```

if there is private, final and static method ⇒ Static Binding

Dynamic Binding =

Animal a = new Horse();

Compiler doesn't know type.

Exception Handling

Powerful mechanism to handle runtime errors, so that the normal flow of the application can be maintained.

- An Exception is an abnormal condition.

- An Exception is an event that disrupts the normal flow of the program, It is an object which is thrown at runtime.

Advantages

- To maintain the normal flow of the applicatio

Types of Java Exception

- Checked
- Unchecked
- Error.

Checked Exception

- must be caught
- must be declared
- recoverable errors

The classes that directly inherit the Throwable class except RuntimeException and Error are known as Checked

eg:- IOException, SQLException etc.

Checked Exception are checked at CompileTime

Unchecked Exception

- should be caught
- could be declared
- recoverable errors.

The classes that inherit the RuntimeException.

eg: ArithmeticException, NullPointerEx", ArrayIndexOutOfBoundExceptn.

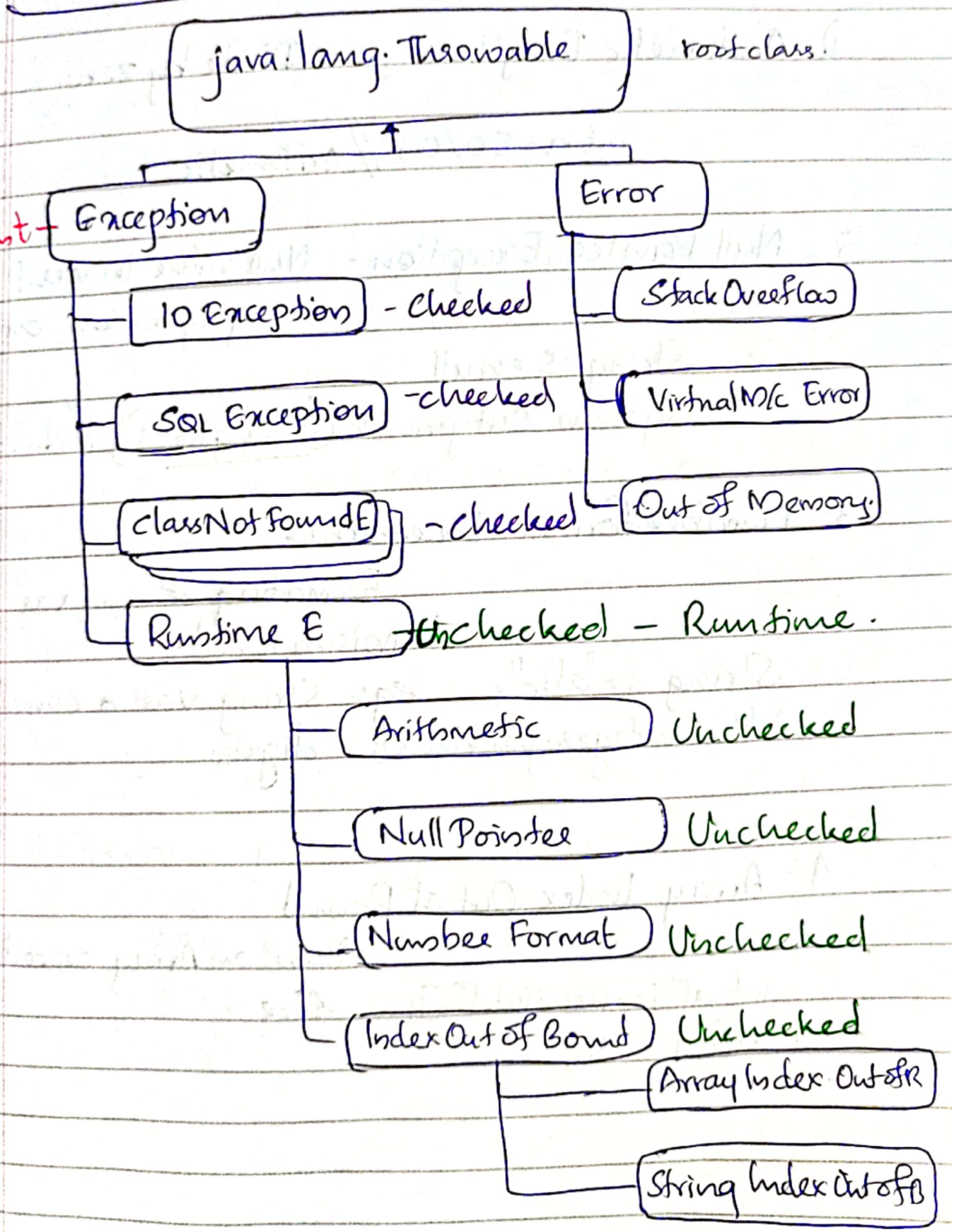
Unchecked are not at CompileTime, but they are checked RunTime.

Error

Error is irrecoverable.

eg: OutofMemory, VirtualM/c Error etc.

Hierarchy of Java Exception



- Common Scenarios - Unchecked

1) Arithmetic Exception - Divide by zero

```
int a = 50/0; // Arithmetic
```

2. Null Pointer Exception - Null value in var & performing any operation on null

```
String s = null
System.out.println(s.length()); // Null Pointer
```

3. Number format Exception -

Formating of any var or number is mismatched.

```
String s = "abc";
int i = Integer.parseInt(s);
```

eg:- String var a converted to digits.

4 Array Index Out of Bound

- When an Array exceeds to its size.

```
int a[] = new int [5];
a[10] = 50;
```

~~Try-catch block~~

Exception Vs Error

Exception - a deliberate act of omission.
Error - An action that is inaccurate or incorrect.

Exception

- It is an event
- interrupts flow of programs.
- Occcurs at runtime and compiletime.
- Occcurs in the code written by developer.
- Recovered using try-catch block & throws.
- checked & unchecked.

- When error is detected, an exception is thrown
- Any exception that is thrown must be caught by the exception handler.
- If programmer forgotten, exception will be caught by Catch-all Exception handler provided by s/m.

- Exception may rethrows if Ex handler is failed to handle it.

Advantages

1. error handling code ⟷ Seperates ⟷ regular code
   understands logical flow of pgm.
2. Continue Program execution after an Except^n is caught and handled.
3. Meaningful Error Reporting.
   - Exception thrown are objects of a class. you can obtain a description of an exception in the form of string & display it in println Stmt.
4. Propagating Errors Up the Call Stack
   - Backtrack to Caller's Automatically.
5. Grouping & differentiating error types.
6. It won't stop abruptly it terminates gracefully by giving appropriate messages.

Errors

- Problems occur due to lack of s/m resources.
- It cannot be caught or handled.
- It indicates serious problems.
- It occurs at runtime.
- Always Unchecked.

eg:-
```
recursiveDemo(int i)
while (i != 0)
{
    i = i+1;
} recursiveDemo(i);
}
```
Stack Overflow.

Differences

- Ex — Can be recovered using try-catch.
  Er — Cannot be recovered.

Ex - Type - 2 categories - Checked & Unchecked
Er - All are Unchecked.

Ex Occurance - Compiletime or Runtime
Er - at Runtime.

Ex - Package - belongs to java.lang.Exceptions
Er - java.lang.Error.

Ex. Known - Only Checked are known to Compiler
Er - Errors will not be known to Compiler.

Ex Causes - mainly by application itself
Er. - by environment in which the application is running.

Try - Catch Block

try

- Used within the method
- Enclose the code that might throw an exception.
- If an exception occured in try block, the rest of the block code will not execute.

So, not to keep the code in try block that will not throw an Ex^n.

- must followed by either Catch or finally.

```
try{
    // code that may throw an exception
  } catch(Exception_classname ref){}
```

```
try{
    // code
  } finally {}
```

Try - Catch Block

try

- Used within the method
- Enclose the code that might throw an exception.
- If an exception occured in try block, the rest of the block code will not execute.

So, not to keep the code in try block that will not throw an Ex^n.

- must followed by either Catch or finally.

```
try{
    // code that may throw an exception
} catch(Exception_classname ref){}
```

```
try{
    // code
} finally{}
```

Catch Block

- Used to handle Exceptions by declaring the type of exception with in the parameter.

- Declared exception must be Parent Class Exception or the genereated exception

- must be used after try block only.

- Single try - multiple catch block ✓

Example 1 - no try-catch.

```
public Class Test {
    P.S.V.M() {
        int data = 50/0  // exception.
        S.O.P ("rest of the code");
    }
}
```

O/p

Exception in thread "main" java.lang.ArithmeticException

rest of the code X

with try catch. checked & unchecked & name changed

```
public Class Test {
    P.S.V.M() {
    try {
        int data = 50/0;  // exception.
    }  → Array Index Bound Exception
    catch (ArithmeticException e) //handle
    {
        S.O.P(e);
    }
    S.O.P("rest of code");
    }
}
```

O/P

java.lang.ArithmeticException : / by zero
rest of the code.

3.
```
public class Test {
    P.S.V.M() {
    try {
        int data = 50/0;
        S.O.P("rest of code");
    }
    catch(ArithmeticException e)
    {
        S.O.P(e);
```

o/p

java.lang.ArithEx : / by zero.

(4)

```
Public class Test {
    P.S.V.M {
        try {
            int data = 50/0;
        }
        catch(Exception e)
        {
            S.O.P(e);
        }
        S.O.P ("rest of code");
    }
}
```

o/p.

java.lang.ArithmeticException : / by zero.
rest of the code.

(5)

```
Public Class Test {
    P.S.V.M {
        try {
            int data = 50/0;
        }
```

```
catch (Exception e)
{
    S.O.P (" Can't divided by zero");
}
}
}
```

O/p

Can't divided by zero.

⑥
```
Public. Class Test {
    P.S.V.M {
     int i, j;
      i= 50;
      j= 0;
     int data;
     try {
            data = i/j;
          }
     catch (Exception e)
     {
        S.O.P (i/(j+2));
     }
  }
}
```

O/p =

25

(7)

```
Public Class Test {
    P.S.V.M {
    try {
        int data = 50/0;
    }
    catch(Exception e)
    {
        int data = 50/0;
    }
    S.O.P("rest of data");
    }
}
```

O/p

Exception in thread "main" java.lang.A~Ex/ by zero.

so enclose code within try & use catch only to handle the exceptions.

Multi - Catch Block - for different tasks handler

- Each catch block - different exception.

-

- At a time only one exception occurs and at a time only one catch blak is executed.

- All catch blocks ordered from most specific to general.

(1)

```
Public Class Test {
    P.S.V.M {
    try {
        int a[] = new int[5];
        a[5] = 50/0;
    }
    catch (Arithmetic Exception e)
    {
        S.O.P ("A.E occurs");
    }
    Catch ( Array Index Out of Bound Ex" e)
    {
        S.O.P ("Ar I.O.B occurs");
    }
```

```
catch (Exception e)
{
    S.O.P ("Parent occurs");
}
S.O.P ("rest of code");
}
```

O/P

A.E occurs
rest of code.

(2)

```
Public class Test { P.S.V.M {
    try {
1 →     int a[] = new int[5];
2 →     a[5] = 30/0;
        S.O.P (a[10]);
    }
3 → catch (A.E e)
4 →     { S.O.P ("A.E occurs"); }
    catch (A.I.O.B.E e)
        { S.O.P ("A.I.O.B.E. occurs");}
    catch (Exn e)
        { S.O.P ("Parent");}
5 → S.O.P ("rest of code");
}
}
```

Compilation →

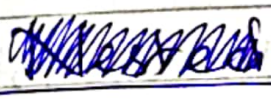
O/P

A.E occurs
rest of code.

eg:- Null Pointer Exception but not given catch
then go to catch (Exception e)
{ S.O.P ("Parent Ex");

eg: If Order of catch changed
ie, I Exception e - Parent
then A.E e.

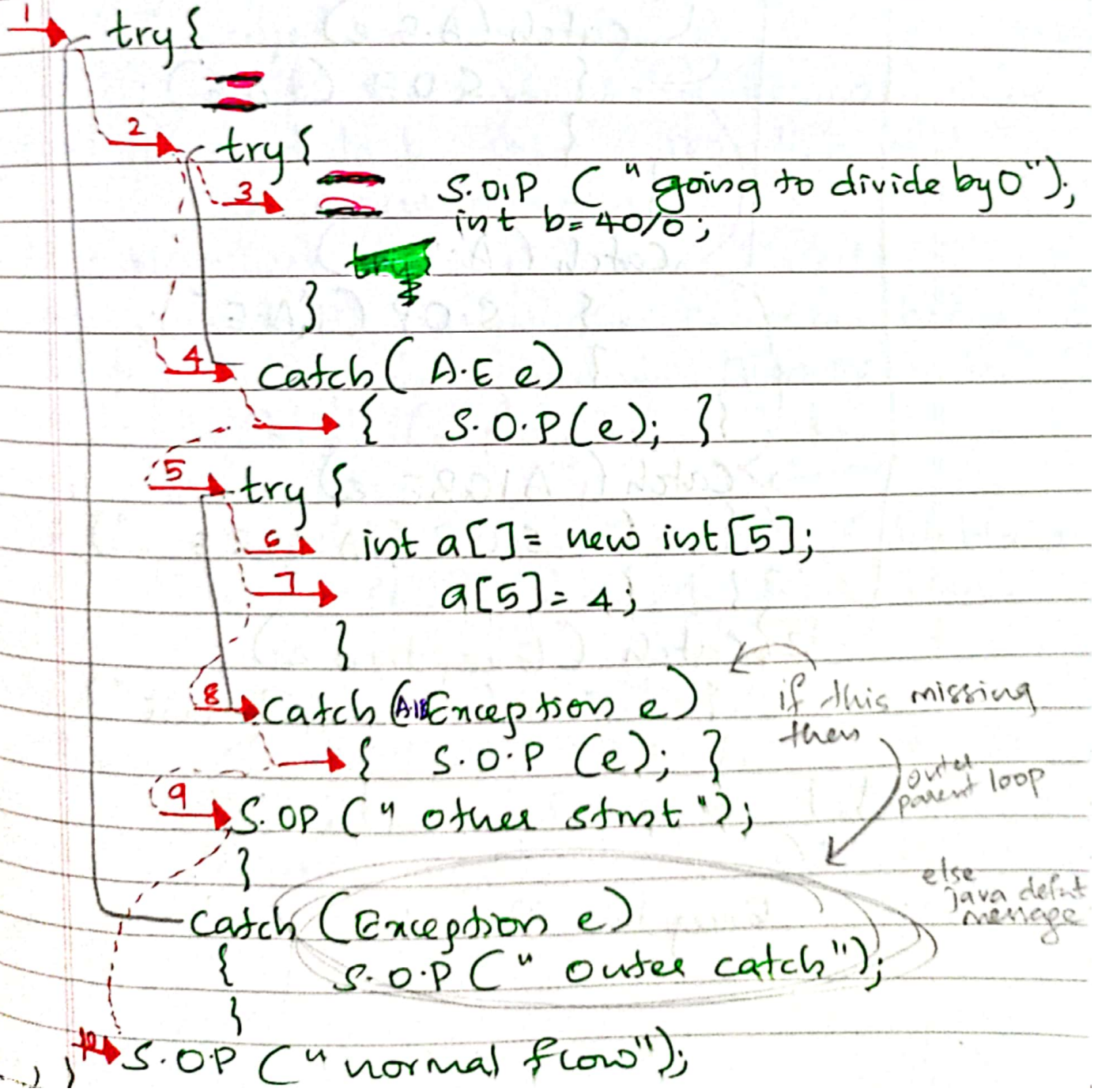
⟹ Compile time error.



Nested

Why use ?

- Situation - A part of block may cause one error & the entire block itself may cause another error.



(2)

```
try{
    try{
        try{
            int a[] = {1,2,3,4};
            S.O.P(a[10]);
        }
        catch(A.E e)
        {   S.O.P("A.E");
        }
    }
    catch(A.E e)
    {   S.O.P("A.E");
    }
}
catch(A10BE e)
{   S.O.P("A10BE e");
}
catch(Exception e)
{   S.O.P("Exception Parent");
}
}
```

A.E

O/P

Exception Parent

Finally

- Used to execute important code such as closing connection.

- Always executed whether an exception is handled or not.
  ∴ It contains all the necessary stmts that need to be printed regardless of the exception occurs or not.

- If you don't handle the exceptions before terminating the programs, JVM executes finally block (if any)

- for each try block — 0 or more Catch blocks
  — only 1 finally block.

- Finally will not executed if the program exits (System.exit())
  or by causing a fatal error that causes the process to abort.

Why use?

finally {
- cleanup code
  eg: closing a file;
  closing a connection;
}

- finally {
  important statements.
  }

— When an Exception doesnot occur?
finally executed after try block.

- Exception occured but not handled by Catch
finally executes after try block and then pgm terminates abnormally.

- Exception occured and is handled by catch
finally executed after try-catch block. Then rest of code.

Throw

- used to throw an exception from a method or any block of code.

- Can throw either checked or unchecked.

- throw keyword - throw custom exceptions.
  throw new exceptionclass("msg")
  throw new IOException("Sorry device error");

- The flow of exec" stops when 'throw' — then checked nearest enclosing try blk to see if it has a catch matching the exception. If match find control is transfer to the start o'wise next try. If no matching then default exception handler.

→ We can also throw user defined exceptions.

- If we throw checked or unchecked exceptions from a method, it must handled the exception (using try & catch) or declares in throws clause.

Unchecked // Runtime & Error

```
Public class Test {
    public static void validate(int age)
    {
      if(age<18)
      {
        throw new A.E("Person not elligible");
      }
      else
        { S.O.P ("Person elligible");
        }
    }
    p.s.v.m
    {
      validate (13);
      S.O.P ("rest of code");
    }
  }
```

O/p

Exception in th . . . . . ! Person is not . . .

Checked

- Should in try & catch
- throws keyword with method definition

```
public class Test{
    P.S. void method() throws FileNotFound
                              Exception(){
      file reader = new filereader(C:\.....)
      BufferedReader file = new BufferedReade
    throw new file not found Exception();
    }

P.S.VM
{ try {
      method();
      }
    catch (FileNotfoundException e)
      {
          e.printStackTrace();
      }
    S.O.P ("rest");
}
```

User Defined

① First create class

name ex parent class

```
class UserDefinedException extends Exception
{
    public UserDefinedException(String str)
    {
        super(str);
    }
}
```

② Then Use this above class for Exceptions

```
Public Class Test
{   p.s.vm
    {
    try
    {   throw new UserDefinedException
                ("This is user defined");
    }
    catch (UserDefinedException e)
    {
        S.O.P("Caught the exceptions");
        S.O.P(e.getMessage());
    }
}
```

o/p

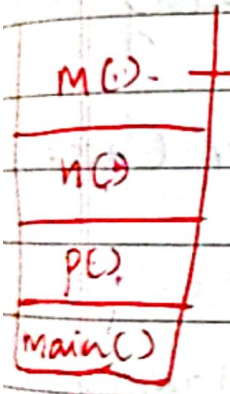
Caught the exception
This is user defined.

Exception Propagation

Unchecked

```
class Test {
    void m()
    {   int data = 50/0;
    }
    void n()
    {   m();
    }
    void p();
    try{
        n();
    }
    catch (Exception e)
        { S.O.P. ("exception handled");
    } }
P.S.V.M
    Test obj = new Test();
    obj.p();
    S.O.P ("rest of code");
```

Checked Ex are not forwarded. error.



Callstack.

o/p.

exception handled

Throws

- Declare an Exception
- Tells programmar that may occur Except^{n}. So programmar bettee give handling code.
- Unchecked Exception = Programmer's fault of not checking code.

- Exception handling mainly for Checked.

method — return type methodname()
throws Except^{n} class name{
}

Which Exception should be declared?
- Checked ✓

Unchecked - Our Control, we can correct Code
Error - Beyond our Control. Symptoms.

Adv

- checked Exception can propagate or forwarded in Call Stack.
- It provides info^{n} to the Caller of Method about Exception.

```
try{
  throw new NullPointerException();
}
catch(NullPointerException e)
{
  S/O.P  rethrow
  throw e;
}
```

Rule

If we are calling a method that declares an exception, we must either caught (1) or declare (2) the exception.

Case 1 - Caught - try-catch
2 - Declare - Specified throws keyword with the method.

Declare Exception.
- (1) Exception doesnot occur then fine
- (2) Exception occurs then it is thrown at runtime because throws doesnot handle Exception.

eg:-
```
class M {
    void method() throws IOException{   // already.
        throw new IOException("device error");
    }
}
class Test
    P.S.V.M throws IOException{   // declared now.
    M m = new M();
    m.method();
}
```

THROW Vs THROWS

Throw – keyword used to throw exceptions inside fn / block of code

Throws – used in method signature.

Throw – Only Unchecked can propagate

Throws – Only checked can propagate
But can declare both.

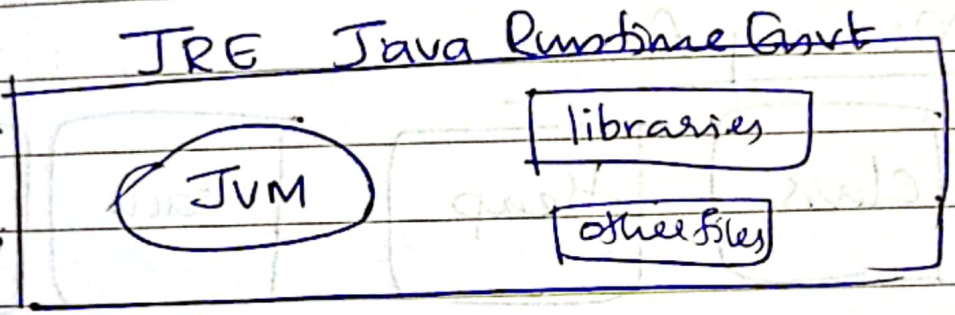
Syntax.

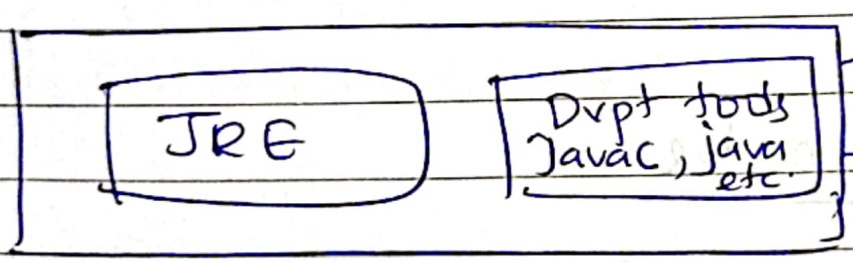
Declaration Throw – Inside method
Throws – Method Signature

Internal Implementation
Throw – Only one at a time
Throws – can declare Multiple Exceptions using throws.

JDK, JRE, JVM = platforms dependent bcoz configuration of OS differs.

JVM - abstract m/c
provides runtime envt in which bytecode executed.

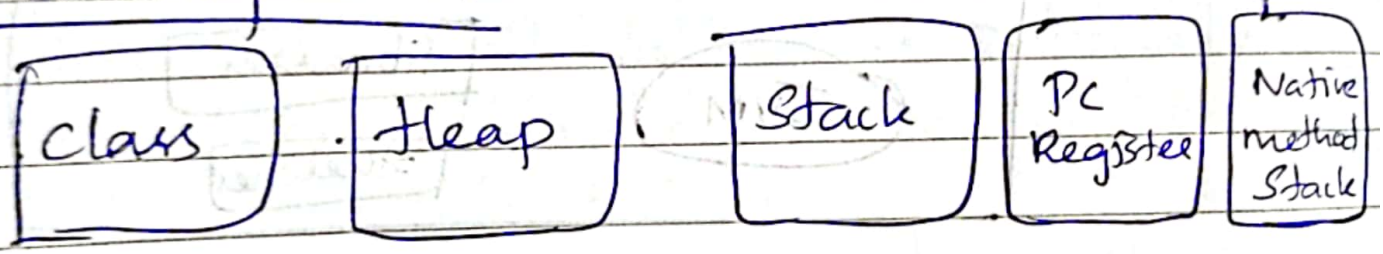




JVM - platform dependent.

- loades code
- Verifies "
- Executes "
- Provides runtime envt

Memory Areas



Class Loader

Subsm of JVM which is used to load class files.

Whenever we run the java programs, it is loaded first by the classloader.

Class Area

- It stores per-class Structure such as runtime constant pool, field and method data, the code for method.

Heap Runtime data area in which objects are allocated.

Multithreading

- Process of executing multiple threads simultaneously.

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

Multithreads

- threads use a shared memory area.
- Context switching b/w thread takes less time than process.

- doesn't block the user because threads are independent
- perform many operations together, so it saves time.
- Threads are independent, so it doesn't affect other threads.

Multitasking

Executing multiple tasks simultaneously

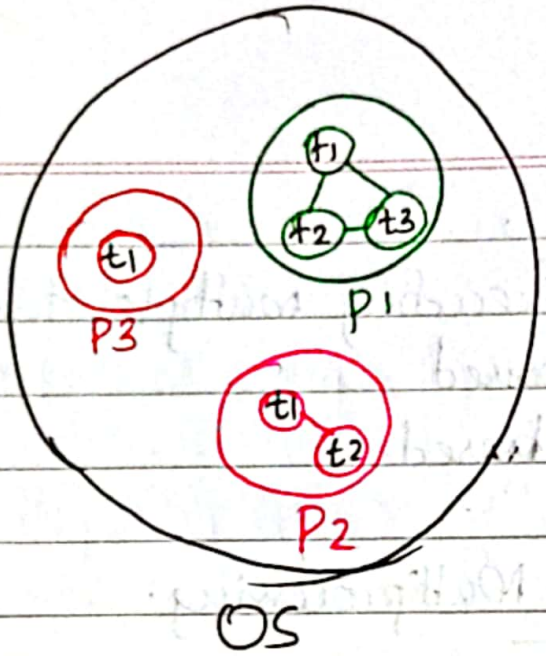
- Process based
- Thread based.

Process based - Multiprocessing.

- Each process — address in memory.
  ie, each has seperate memory area.
- A process is heavy weight.
- Cost of comm" b/w process is high.
- Switching takes time // saving & loading registers.

Thread based Multitasking

- Threads share same address space.
- It is lightweight.
- Cost of comm" b/w the thread is low.
- Each thread - atleast one process.
- At a time one thread is executed only.



Multithreading = ability to execute multiple different parts of the code.

Normally Java use one thread
But we can break off to multiple threads

Creation

2ways

→ extends Thread class
→ implements Runnable.

eg:

```
public class MultiThreadingThing extends Thread
{
    @Override
    public void run()
    {
        for(int i=1; i<=5; i++)
        {
            S.O.P(i);
            try{
                Thread.sleep(1000);
            } catch(InterruptedException e){ }
        }
    }
```

sleeps 1 sec in each iteration prints.

```
public class MultiThreading {
    P.S.V.M (String[] args)
    {
        MultiThreadThing obj = new MultiThreadThing();
        obj.start();
```

(or do it in seperate thread)

obj.run() will work but won't give you new thread, So we use obj.start();

Java will branch off new thread and start running this one method.

O/P
1
2
3
4

Then we create second thread

```
MultiThreadThing obj2 = new MultiThreadThing();
obj2.start();
```

	start()	run()
o/p	1	1
	1	2
	2	3
	2	4
	3	5
	3	1
	4	2
	4	3
	5	4
	5	5

```
for (int i=0; i<5; i++)
{
  MultiThreadThing obj=new MultiThreadThing();
    obj.start();
}
```

o/p

1

1

1
1
1
2
2
:
5
5

```
public class MultiThreadThing extends Threads
{
    int ThreadNum;
    public MultiThreadThing (int ThreadNum)
    {
        this.ThreadNum = ThreadNum;
    }
    @ override.
    void run()
    {
        for(int i=1 ; i<=5 ; i++)
        {
            S.O.P(i, "from thread "+threadNum);
        try {
            Thread.sleep(1000)
        } catch(InterruptedException e)
            { }
        }
```

```
public class MultiThread
{   P.S.V.M()
    {
      for(int i=0; i<=3; i++)
      {
        MultithreadThing obj = new Mn();
        obj.start();
      }
    }
```

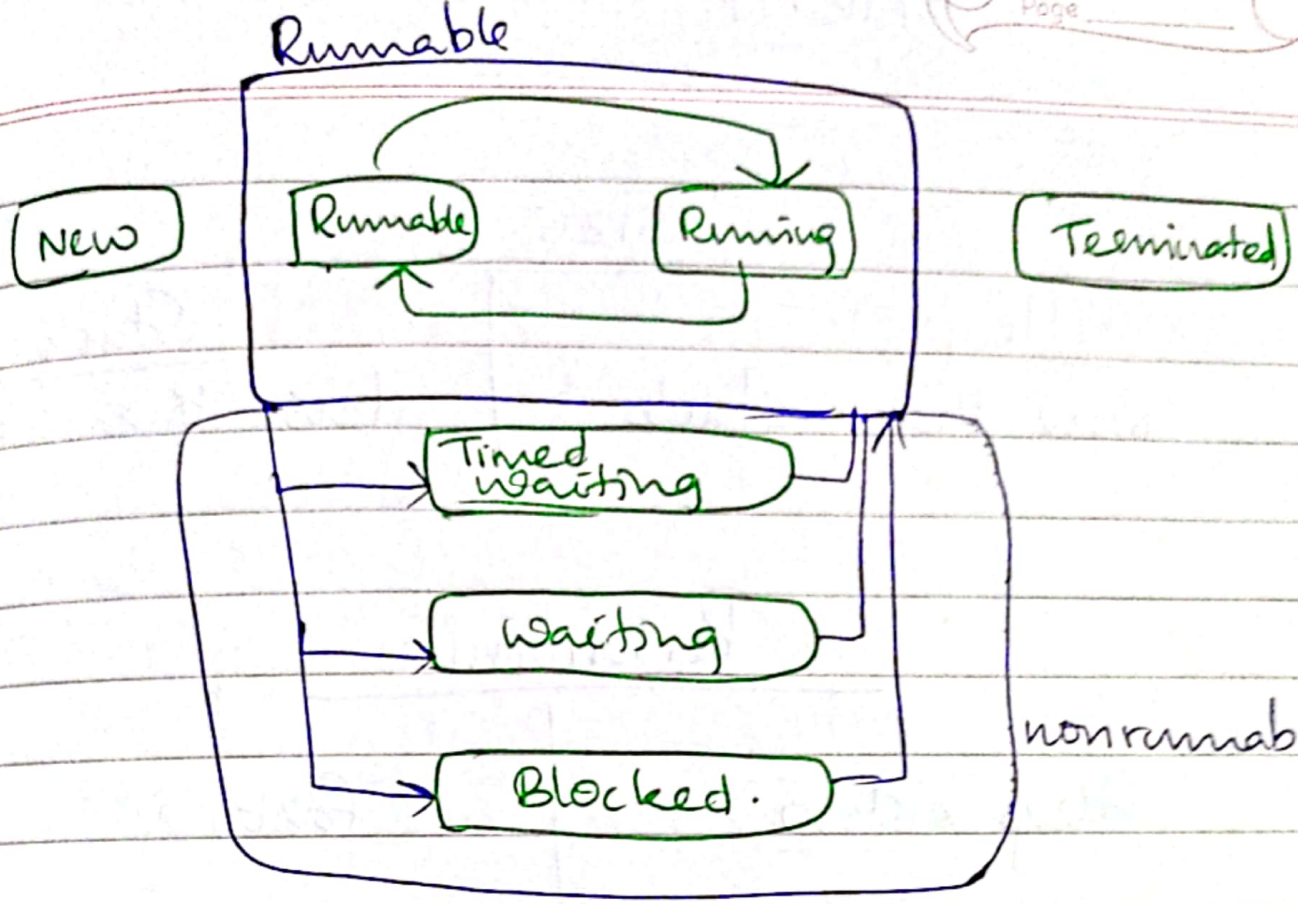
O/p

```
1 from thread 1
1 from thread 3
1 from thread 2
1 from thread 0.
2 from thread 3
2 from thread 1
```

// If you do implements Runable.

```
main()
{
  MultithreadThing obj = new MultiThreadThing();
  Thread myThread = new Thread(obj);
  myThread.start();
}
```

Extra in Runable



Blocked

waiting for a monitor lock locked & trying to access section of code, by some other thread.

Waiting

" - for some other thread to perform a particular action.

HEAP Vs STACK

Size	
Heap	**Stack**
more than stack	less than heap

Performance	
Heap is slow	fast

Life	
More Life from start to end of application	less life / short life.

Accessible	
objects Accessible globally	objects stored in stack cannot access across thread.

Application	
each part of an appln used at the time of exec^n	used in parts means one at a time on exe^n of thread.

Specific

mainly appl^{n} specific	mainly thread specific

Efficient

No efficient use of space	Used Efficiently.
Object stored in Heap	Its reference stored in Stack.
Random Access	Particular Order Access.
More Complex don't know Memory is occupied or not	Simple & Easy.

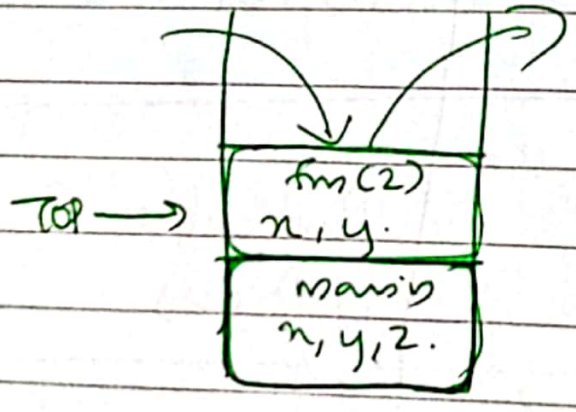
Stack

- Exception handling & Execut^{n} of thread.

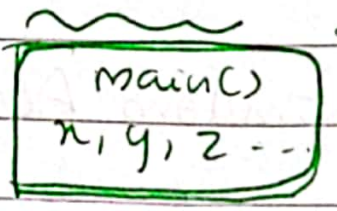
Values in Stack – exist within the scope of method or fns they created.

If that fns executed or returns value then var are removed.

- Only stores local var primitives.



each thread



Garbage.

Garbage

- Unused Memory ✗

Unreferenced

- nulling the ref.
- assigning a ref to another.
- Anonymous obj. etc.

① e=null

② e1 = e2 → obj. unref.

③ new Employee();

finalize.

GC only collects those objs of new keyword
If you created any obj without new
you can use finalize method to perform
clean up processing

It is invoked each time bf the obj is
garbage collected.

Daemon thread

```
public class Test{
    public void finalize()
    {
        S.O.P ("obj is g.c");
    }
    public s. v. m
    {
        Test s1 = new Test();
        Test s2 = new Test();
        s1 = null
        s2 = null.
        System.gc();
    }
}
```

① ② ③

o/p

obj is gc
obj is gc.