Java OOPs – object oriented programing

we can take any number of classes in Java

Java source file structure

```
class A
{
}

class B
{
}

class C
{
}

class D
{
}
```

0 or 1 public class is allowed, more than one public class leads to error

if there is no public class we can save as any name but if there is public class we must save with that specific class name

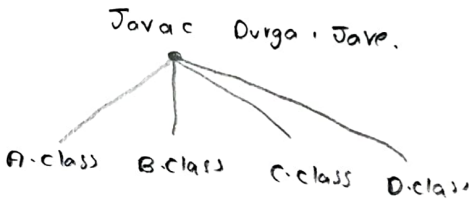
if ther is public class B then we must save file as B.java only

```
class A
{
    P S V main (String[] args)
    {
        S.o.p ("A class")
    }
}

class B
{
    P S V main (String[] args)
    {
        S.o.p ("B class")
    }
}

class C
{
    P.S.V main (String[] args)
    {
        S.o.p ("C class")
    }
}

Class D
{
}
```

Saved with Durga.java

we can take any name because there is no public class

for every class there is one dot class file generated



if we run A class
↳ A class main exigut

if we run B class
↳ B class main exigut.

if we run D class
↳ error
Because there is no main class

import java.util.Arraylist — explicit import

import java.util.* — Implicit import

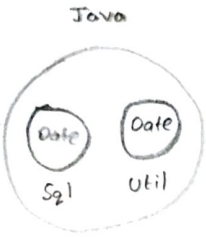
any this will work

redisability is important so we need to explicit import.

Java.lang - by default it can access without writing import

Default Pkg

Package



there are two types of dates in Java

Java.util.Date

Java.sql.Date

both are differentiated with package

① Name conflits resolved

② modularity of application

③ maintainability of application

④ seguvity. improved

a set of classes and interfaces grouped together are knowns as Package.

we need to use silent domain name because it is unique.

```
Package com.durgasoft.ocja;

Public class Test
 {
    P S V main (String [] args)
     {
        S.o.pln ("Package");
     }
 }
```

Javac Test.Java

cwd

Saved in current working directo

So no use so

Javac -d . Test.Java

cwd
- com
  - durgasoft
    - ocja
      - Test.cla

Note :- only 1 or 0 Package is allowed in Java program.

order ↓

Package Statement	— 0 or 1 allowed
import Statement	- any number
Class / interface / enum	- any number

Class level modifiers

Public class — it can access this class anywhere.

default Class — only with in the package

Public
<default>
abstract
final
strictfp — strict floating point

if class is abstract then object creation is not allowed

if the class is final then child class creation is not allowed

(APDFS)

Top level class
```
Class Test
{
```
——— Public, <default>, abstract, final, strictfp

Inner class
```
    Class Inner
    {
    }
}
```
——— Private, protected Static

(PSP)

Ex

```
Package Pack1;

Public class A
{
}
```

Public bool
Public int
Public transport

```
Package Pack2;

import Pack1.A;

Public Class B
{
    P.S.V main (String[] args)
    {
        A a = new A();
    }
}
```

class A is accesed

abstract modifier → abstract class
→ abstract methode

abstract class cannot have body

* abstraction is the process of hiding certain details and show only essential info to user

abstract methode - method which has only declaration but not implementation. it ends with semicolon.

- child classes are responsable for implementation

```
Public class Vehical
{
    Public int getNoof wheels ();   // end with semicolon
}
```

* only be used in an abstract class, and it does not have a body the body is provided by the subclass (inherited from)

Example Real time example

```
class loan
{
    Public double getIntrestrate ();
}
```

- we cannot say how much intrest genevated at instant we it depends on many parameter if it is personal loan, houseloan or gold lone.

abstract class

class that is not Completely implemented

abstract class is a restricted class that Cannot be used to create objects

(to access it, it must be inherited from another class)

```
Abstract class Test
{
    ≡
}
```

```
Test t = new test();
t.m()
```

Ex:-

abstract class

```
abstract class Animal {

    Public abstract void animal Sound();  — abstract method

    Public void sleep() {

        System.out.println("zzz");
    }
}
```

} sub class

Even though a class does not contain any abstract methods we can declar as abstract class.

```
abstract class Test {

    Public void m1() {}

    Public void m2() {}

    Public void m3() {}

}
```

like adapter class

Java adapter class provide the default implementation of listener interfaces. if you inherit the adapter class, you will not be forced to provide the implementation of all the methods of listenor interfaces. so it saves code

Java.awt.event class ✓

if the class contain any abstract method then it is declared as abstract class.

```
abstract class Test      (Parent class)
{
    Public abstract void m1();
    Public abstract void m2();
}
```

} 2 ab methods

```
(child)
Public class Subtest extends test
{
    Public void m1() {

    }
}
```

only one is implemented

X wrong error will generated

" Child class is responsable ~~for~~ to Provide implementaion ~~to~~ for evevy abstract method but Here only m_1 is implemented So it generate error "

Solution :- create m_2 implementation or make child class as abstract class, we can now implement m_1 only. the m_2 is handled by Sub sub child

```
Class SubSubTest extends Test
```

```
abstract class Vehicle
    {
        Public abstract int getNoofwheels();
    }

Class Bus extends Vehicle
    {
        Public int getNoOf wheels()
        {
            return 6;
        }
    }

Class Auto extends vehicle.
    {
        Public int getNoof wheels()
            {
                return 3;
            }
    }

Class Test
{
    Public static void main (String[] args)
    {
        Bus b= new Bus();
        S.o.pln ( b.getNoOf wheels()); → (b.getNoof wheels());

        Auto A = new Auto();
        S.o.pln ( a.getNo of wheels());
    }
}
```

- objects can be created to this classs because it is Not abstract.

member modifiers

Public – global

Corresponding class must also Public (visible)

if void m1() is not public we cannot access in Class B

```
Package Pack1;
class,
Class A  — but class is not public
{
    Public void m1()  — method
    {                  (member is public)
        S.o.pln("A class")
    }
}
```

A.Java – Saved name

```
Package Pack2
import Pack1.A;
Public class B
{
    p.s.v. main (String [] args)
    {
        A a = new A();
        a.m();
    }
}
```

X

if class A is Public then it will exigute

javac -d . A.java

even though member in class is public the class is not public so we cant access in class B

Private – only in class. <default> – only in package

```
class A
{
    Private void m1() — only visible in this class only
    {
        System.out.println(" A class method");
    }
}
Public class Test
{
    Public static void main (String[] args)
    {
        A a = new A();  — Connot access this
        a.m1();
    }
}
```

Protected

With in the current Package we can access anywhere but with in the OutSide Package we only access in child classes.

Protected = <default> + child

```
Package Pack1:

class A
{
    Protected Void m1()
    {
        System.out.println(" A class protected method");
    }
}

Class B extends A
{
    Public Static Void main (String[] args)
    {
        A a = new A();   - Parent reference , Parent object
        a.m1();

        B b = new B();   - child reference , child object
        b.m1();

        A a1 = new B();  - parent reference, child object
        a1.m1();
    }
}
```

```
Package Pack2
import Pack1.A;        // child of Pack1
Public class B extends A
{
    P.S.V main (String [] args)
    {
        A a = new A();          ×
        a.m();
        B b = new B()           ✓
        b.m();
        A a1 = new B();
        a1.m1();                ×
    }
}
```

only use reference B because it is protected

Visibility	Public	protected	default	Private
with in same class	✓	✓	✓	✓
From child class of Same package.	✓	✓	✓	×
From Non-child class of same Package.	✓	✓	✓	×
From child class of outside package.	✓	✓ we should use child ref only	×	×
From Non-child class of outside package	✓	×	×	×

Private < default < protected < public

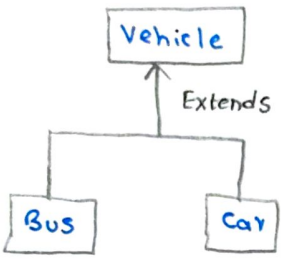
interface

Any Service Requirement specification (SRS)

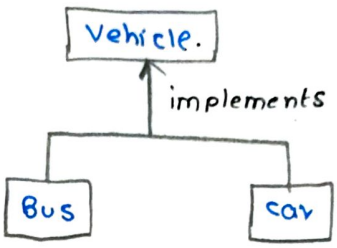
Any contract between client & service provider

100% Pure abstract class in old versons but not now

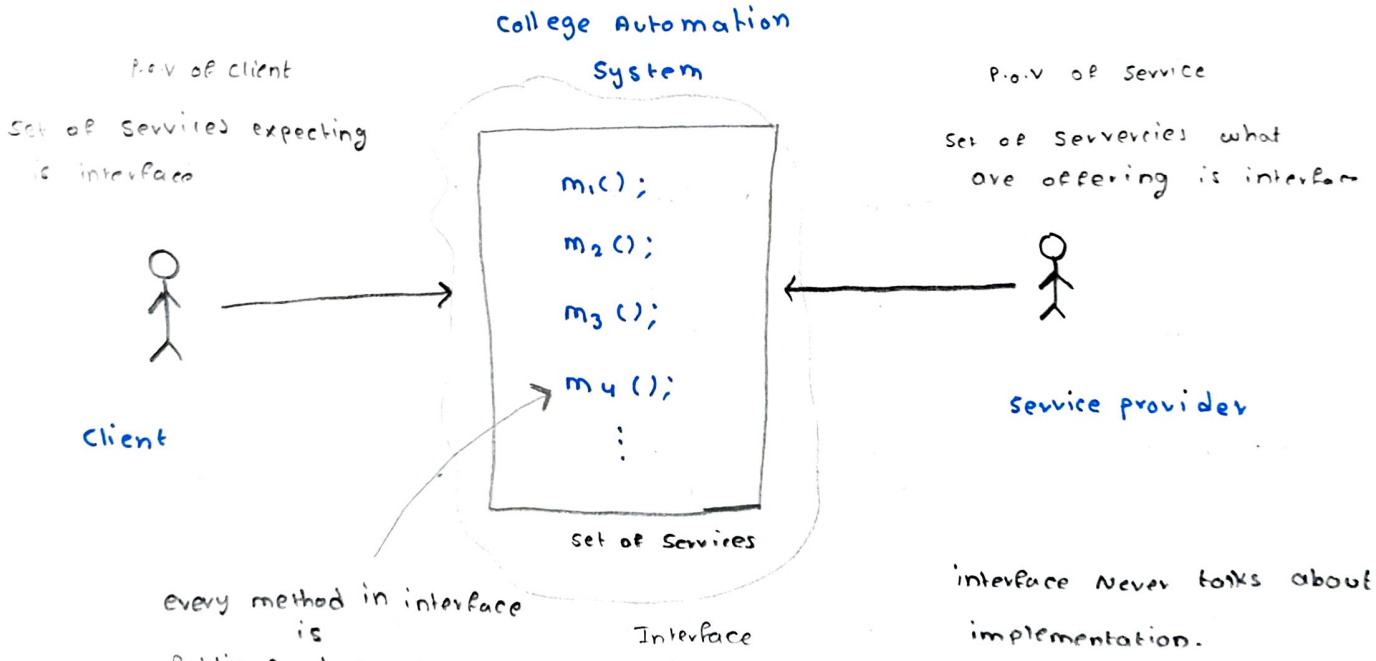
abstract class



interface



Ex



But Not 100% pure abstract class

Because Default, Static & private method are allow from 1.8 & 1.9 Version

implementation of interface

```
interface Interf
{
    Public Void m1();
    Public Void m2();
}
```

method should not have a return value

By default it is public & abstract even if we didn't mentioned Public

```
class Service provider implements Interf
{
    Void m1()
    {
    }
}
```

wrong

```
class Service provider implements Interf
{
    Public Void m1()
    {
    }
    Public Void m2().
    {
    }
}
```

must mention Public for there are two methods so we should provide two implemental

if we only want m1 implementation

```
abstract class service provider implements interf
{
    Public void m1()
    {
    }
}
```

```
class Sub servrie provider extends servic provider
{
    Public Void m2()
    {
    }
}
```

This subchid is vesponsable for implement interface which is not provided in above class.

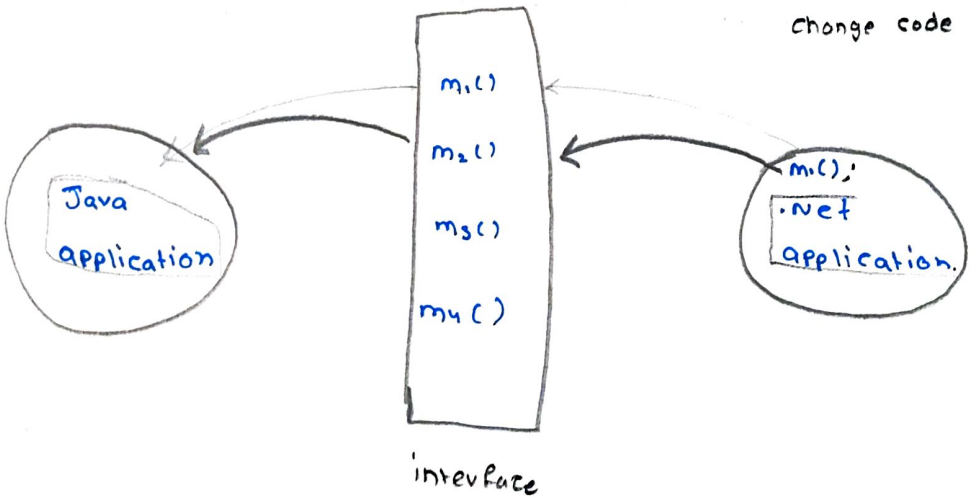
OOPS

Data hiding

```
Class  Bankaccount
{
                                    By declaring data member as private we can implement
    Private  double  balance;       Data security

    Public  double  get Balance ()
    {
        // validation.  ──→ is it true user or not, by providing
                            account creditionals or otp
        if  valid
            return  balance;
    }
}
```

Abstraction

Hiding interal implementation Just Highlet the set of Services we are offering.



without effecting end user we can change code

Adv

1. Security
2. Enhancement
3. maintainibility
4. modularity

.net has to call m_1 internally implemented by Java application and give the result. But internal implementaion is unknown to .net

Abstraction example — real time Bank operation.

Encapsulation

Encapsulation = Data hiding + Abstraction

The process of grouping data members and corrosponding methods or behaviours is called Encapsulation.

Capsule

all chemical ingredients are capsulated in one capsule

```
Class Student
{
    String name;
    int age;
    int rollno;
    int marks;
    +
    read() { }
    write() { }
    :
}
```

Hiding Data behind methods

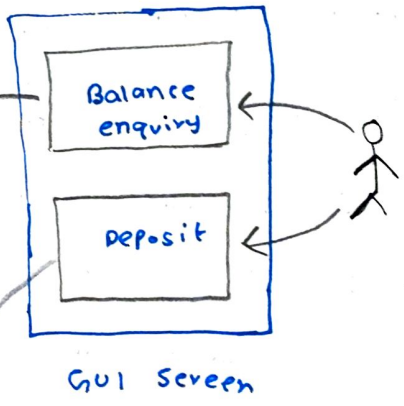
we are getting security the we need to sacrify something that is Time consuming process and Performance is low

Because for any transfer we need to validate so it consume time

Adv

① Security
② Enhancement
③ maintaiability
④ modularity.

```
Class Account
{
    Private double balance;   (Data hiding ✓)

    Public double getBalance()
    {
        // validation
        return balance
    }

    Public void setBalance(double amount)
    {
        // validation
        this.balance = this.balance + amount;
    }
}
```



Gui Screen

internal code is not visible to user he can only see Buttons so it is abstraction.

Data hiding + Abstraction = ✓

Tightly encapsulated class

When every data member is declared as private then it is Tightly encapsulated

100 % Security

```
Class A   Tightly encapsulated Class
{
  Private int x=10;
}

class B extends A   Not
                    Tightly encapsulated
{
   int y = 20;
}

class C extends A
{
   Private int z = 30;
}
```

Tightly encapsulated

```
Class A
{
   int x=10;          Not Tiltly
}

class B extends A
{                                  Not Tilly
     private int y = 20;
}

class C extends B                  NOT Tilly.
{
     Privat.
     ~~Privat~~ int z = 30;
}
```

If parent class is not tightly encapsulated even when we declar the child class as private then it will not make it tightly encapsulated.

```
C c = new P();  ✗
```

```
P p = new C();
p.m1()  ✓
p.m2()  ✗
```

Inheritance

1. IS-A Relationship
2. Code reusability
3. Using extends keyword

```
class P
{
   Public void m1()
   {
      S.o.pln ("Parent");
   }
}
class C extends P
{
   Public void m2()
   {
      S.o.pln ("child");
   }
}
```

class P has one method

class C has Two methods Both m1 & m2 as it is inherit.

we can call parent specific method using child reference but we cannot call child specific method using parent reference.

P p = new P();	C c = new C();
p.m1(); ✓	c.m1(); ✓
p.m2(); ✗	c.m2(); ✓

importance of inheritance

without inheritance.

```
class Hloan
{
    300 methods
}
```

```
class Ploan
{
    300 methods
}
```

```
class Vloan
{
    300 methods
}
```

900 methods

90 hr ~

with inheritance

let there are 250 methods are same for all classes

```
class loan
{
    250 methods
}
```

```
class Hloan extends loan
{
    50 methods
}
```

```
class Ploan extends loan
{
    50 methods
}
```

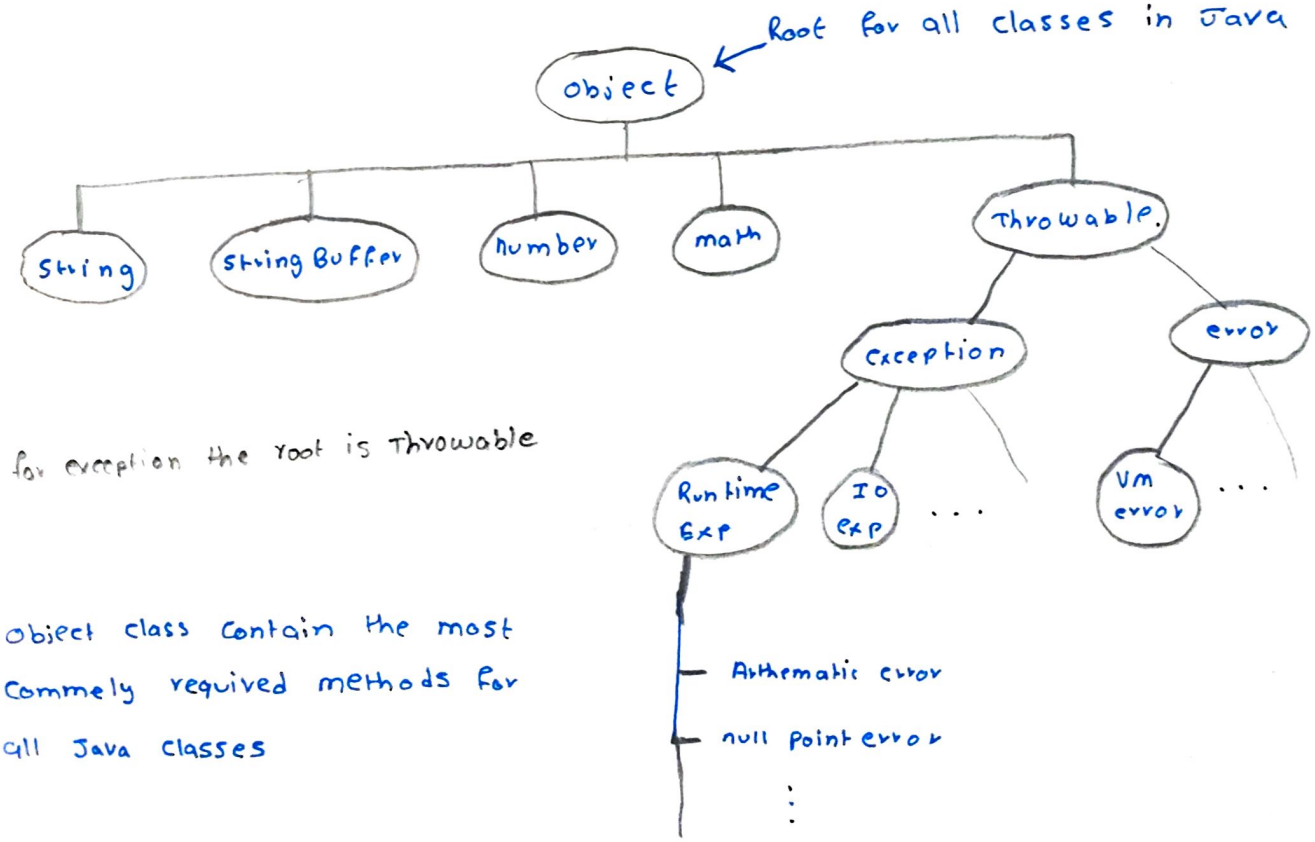
```
class vloan extends loan
{
    50 methods
}
```

400 methods

40 hr ~

improved code reusability

less development time

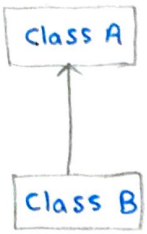


for exception the root is Throwable

object class contain the most commely required methods for all Java classes

Types of inheritance

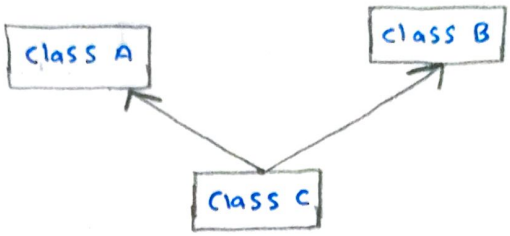
① Single inheritance




```
Class B extends A
```

② multiple inheritance

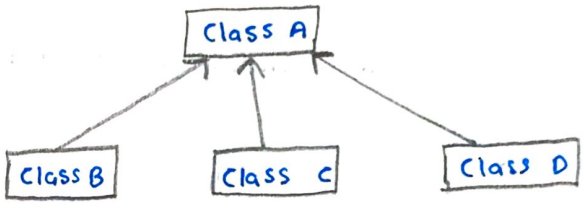
(Not supported in Java)



③ multi-level inheritance

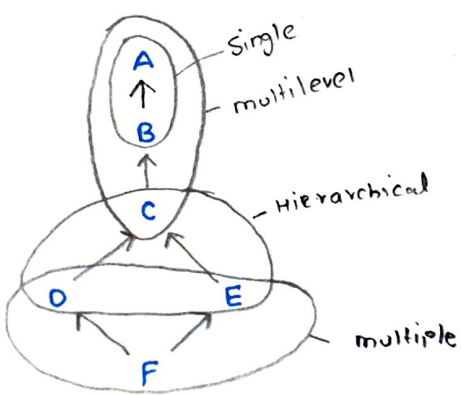

```
Class A
 {
 }
Class B extends A
 {
 }
Class C extends B
 {
 }
```

④ Hievarchical inheritance




```
Class A
 {
 }
Class B extends A
 {
 }
Class C extends A
 {
 }
Class D extends A
 {
 }
```

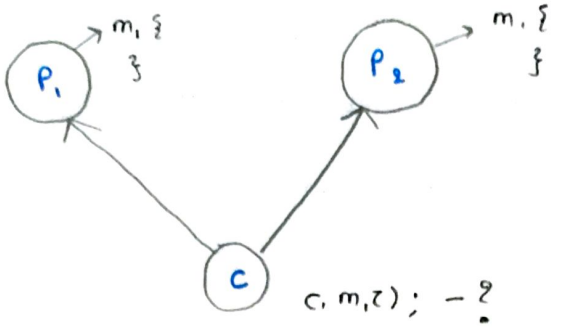
⑤ Hybrid Inheritance



(Not Supported in Java)

Because it is not possible to create multiple inheritance

Why Java wont Support multiple Inheritance ?



Class

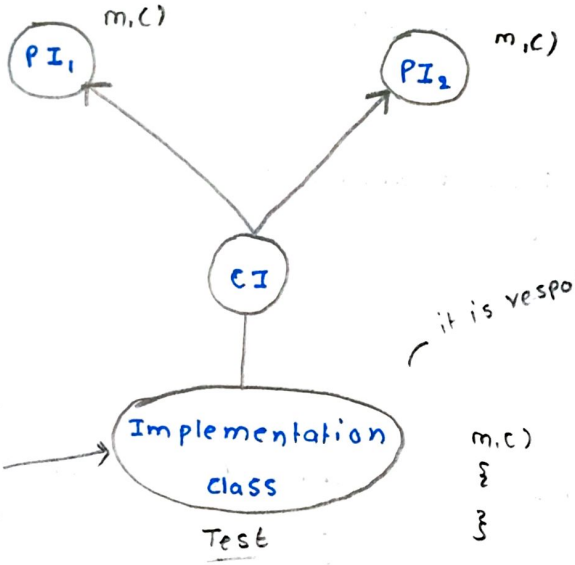
it has common methods in both Parent classes then there will be proble when we called the method from child class.

* Diamond Access problem
* Ambiguity problem

class c extends P1, P2 { }

→ by default Java not support multiple Inheritance but using interface it is possible.

But we can use multiple inheritance in interface.



PI1 P1 = new Test();

P1.m1()

```
class P1
{
}

class P2
{
}

class C extends P1,P2
{
}
```

```
Interface P1
{
}

interface P2
{
}

interface C extends P1,P2
{
}
```

* Because in classes multiple method implementations are there but in interface only declaration are available but not implementations

∴ Java provide Support for multiple Inheritance in interface but not in classes

Python Support multiple inheritance.

```
Class P1 :
    def m1(self):
        Print ("P1 method")

Class P2 :
    def m1(self):
        Print ("P2.method")

Class C (P1, P2):
    c = C()
    c.m1()
```

First one will exigutad

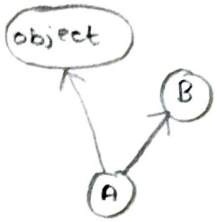
So P1 class m1 method is called

if P1 not have m1, then it will check in P2

interview Quetion

Q) we all know that every class in Java is child of object. then.

```
Class A extends B
{
}
```



A) what actually happening is.

-if our class doesent extends any other Class then only our class is divect child class of object class

```
Class A {
}
```


```
class A extends B
{
}
```

multilevel

This is how it look lik

Cyclic inheritance

Not support in Java.

Not in Python

```
Class A extends A
{
}
```


```
Class A extends B
{
}

Class B extends A
{
}
```

Better to take single Class instead of this

method signature

(return type — only int will return)

```
Public int m1 ( int i, float f )
```

↓

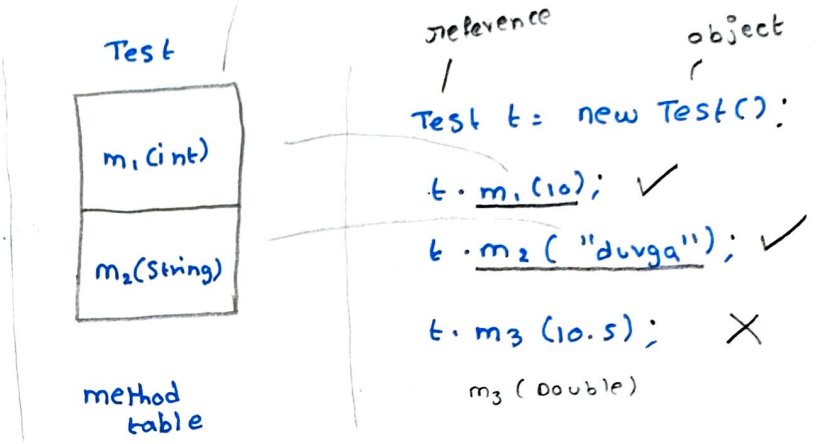
Signature: `m1 ( int, float )`

- `m1` — method name
- `int, float` — argument

Compiler will use method Signature while resolving method calls

For every class the compiler will main one table is called method table

```
class Test
{
    Public void m1 (int i)
    {
    }

    Public void m2 (String s)
    {
    }
}
```



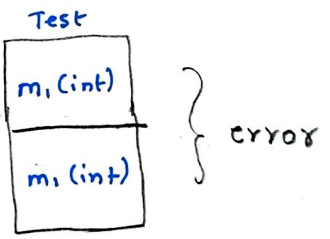
Two methods with same signature is not allowed so it produce CE

```
class Test
{
    Public void m1 (int i)
    {
    }

    Public int m1 (int j)
    {
        return 10;
    }
}
```

m1(int)

m1(int)

```
Test t = new Test();
t.m1(10);
```



NOT in C

Overloading (Same name but different arguments)

Not allowed this in C

m1(int)
m1(float)

in c overloading is not allowed so different types of abs functions are there for different types of data types.

abs (int)
labs (long)
fabs (float)

more flexibility

overloaded methods

```
Class Test
{
    Public void m1()
    {
        s.o.pln ("no-arg");
    }
    Public void m1(int i)
    {
        s.o.pln ("int-arg");
    }
    Public void m1(double d)
    {
        s.o.pln ("double-arg");
    }
}
```

```
P.S.V main (String [] args)
{
    Test t = new Test();
    t.m1(); ✓   No arg
    t.m1(10); ✓   int arg
    t.m1(10.5); ✓   double d
}
```

which method is to exigute in overloading method resolution take caves by compiler based on reference type

so it is called as
- Compile time polymorphism
- Static polymorphism
- Early binding

* * * * *

t.m1('a'); - let program has only int & String

Here a is char. first compiler checks for argument char as we know there is no char. the the compiler will no directly produce CE error. it will automaticaly promot char to the int and then it will check for int argument. So it will give o/p :- int-arg

Byte → Short
Char
int → long → float → double

(automatically promotion) in overloading - promotion is done simultaneously

if it is double the CE occurs → No Suitable method

C-2

Exact match of argument will get the chance

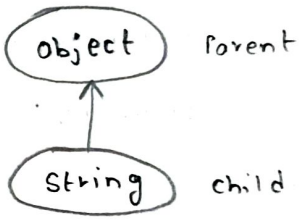
```
Class Test
{
    Public void m1(object o)
    {
        S.o.pln("object version");
    }

    Public void m1(String s)
    {
        S.o.pln("String version");
    }

    P.S.V main(String[] args)
    {
        Test t = new Test();
        t.m1(new object());   object version
        t.m1("durga");        String version
        t.m1(null);           String version.
    }
```

(story)

* let we need a signature in document by mayor. our step is to reach the assistant of mayor he will make our doc signed by mayor. It is easy to reach assistant rather than going to mayor.

↓

* null is valid for any reference
  we can provide in String
  we can provide in object



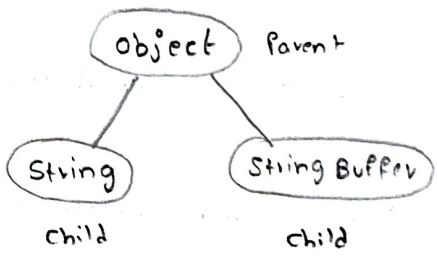
C-3

Both methods got matched but no relation blw arguments.

t.m1(null); CE ✗

```
Class Test
{
    Public void m1(String s)
    {
        S.o.pln("String version");
    }
    Public void m1(StringBuffer sb)
    {
        S.o.ph("String Buffer Version");
    }
    P.s.V main(String[] args)
    {
        Test t = new Test();
        t.m1("durga");                      String Version
        t.m1(new StringBuffer("durga"));    String Buffer Version
```

overloaded methods



C-4

```
Class Test
{
    Public void m1(int i)
    {
        S.o.pln("General method");
    }
    Public void m1(int... i)
    {
        S.o.pln("Var-arg method");
    }
    P S V main (String[] args)
    {
        Test t = new Test();
        t.m1();      Var-arg method
        t.m1(10,20); Var-arg method
        t.m1(10);    General method
    }
}
```

overloaded

1.0 Version ← m1(int i) → m1(10);

m1(int... i) → m1(); m1(10); m1(10,20); m1(1,2,...);

1.5 Version

t.m1(10) - Both matched arguments.

m1(int i); — came in V1.0

m1(int... i); - 1.5

old is gold so this is exiguted

Note :- Var-arge will always get least priority in Java like default in Switch case

C-5

```
Class Test
{
    Public void m1(int i, float f)
    {
        S.o.pln("int-float version");
    }
    Public void m1(float f, int i)
    {
        S.o.pln("float-int version");
    }
    P.S.V main(String[] args)
    {
        Test t = new Test();
        t.m1(10, 10.5f); int-float
        t.m1(1.5f, 10);  float int
        t.m1(10,10)      CE
    }
}
```

There are two ways

t.m1(int, int)

t.m1(int, float) ✓ ✗ t.m2(float, int) ✗ ✓

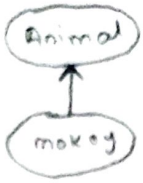
t.m1(int, int) → Promot → t.m1(int, float) ✓

t.m1(int, int) → Promot → t.m1(float, int)

Both are same so compiler time error

Same for Both float Values.

C-6




```
class Animal
{
}
class monkey extends Animal
{
}

class Test
{
    Public void m1(Animal a)
    {
        S.o.pln ('Animal ref');
    }

    Public void m1 ( monkey m)
    {
        S.o.pln ( 'monkey ref');
    }

    P.S.V main ( String[]args)
    {
        Test t = new test ();

        Animal a = new Animal();

        t.m1(a);      Animal ref

        monkey m = new monkey();

        t.m1(m);      monky ref

        Reference        object
        Animal a1 = new monkey();

        t.m1(a1);     Animal ref  ---->  Depends on type Not object
    }
```

creating an object of monkey class and assigning it to a variable of type Animal

method overriding

that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of the superclass or parent classes.

The method signature (name, return type and parameters) must be the same in both the subclass and superclass.

- Runtime polymorphism
- Dynamic polymorphism
- late binding

```
Class P
{
    Public void property()
    {
        S.o.pln (" cash + Gold + land");
    }
    Public void marry()
    {
        S.o.pln (" Appalamma | Subbalakshmi");
    }
}
```

↑ overridden method

```
class C extends P
{
    Public void marry()
    {
        S.o.pln ("Katrina");
    }
}
```

↑ overriding method

```
class Test
{
    P.S.V main ( String [] args)
    {
        P p = new P();          - Parent method
        p.marry()

        C c = new C();          - child method
        c.marry();

        P P1 = new C();         - Child method
        P1.marry();
    }
}
```

```
P P1 = new C();
```

Compiler: check if the method is available or not then give green signal if it exists.

Jvm: give chance only for overriding method

* in overloading method resolution take care by Compiler based on reference type.
* in overriding method resolution take care by Jvm based on runtype object

Method overriding Rules

① Return type must be Same – until 1.4V Come

From 1.5 co-varient return types are allowed

```
class P
{
        Object
   Public Void m1(int i)
    {
    }
}

Clasc c Extends classP
{
        String ✓
    Public Void m1 (int i)
    {
    }
}
```

Here String is child of object

Co-varient return types ↓

if parent class method is object class return type then in child class the child class of object class are allowed

Co-varient concept only applicable for objects not for primive

only object types Not Primitive ↓

Parent method Return type → object

Child method Return type → object, String, StringBuffer...

Number → Number, Integer, Float

String → Object ✗

Double → Int ✗

In object type co-varient object types are allowed in child

In primitive type ~~Co-varient~~ the child must also primitive type method.

② No relation b/w m1 in class P & m1 in class C

```
Class P
{
   Private void m1(){
   }          Private method for P
}

class c  Extends P
{
   Private void m1(){
   }          Private method for C
}
```

P p = new C();

p.m1(); ⟶ error because Private method Not Called out side class

[Not overriding] Valid but Not overriding

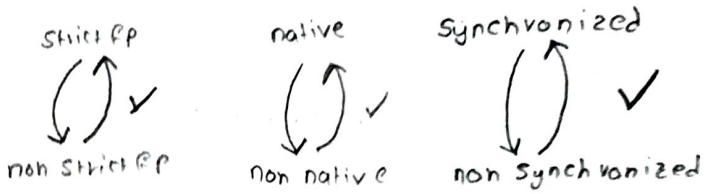
③

```
Class P
{
    Public final void m()
    {
    }
}

Class C extends P
{
    Public void m1()
    {
    }
}
```

→ means it is the final implementation no one is allowed to overide

CE :- Overidden method is final

even when child class is also final Compile time error occur when try to overide.



Parent method	final	non-final	abstract	non-abstract
child method	non-final final ✗	final ✓	non-abstract ✓	abstract ✓ (This child will Provide implementation)

Conclusion:- except static the only one not possible is overiding final method.

④

Decreasing scope

Class P (Public > Protected) ✗

```
{
    Public void m1()
    {
    }
}

class C extends P
{
    Protected void m1()
    {
    }
}
```

Protected → Public ✓

(Protected < Public) ✓ increasing scope

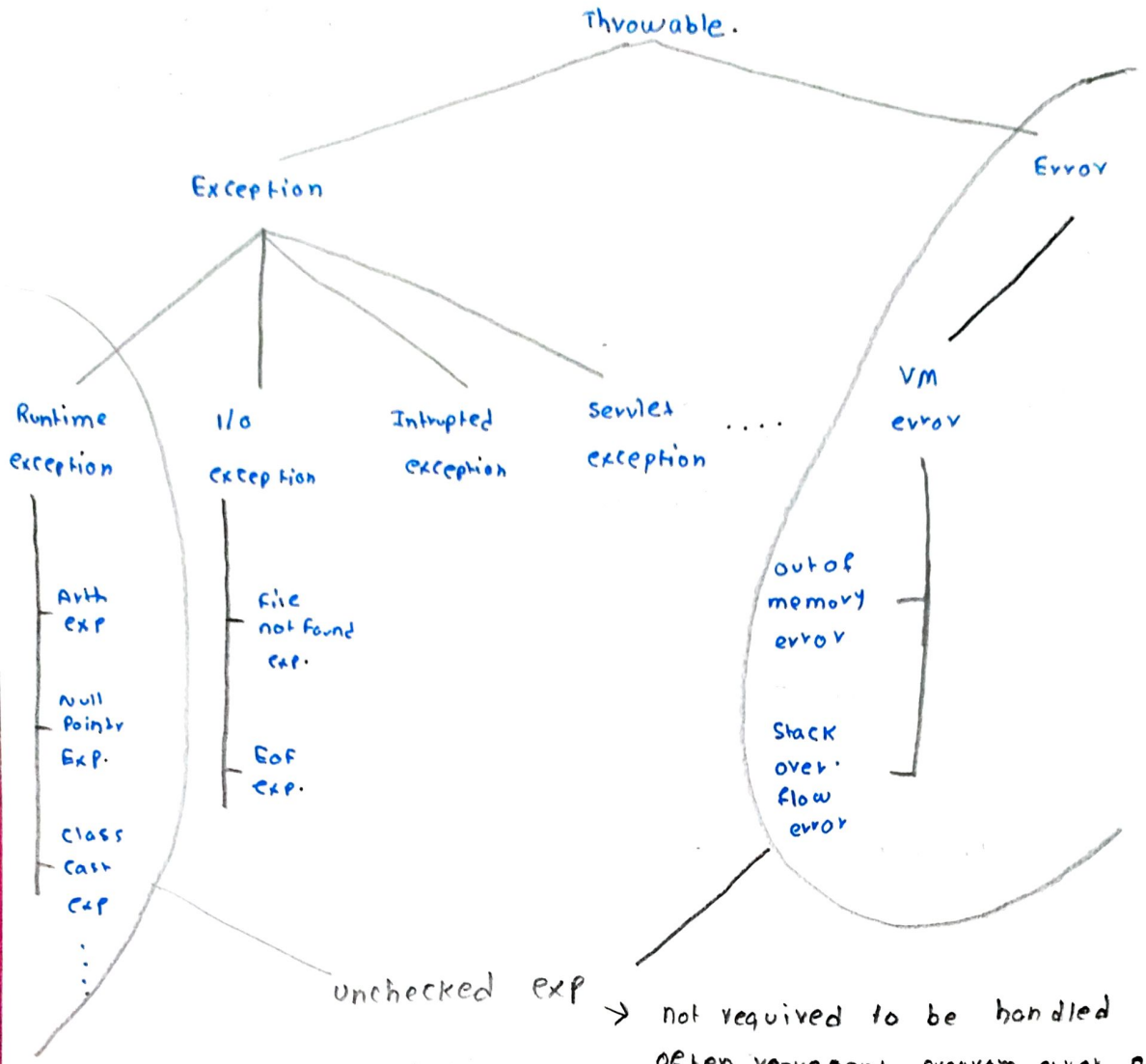
No possible to override

Private < default < protected < Public

Scope

Parent method	Public	Protected	default
Child method	Public	Protected Public.	default Protected Public.

Public → Protected — Reducing scope of access so Not valed.



Developer must check

↓

checked exception must be either caught using try-catch blocks or declared in the method signature

unchecked exp → not required to be handled explicitly and often represent program error or unexpected runtime conditions

Runtime exception and its child classes & error and its child classes are unchecked exceptions. Remaining are checked exceptions.

Rule :-

If child class method throws any checked exception complesory the parent class method should through the same checked exception or its parent.

```
class P
{
    public void m1() throws Exception
    {
    }
}

class C extends P
{
    public void m1()
    {
    }
}
```

✓

```
class P
{
    public void m1()
    {
    }
}

class C extends P
{
    public void m1() throws Exception
    {
    }
}
```

✗ error

① P : Throws Exp ✓
C :

② P :
C : Throws Exp ✗

③ P : ~~Ex~~ Throws Exp ✓
C : Io Exp

④ P : Io Ex ✓
C : EoF , FNFE

⑤ P : Io Ex ✗
C : EoF , IE

⑥ P : Io ✓
C : EoF , NPE

Checked allowed

Static - used to declar members that belong to the class rather than instance of the class

method Hiding

overriding w.r.t Static methods

```
class P                    class level method
{
  Public Static Void m1()
  {
  }
}

class C extends P          No overriding
{
  Public void m1()
  {
  }
}                          object level method
```

✗

```
class P                    object level method
{
  Public void m1()
  {
  }
}

class C extends P          Class level method
{
  Public Static void m1()
  {
  }
}
```

✗ No overriding

```
Class P
{
  Public Static void m1()
  {
  }
}

class C extends P
{
  Public static void m1()
  {
  }
}
```

No error ; Seems like overiding is done

- it is not overiding
  it is method hiding.

The Static Keyword in Java is mainly used for memory management.

The static keyword in Java is used to share the same variable or method of a given class.

Ex:

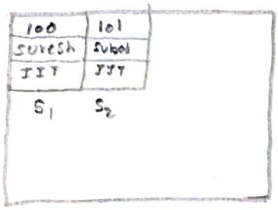
Here we are creating a class for a colg 'IIT Hyd'

Here for every student 'IIT Hyd' is same.

```
class Student {
    int rollno;
    String name;
    String Colg; "IIT Hyd"
}
```

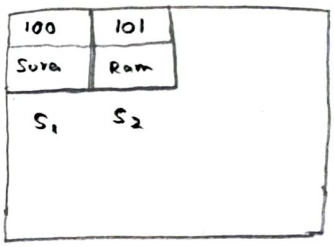
```
Public class main {
    Public Static void main (String [] args) {
        Student S1 = new Student ('100', "Subash", "IIT");
        Student S2 = new Student ('101', "Suvesh", "IIT");
```

→ it says To exigute this method No objects are required it will self exigute with the help of class



Heap

now By assagining colg as static. like.



Heap

Static String colg = "IIT Hyd"

↓

Student S3 = new Student ('102', "Ramesh"); → enough

Reduce space

```
Class P
{
    Public Static void m1()
    {
        S.o.pln ('Parent');
    }
}
class C extends P
{
    Public Static void m1()
    {
        S.o.pln ('child');
    }
```

```
Class Test
{
    P.S.V main (St)
    {
        P p = New P();
        P.m1();

        C c = new C();
        C.m1();

        P P1 = new C();
        P1.m1()
    }
}
```

If it is overriding we will get o/p like this

But it get

Parent	Parent
child	child
Child	Parent

method Hiding	overriding
① Both Parent class and child class methods should be Static	① Both Parent class and child class methods Should be non-Static
② method resolution always takes care by compiler based on reference type	② method resolution always takes care by Jvm based on runtime object
③ method hiding is ~~run-tim~~ Compile time polymorphism (or) Static Polymorphism (or) early binding	③ overriding is Runtime polymorphism (or) Dynamic polymorphism (or) late binding

method hiding occurs because static methods are resolve at compile time.

overriding w.r.t Var-arg

```
Class P                          Var.arg
{                                  |
    Public void m1 (int... i)
    {
        S.o.pln ("Parent");
    }
}

Class C extends P
{                                Normal
                                   |
    Public void m1 (int i)
    {
        S.o.pln ("child");
    }
}
```

it is overloading

```
Class Test
{
    P.S.V main (String [] args)
    {
        P.P = new P()
        P.m1();     - Parent

        C.C = new C();
        C.m1();     - Child

        P.P1 = new C();
        P1.m1(10);  - ~~Child~~ Parent
    }
}
```

we can make it overriding by making both methods Var-arg (Child & Parent)

Variable hiding or shadowing

overriding w.r.t variables

```
class P
{
  String s = "Parrent";
}

class C extends P
{
  String s = "child";
}

class Test
{
   P.S.V main (String[] args)
   {
      P p = new P();
      S.O.pln ( P.s);

      C c = new C();
      S.O.pln ( C.s);

      P P1 = new C();
      S.O.pln (P1.s);
   }
}
```

P→instance C→instance	P→ Static C→ non-static	P-non Static C- Static	P- Static C- Static
P	P	P	P
C	C	C	C
P	P	P	P

From 1.5 covarient Retur type allowed

Property.	overloading	overriding
method names	must be same	must be same.
Argument Types	must be diff	must be Same
Private / Final / Static methods	Can be overloaded	cannot be overriden
Return types	No restriction	must be same (1.4)
Throws clause	No restriction	if child method through any checked exptss compulsary parent class method should throwsin the same CE or Parent
method resolution	Compiler based on refevence type	Jvm base on runtime object
other names.	Compiletime polymorphism Static Polymorphism Early binding	Run time polymorphism Dynamic Poly morphism Late binding.

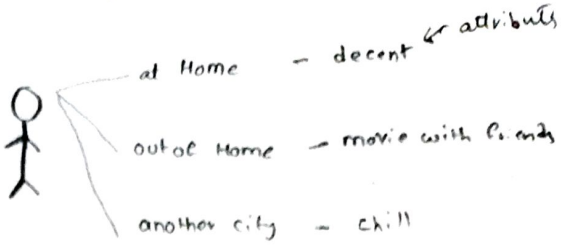
Polymorphism

Poly means many

morphs means forms

ability of a Variable, Function or object to take on multiple form.

Poly technic - many technolgies



attributes

abs (int)

abs (float)

abs (long)

overloading

Same method name but different argument types

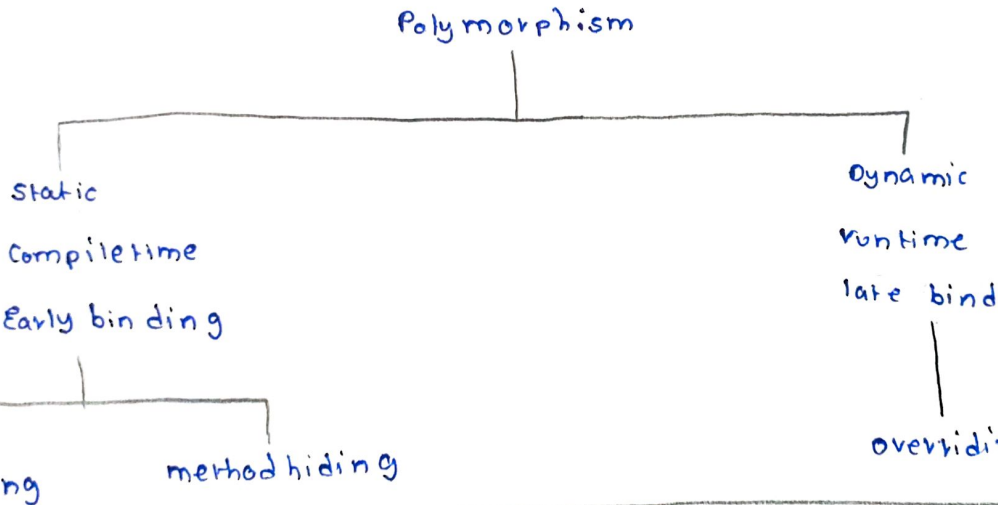
```
class P
{
    marry ()
    {
     S.o.Ph ("Subbu");
    }
}

class C extends P
{
     marry ()
     {
      S.o.pln ("Katrin");
     }
}
```

overriding

overloading & overriding are the best example of polymorphism.

Joke :-

a boy start love with word friendship. but girl end love with the same word Friend ship. word is same but attitude is different



3-pillers of oops

abstraction, data hiding

① encapsulation — Security

② Inheritance. — Reusability

③ polymorphism — Flexibility

object typecasting

```
object o = new String ("durga");
StringBuffer sb = (StringBuffer) o ;
```

is it valide ?

```
A B = (C) d ;
```

we are converting d type object to C and we are assing to A type refference variable B

Object o = new String ('durga')

StringBuffer Sb = (StringBuffer) o ; → A B = (C) D ;

(StringBuffer) o : object type; changing

Compile time checking - 1

Check relation between C & D either Parent to child or child to parent or Same type.

Rule-1 Passed

lets continue later

---

constructor — used to allocate memory

Methods are called after creating object

But constructor are called during the creation of object

```
Public class C {
    P.S.V main (String [] args) {
        C c1 = new C();
    }
}
```

This is constructor

default constructor (always Public)

we can have number of constructors

what happen inside, Java will creat a default constructor

```
Public C() {
}
```

→ look like method but different

① method name ≠ class name

② method must have return type but it not there

Rule

* Constructor name must be class name
* No return type provided
* By default Java provide Public to default constructor (implicit constructor)

explicit constructor

```
Public C() {
}
```

Non-Parameteried const

```
Public C(int a) {
}
```

Parameterised constructor

we can add both parameterised & Non Parameterised Constructor explicitly.

or we add constructor explicitly Java don't creat by default

* Default constructor is not visible if we want to see open dot file after compilation then there will be visible.

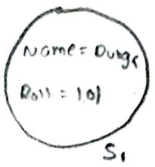
See back →

Constructors

is specialy created to initialization of object

```
Student S1 = new Student ("Duga", 101);
```

↑ Using new object is created



By new object is created

By new default values are assigned

access specifier allowed are

Public, <default>, Protected, Private

```
class Student
{
    String name;
    int rollno;
    Student ( String name , int rollno)
    {
        this . name = name;   ← means current object
        this . rollno = rollno;
    }

    P.S.V main ( String[ ] args)
    {
        Student S1 = new Student ( "Durga", 101);
        S.O.ph ( S1 . name + " " + S1 . rollno);
    }
}
```

Programer code

```
class Test
{

}
```

```
Public class Test
{

}
```

compiler code

```
class Test
{
    Test ()
    {
        super();
    }
}
```

```
Public class Test
{
    Public Test ()
    {
        super();
    }
}
```

super
this

we can use only inside constructors

we should use only in first line.

we can use only one but not simultaneously.


```
class P
{
    P()
    {
        super();
    }
}
```

No args

```
class C extends P
{
    C()
    {
        super();
    }
}
```

No args

```
class P
{
    P()
    {
        super();
    }
}
```

No args

No args

```
class C extends P
{
    super();
}
```

```
class P
{
    P(int i)
    {
    }
}
```

CE error

```
class C extends P
{
    C()
    {
        super();
    }
}
```

No args

Exception Handling In Java

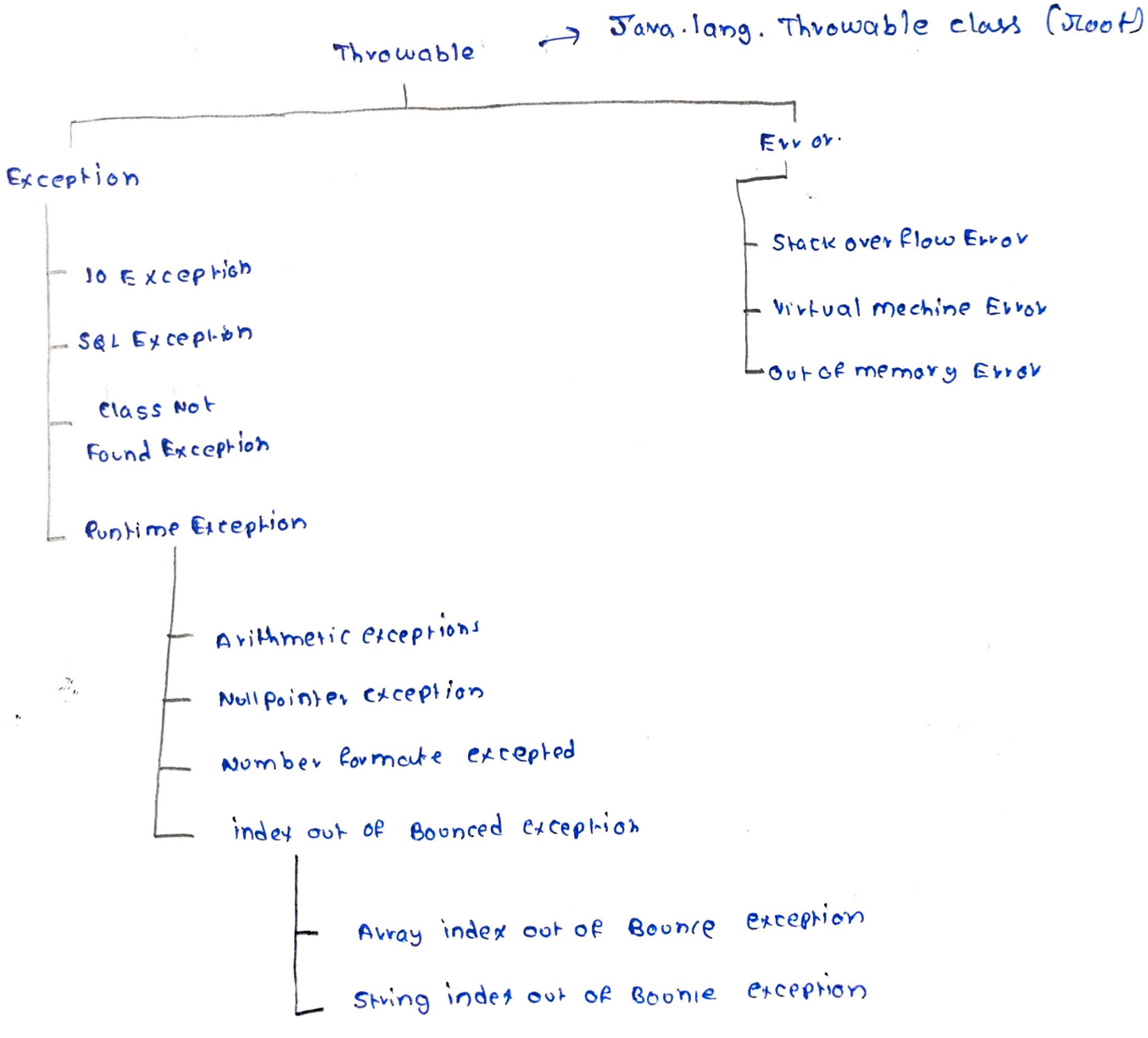
Powerful mechanism to handle the runtime errors so the normal flow of applicatio. can be maintained.

mechanism to handle runtime errors such as Class Not Found Exception, IOException, SQL Exception, Remot Exception etc.

Cove advantage of exception handling is to maintain the normal flow of application.

```
Statement 1;
Statement 2;
Statement 3; // exception occur
Statement 4;
Statement 5;
```

let there are 5 statements in a Java program and an exception occur at 3 the rest of the code will not performed, However when we perform exception handling the rest of the statements will be executed.



Exception Handling

- Checked exception - checked at compile time (Developer must check) — IO, SQL
- Unchecked exception - checked at run time. - no required to be handled explicitly — arith, null pointer
- Error - is irrecoverable - out of memory Error, Vm error ...

Exception Keywords

overriding possible

Try - used to specify a block where we should place an exception code. It means we can't use try block alone try block must be followed by either catch or finally

Catch - catch block used to handle the exception, it must be preceded by try block which means we can't use catch block alone.

Finally - used to put important codes such as clean up code eg: closing the file or closing connection.

Throw - Throw keyword is used to throw an exception

Throws - used to declare exception it specify that there may occur an exception in the method. it doesn't throw an exception it is always used with method signature.

Throw an exception

Indicate what exception type may be thrown by method

Java multiple catch Block

```
try {
} catch () {
} catch () {
}
```

Java nested try

```
try {
}
```

```
try {
    try {
        try {
        } catch () {} } catch () {} }
```

Throw	Throws
① used to throw an exception explicitly in the code, inside the function or block of code	• used in the method signature to declar an exception which might be thrown by the function.
② we can only propagate unchecked exception	• we can declar both checked and unchecked exceptions
③ used within method	• used with the method signature.
④ Throw only one exception at a time	• we can declar multiple exception.

Final Keyword	Finally Block	Finalize method
access specifier which is used to apply restrictions on a class, method or variable. final method cannot be overriden final class cannot be inherited	Block in exception handling to exigute the important code whethe the exception occurs or not	method in Java which is used to perform clean up processing just before object is garbage collected Finalize() Perform necessary cleanup befor object is destroye. Such as releasing resources or detaching event listners

Array in Java

Collection of similar type of elements which has contiguoes memory location.

Adv :- code optimization :- we retrive or sort the data efficiently.

Random access

DAdv :- Size limit :- we can only store fixed size of elements in the array. it doesent grow it size at runtime.

import java.util.Arrays;

methods

Arr.sort();

Arr.fill(); - fill with single value

Arr.toString(); - convert arry to string

Arr.copyOf(); - coppy an array

Arr.equals(); - check if two arrays are equal

Arr.binarySearch(arr, 3); - Do binary Search to find a elements.

Arrays.sort(Arr);

String, StringBuffer and StringBuilder

most commonly used object is String

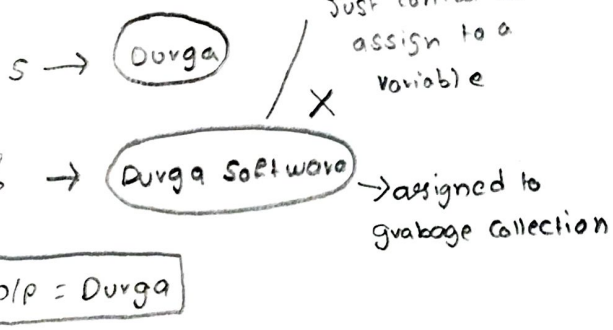
Sequence of char is String

what is difference between String & StringBuffer

* String objects are immutable (cant change)

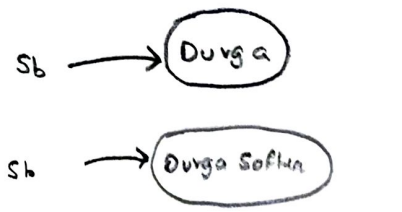
* StringBuffer objects are mutable (can change)

① String s = new String("durga");
s.concat("software");
s.o.println(s);



o/p = Durga

StringBuffer sb = new StringBuffer("durga");
sb.append("software");
s.o.println(sb);



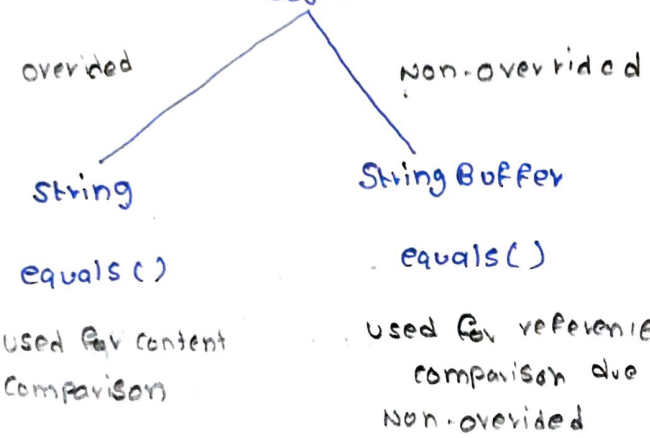
o/p = durga software

② address is compared

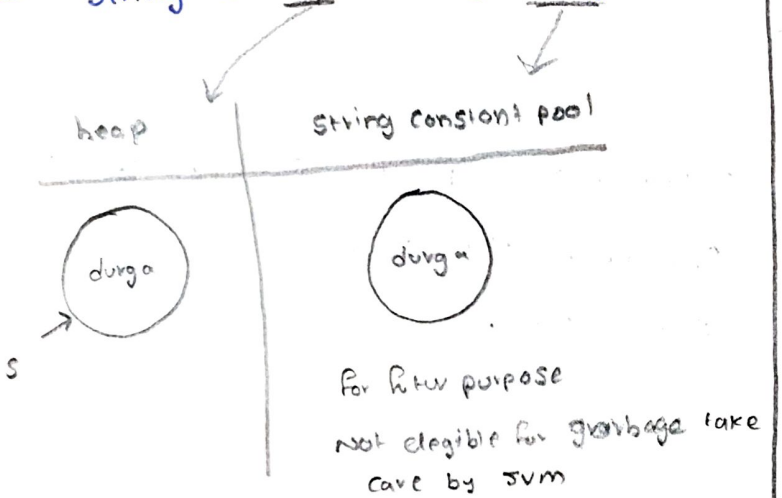
```
String s1 = new String ("durga");
String s2 = new String ("durga");
S.o.pln (s1 == s2);  false
S.o.pln (s1.equals(s2));  true
```

```
StringBuffer sb1 = new StringBuffer ("durga");
StringBuffer sb2 = new StringBuffer ("durga");
S.o.pln (sb1 == sb2);  false
S.o.pln (sb1.equals (sb2));  false
```

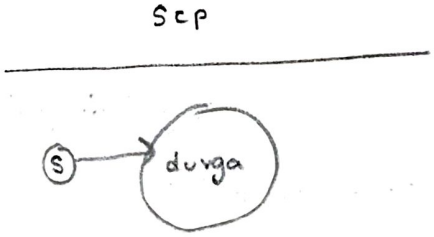
object → equals() - used for reference comparision in object class.



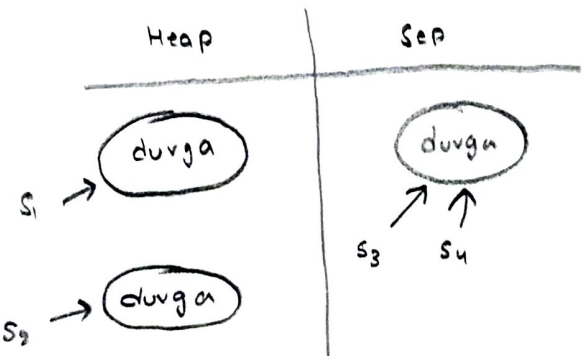
③ String s = new String ("durga");



String s = "durga";

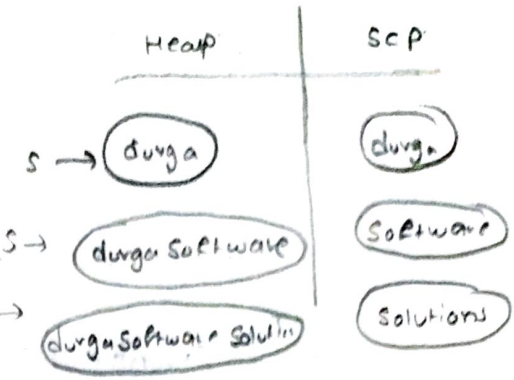



```
String s1 = new String ("durga");
String s2 = new String ("durga");
String s3 = "durga";
String s4 = "durga";
```



Same content only one is allowed

```
String s= new String ("durga");
s.concat (" Software");
s = s.concat (" Solutions");
```



special memory management

Advantage / importance of SCP

String Constant Pool

if there are 1000 objects in project >900 are Strings

improve reusability of object

Just becaus of SCP String objects are immutiable

Quetions Imp

① Why SCP concept is available only for String object but not for StringBuffer ?

String is most used object so they provided SCP memory management.

② Why String objects are immutiable where as StringBuffer objects are mutiable ?

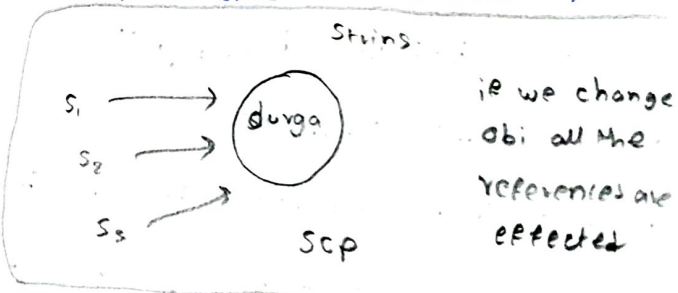
in String SCP only one object is created and they are referred by many references, if we change the object then the will be many references effected

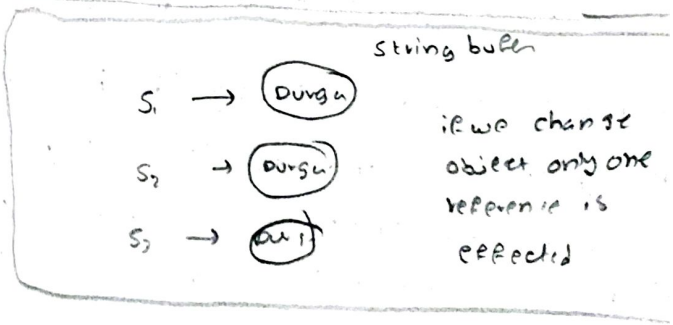
③ Inaddion to String objects any other objects are immutable in Java ?

All wrapper class objects are also immutable.

↓

Byte	Float
Short	Double
Int	Character
long	Boolean





Important Constructors of String class

① String s = new String (); Creating equivalent String object

② String s = new String (String literal);

③ String s = new String (StringBuffer sb);

④ String s = new String (StringBuilder sb); For given StringBuilder an equivalent String object is created.

⑤ String s = new String (char[] ch);

```
char[] ch = {'j','a','v','a'};
String s = new String(ch);
s.o.pln(s); → Java
```

Important methods of String class:

① public char charAt(int index);

```
byte[] b = {97,98,99,100};
String s = new String(b);
s.o.pln(s); abcd
```

② public String concat (String s);

```
String s = "durga";
s.o.pln(s.charAt(3)); g
```

③ Public boolean equals (String s or Object o) (overriding version)

④ Public boolean equalsIgnoreCase (String s)

⑤ Public boolean isEmpty()

⑥ Public int length()

⑦ Public String replace (char old, char new) abab → bbbb (a → b)

⑧ Public String subString (int begin) (spell spral)

⑨ Public String subString (int begin, int end)

⑩ Public int indexOf (char ch);

⑪ Public int lastIndexOf (char ch);

⑫ Public String toLowerCase() & toUpperCase()

```
String s = "durga";
s = s.concat("softwave");
```

S += ✓

① s → durga

② s → durga

durga softwave

③ s → durga

durga softwave

String	Array
s.length()	s.length

NOTE

```
String s = "abcdefg"
s.substring(3);
↓
defg
```

```
String s = "DURGA"
s.o.pln(s.equal("durga"));
s.o.pln(s.equalsIgnoreCase("durga"));
```

o/p → False, True

final vs immutability

if we add final to StringBuffer is it provide immutability → No

> if we dont want to change the object then it is immutability
>
> if we dont want to reassigning the object is is final

```
class Test
{
  P.S.V main (String[] args)
  {
    final StringBuffer sb = new StringBuffer ("durga");
    sb.append ("Softwave")
    S.O.println (sb) : → o/p :- durga Softwave
  }
}
```

sb = new StringBuffer ("ravi"): → CE → re assignment is not possible

Important

- ✓ final Variable — which one is meaningful
- ✗ final object
- ✗ Immutable variable
- ✓ Immutable object

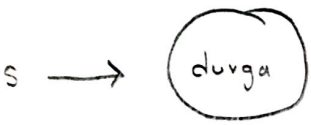
Impossable to make StringBuffer immutable.

StringBuffer

if the content is fixed then go with String if the content is keep on changing frequently then go with StringBuffer. for frequent change a new object is created in String internally. So in StringBuffer we can change easily.

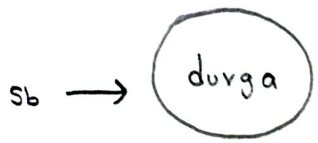
sb.capacity() → o/p = 16

String s = new String ("durga");



length: 5 capacity: 5

StringBuffer sb = new StringBuffer ("durga");



New capacity = (current capacity + 1) 2

length: 5 capacity-16 - default

after reaching 16 char new capacity allocated

Constructors of StringBuffer

① StringBuffer sb = new StringBuffer();

② StringBuffer sb = new StringBuffer(int initialcapacity); ← mention How much capacity you need

let 1000 after 1000 it normally increase

③ StringBuffer sb = new StringBuffer(String s);

↓

Capacity = s.length() + 16

```
SB sb = SB("durga");
S.o.pln(sb.capacity());
o/p = 21
```

Capacity = s.length() + 16

Important methods of StringBuffer;

① Public int length();

② Public int capacity();

③ Public char charAt(int index) →

```
SB sb = new SB("durga");
S.o.pln(sb.charAt(3)); g
S.o.ln(sb.charAt(30));
```

String index out of bounce

④ Public void setCharAt(int index, char newchar)

⑤ Public SB append(String s)

append(int i) we can take any type (overloaded)

⋮

⑥ Public SB delet(int begin, int end); (begin = index)

⑦ Public SB deletCharAt(int index);

⑧ Public SB reverse();

⑨ Public void setLength(int length);

```
SB SB = new SB("AiswavyaAbhi");
Sb.setLength(8);
S.o.pln(sb) → Aiswavya
```

⑩ Public void ensuveCapacity(int capacity)

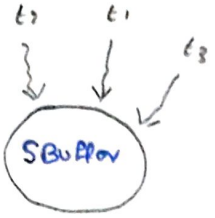
⑪ trimToSize();

```
Sb SB = new SB();
S.o.ph(sb.capacity()); 16
Sb.ensuveCapicity(1000);
S.o.ph(sb.capacity()); 1000
sb.trimToSize();
S.o.pln(sb.capacity());
```

ie sb = abc the o/p = 3 Not default - 16

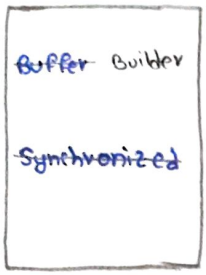
StringBuilder

Every method present inside String method is Synchronized



↓

The process that allows only one thread at a particular time to complete a given task entirely.
(Performance problem)
(increases waiting time of threads)
So StringBuilder is introduced to over come this problem.



Just replaced Buffer to Builder and Synchronized is removed and save file as StringBuilder.Java

StringBuffer.Java
StringBuilder.Java

non-synchronized version of StringBuffer is known as StringBuilder.
methods & Constructors are same as StringBuffer.

StringBuffer	StringBuilder
① most of the methods present inside StringBuffer are synchronized	① No method is synchronized
② At a time only one thread is allowed to operate on StringBuffer object and hence it is thread safe	② At a time multiple threads are allowed to operate on StringBuilder object and hence it is not thread safe.
③ Threads are required to wait to operate on StringBuffer object and hence relatively performance is low	③ Threads are not required to wait to operate on StringBuilder object and hence relatively performance is high
④ introduced in 1.0V	④ introduced in 1.5 V

String is also thread safe
all wrapper class object are thread safe

method chaining

Return type is Sb again

Sb.m1().m2().m3() — method chaining

StringBuilder

- SB append()
- SB reverse()
- SB insert()
- SB delete()

Return type is Same

Ex:-

```
StringBuilder sb = new StringBuilder();
sb.append("durga").append("solution").reverse().insert(2,xyz);
So.pln(sb)
```

Collections

Arrays.

```
int x=10;
int y=20;
int z=30;
.
.
.
1000 values
```

int[] x = new int[1000];

x → | 10 | 20 | 30 | .. .. .. | 1000 |

1001 element is not possible

after declaration we can't increase or decrease the size

① Fixed in size

② arrays can hold only homogeneous elements (use object array)

③ For every requirement we need to write program like sort, is array is in the arry. For every sorting programer must write the program, (for searching also) there are no readymade methods.

There are some limitations in array - using single variable to store multible elements.

> an array is an indexed collection of fixed no of homogenious data elements

To over come this we uses collection concept.

Collections

① growable in size

② can hold both homogenious & hetrogenious Data

③ Ready made methods are avalilable.

> Array - good performance
> Collection - good memory grow

In there are advantages there must be some disadvantages

* If we know the size inatilly it is recomended to go with Array.

what happen when we have collections rather than array in size.

① l → | 0 | 0 | 0 | . . . | 0 |

length is full if we want to add another element impossible in array comming to collections

l → | 0 | 0 | 0 | . . . | 0 | ← 11th element

② New arraylist of bigger size than previous one is created

③ Now each element from previous arraylist is copied into new awaylist of big size.

④ garbage collection

reference is changed & garbage collection is done.

See if there are 1crore element & we need to insert 1crore+1 element.

(Poor performace) many copy task is requires

Array	Collections.
① Fixed size	① growable size
② memory wise ↓	② memory wise ↑
③ performance ↑	③ Performace ↓
④ only homogenious	④ Both homogenious & hetrogenious.
⑤ No Redymade methods	⑤ Ready made methods available.
⑥ Hold premitive & objects	⑥ only objects.

Collection :-

if we want to represent a group of individual objects as a single entity then we should go for collection.

Collection Frame work :-

Contain several classes & interfaces which can be used to to represent a group of individual objects as a single entity.

9 key interfaces of collection Frame work

most common

① Collection — for every collection objects, method required like add, remove, delet are available in collection intrface.

② List

③ Set

④ Sorted Set

⑤ Navigable set

⑥ Queue

⑦ map

⑧ Sortedmap

⑨ Navigable map.

Collection Interface is ROOT

collection (I) → if we want to represent a group of individual objects as a single entity then we should go for collection.

→ collection interface defines the most common methods which are applicable for any collection object

→ in general collecton interface is considered as root interface of collection frame work

There is no concrete class which implements collection interface directly.

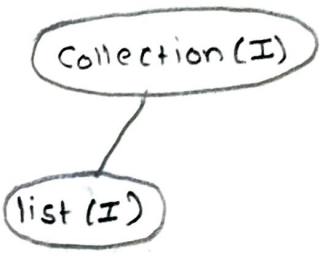
Q) What is difference between Collection & Collections

↓ Interface ↓ Class

Collection is an interface if we want to represent a group of individual objects as a single entity then we should go for collection

Collections is an utility class present in java.util package to define several utility methods for collection objects like sorting, searching etc.

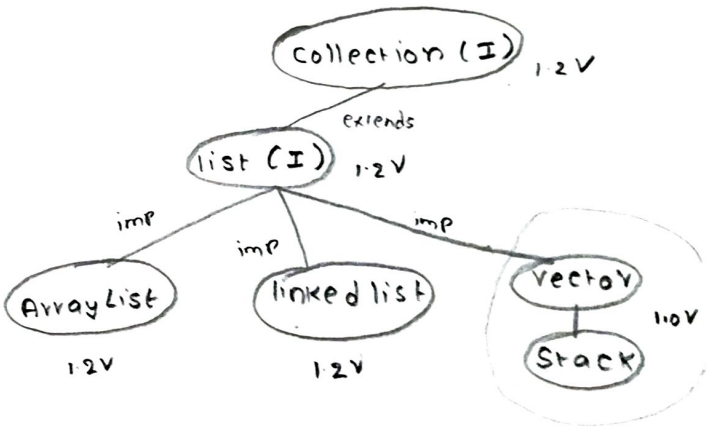
Collections.sort(l) → array list



① list (I)

it is the child interface of collection

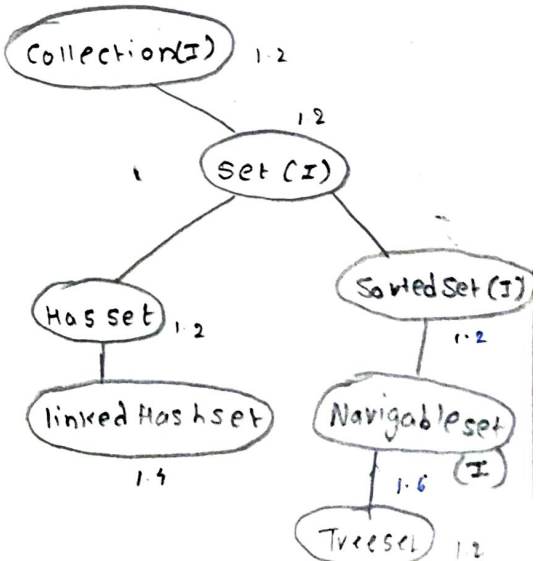
if we want to represent a group of individual objects as a single entity where duplicates are allowed and insertion order must be preserved then we should go for list.



③ Set (I)

it is the child interface of collection

if we want to represent a group of individual objects as a single entity where duplicates are not allowed and insertion order not required then we should go for Set



④ Sorted set (I)

it is the child interface of Set if we want to represent a group of individual objects as a single entity where duplicates are not allowed & all objects should be inserted according to some sorting order then we should go for Shorted set

⑤ Navigable Set (I)

it is the child interface of Sorted set. it contains several methods for Navigation purposes

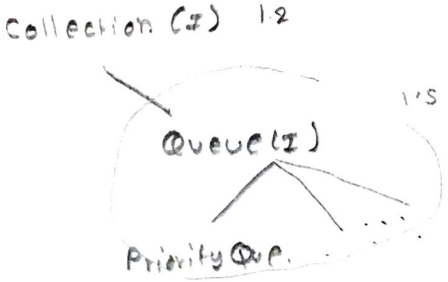
⑥ What is diff b/w list & set

To declare a class that implements an interface, you include an implements clause in the class declaration. your class can implement more than one interface so implements keyword is followed by a comma-separated interface implemented by class

⑥ Queue (I)

it is the child interface of collection if we want to represent a group of individual objects prior to processing then we should go for Queue. usually Queue follow first-in-first out But based on our requirement our own priority order also

Ex:- Before sending a mail all mailid's we have to store in some data structure in which order we added mail ids in the same order only mail should be delivered. for this requirement Queue is best choice.



all the above interfaces (Collection, list, Set, Sorted set, Navigable set & Queue) ment for representing a group of individual objects if we want to represent a group of objects as key value pairs then we should go for map

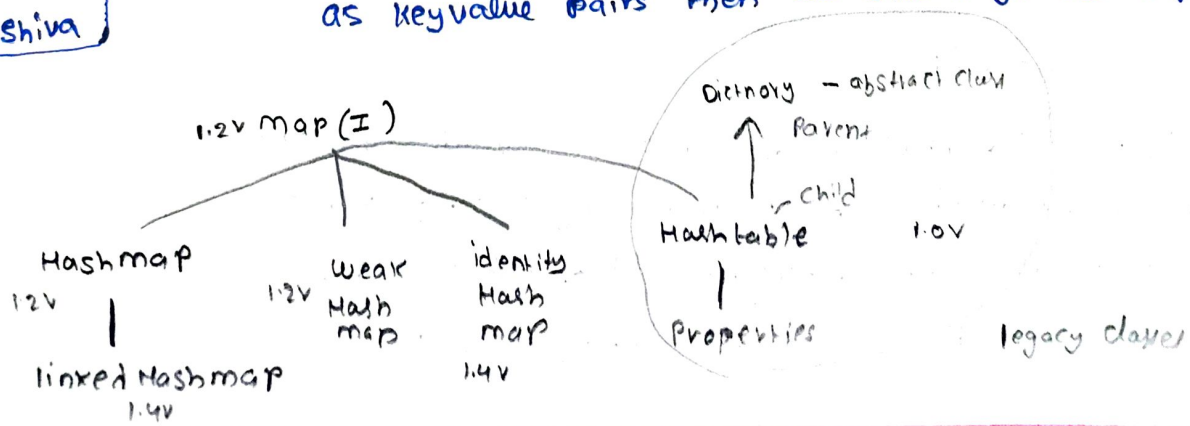
⑦ map (Key-value pairs)

Key	value
S.No	Name
101	durga
102	Ravi
103	Shiva

Duplicates Not allowed

map is not child interface of collection.

if we want to represent a group of objects as keyvalue pairs then we should go for map



Sorted map (I)

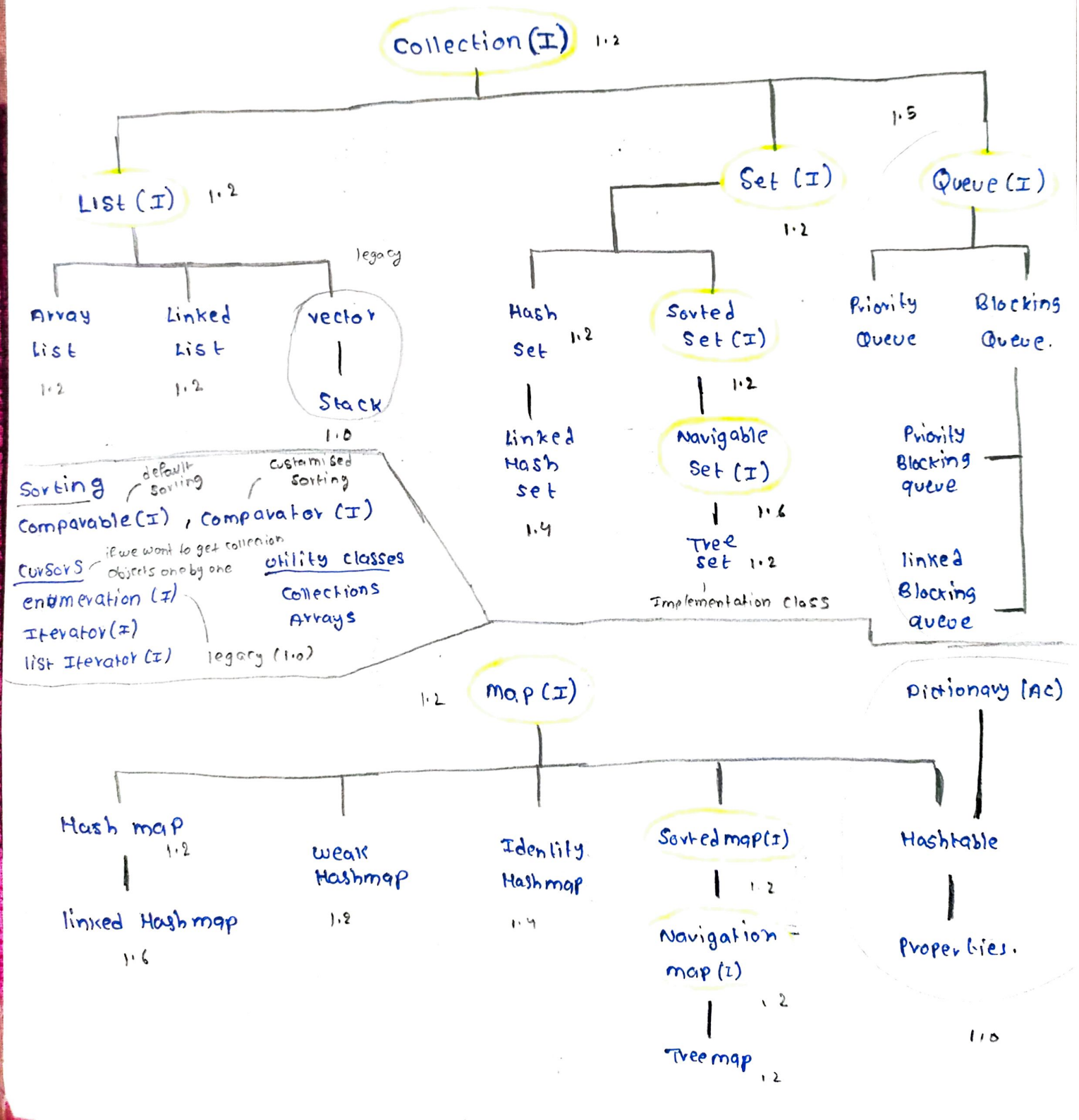
Child interface of map if we wont to represent a group of key value pairs according to some sorting order of keys. Then we should go for Sorted map (I)

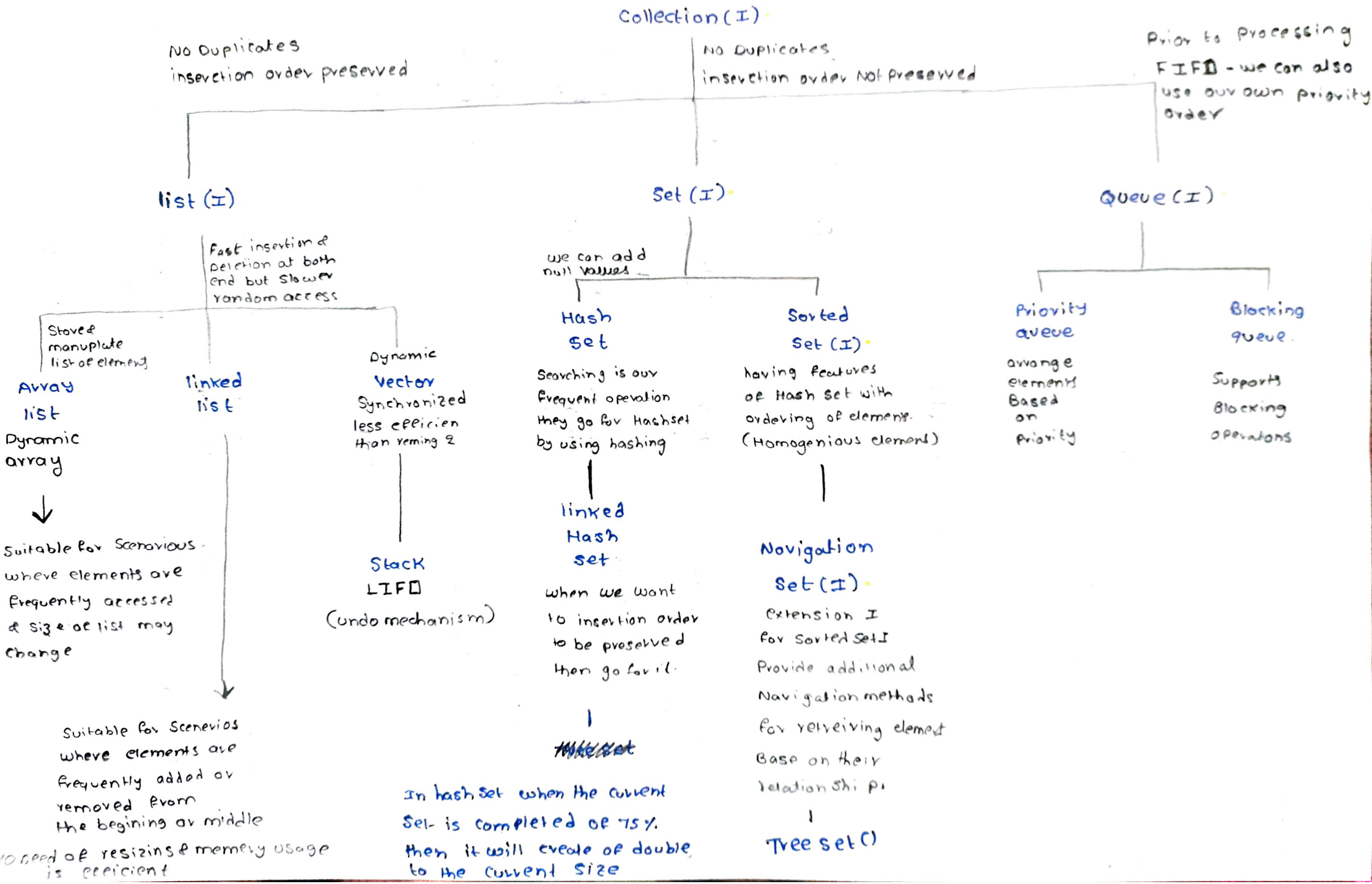
In Sorted map the sorting should be based on key but not based on value

map (I) 1.2v
- Sorted map (I) 1.2
  - Navigable map (I) 1.6
    - Tree map 1.2

Navigable map (I)

It is the child interface of Sorted map it defines several methods for navigation purposes




Collection (I)
No Duplicates
insertion order preserved
No Duplicates
insertion order Not preserved
Prior to Processing
FIFO - we can also use our own priority order
list (I)
Set (I)
Queue (I)
Fast insertion & deletion at both end but Slower random access
we can add null values
Stored manuplate list of elements
Array list
Dynamic array
Suitable for Scenarious where elements are frequently accessed & size of list may change
linked list
Suitable for Scenerios where elements are frequently added or removed from the begining or middle
No need of resizing & memory usage is efficient
Dynamic
Vector
Synchronized
less efficien than reming &
Stack
LIFO
(undo mechanism)
Hash set
Searching is our Frequent operation they go for Hashset by using hashing
linked Hash set
when we want to insertion order to be preserved then go for it
In hash set when the current set is completed of 75%. then it will create of double to the current size
Sorted Set (I)
having features of Hash set with ordering of elements. (Homogenious elements)
Navigation Set (I)
Extension I for Sorted SetI
Provide additional Navigation methods for retreiving element Base on their relationship
Tree set()
Priority queue
arrange elements Based on Priority
Blocking queue
Supports Blocking operations

Key-Value Pairs

map (I)

Dictionary (AC)

Hash map

Key-Value pair

widely used implemention of map interface.

it uses hash table to store key-value pairs & provide constant time complexity for basic opevations.

linked Hash map

maintain order in which key were inserted

weak Hashmap

Keys of the map ave stored in a weakReference. This means the garbage collector can collect the keys even if they ave still refevenced by map.

This can be useful in situations where you want to store a cache of objects & we don't want the cache to keep growing.

Identity Hash map

unlike standard implementations like Hashmap. which uses the equal() method for comparison

identity Hashmap uses reference equality (==) in place of object equality (equals())

when comparing keys (e-values)

Sorted map (I)

Some sorted order of keys

No values sorted order

Navigable map(I)

Defines several methods for Navigation purpose

Hash table

Hash table class implements a hash table, which maps keys to values Any non-null object can be used as a key or as a value to sucessfully store & retrieve objects from a hashtable. the object used as a keys must implement the hashCode method & the equals method.

```
Import java.util.*;

List <String> list1 = new ArrayList();
List <String> list2 = new LinkedList();
List <String> list3 = new Vector();
List <String> list4 = new Stack();

Set <String> S1 = new HashSet();
Set <String> S2 = new LinkedHashSet();
Set <String> S3 = new TreeSet();

Queue <String> q1 = new PriorityQueue();
Queue <String> q2 = new ArrayDeque();

Map <Integer, String> m1 = new HashMap();
Map <Integer, String> m2 = new LinkedHashMap();
```

ArrayList<String>(); .add(i,E) (element index) (retrive & remove ist element)

.add(), .get(), .isEmpty()

getFirst(), getLast(), .poll(), .pollFirst, pollLast

.addFirst(), .addLast(), .clear(), .lastIndexOf()

firstElement(), .indexOf, remove()

(return top element)

empty(), push(), pop(), peek(), search()

(Add all)

size(), toArray()

add(), addAll(collection), clear(), remove(), removeAll

.add(), .addAll(collection), clear(), remove(), removeAll

Cursor – an iterator used to iterate or traverse or retrive a Collection or Stream objects elements one by one.

if we want to retrieve objects one by one from the collection, then we should go for cursors.

There are three types of cursors are available in Java.

① Enumeration

```
Enumeration e = V.elements();
```
(V: vector object)

has two methods

① hasmoreElements();

② nextelement();

we can use enumeration to get objects one by one from the old Collection objects (legacy Collections).

we can create enumeration object by using elements() method of Vector class

* Public boolean hasmoreElements();
* Public Object nextElement();

```
Vector V = new Vector();
for (int i=0; i<=10; i++)
{
    V.addElement(i);
}
S.o.println(V); → [0,1,2,3,4,5,6,...10]

Enumeration e = V.elements();
while (e.hasMoreElements())
{
    Integer I = (Integer) e.nextElement()
    if (I%2 == 0)
        S.o.println();
}
```

only forward direction

Printing even number

② Iterators (universal cursor)

There are some limitations in enumeration

↓

- only for legacy collections
- No remove operation

```
Iterator itr = c.iterator();
                |
            any collection object
```

methods

① Public boolean hasNext();

② Public Object next()

③ Public void remove()

limitations of Iterator

only forward direction

No Replace.

No Adding element

```
AL l = new AL();
for (int i=0; i<=10; i++)
{
    l.add(i)
}.
S.o.pln(l); [0,1,2,3 ... 10]
Iterator Itr = l.iterator();
while (itr.hasNext())
{
    Integer I = (Integer)itr.next();
    if (I%2 == 0)
        S.o.pln(I);
    else
        itr.remove();   [0,2,4,6,8,10]
}
```

ListIterator (most powerful Cursor)

Bidirectional Cursor

Read
Remove
Replace } ✓
Add

```
ListIterator ltr = l.listIterator();
```

methods

```
Public boolean hasNext()
Public Object  next()
Public int     nextIndex()
```
} Forward

```
Public boolean hasPrevious()
Public object  previous()
Public int     previousIndex()
```
} Backward

```
Public void remove()
Public void set(object new)
Public Void add(object new)
```

```
LinkedList l = new LinkedList();
l.add("Balakrishna");
l.add("venki");
l.add("chiru");
l.add("nag");
s.o.pln(l)
```

```
ListIterator ltr = l.listIterator();
while (ltr.hasNext()) {          (ie Next elements is there give
    String s = (String) ltr.next();
    if (s.equals("venki"))
    {
        ltr.remove();
    }
    else if (s.equals("nag"))
    {
        ltr.add("chaitu");
    }
    else if (s.equals("chiru"))
    {
        ltr.set("chavan");
    }
}
```

Set of instructions that can run independently

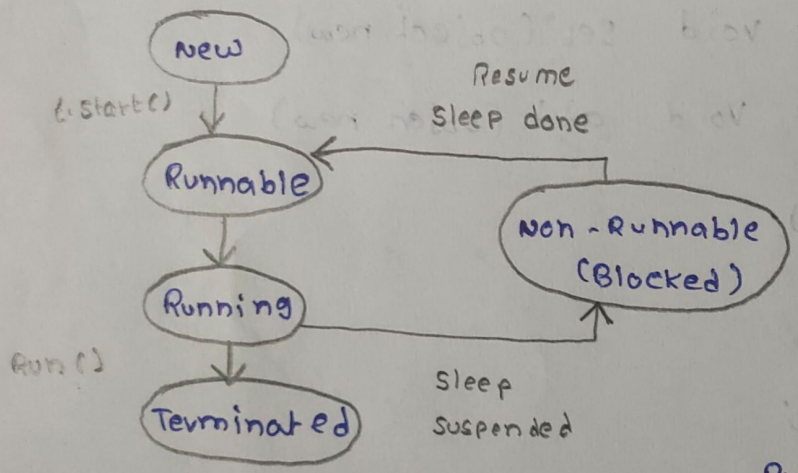
Thread Concept in Java

Before introducing thread concept, we are were unable to run more than one task in parallel. it was drawback, and to remove that draw back Thread concept is introduced.

Thread is very light-weighted process or we can say smallest part of the process that allows a program to

Another benfit of using thread is that if a thread gets an exception or an error at the time of its execution, it dosen't affect the execution of the other thread. All the threads share a common memory and have their own stack, local variables and program counters. When multiple threads are executed in parallel at the same time, the process is known as multithreading.

model



New : thread is in New when it gets CPU time

Runnable : under exigution

Running

we can define a thead by implementing runnable interface

Runnable interfac present in Java.lang it contains only one method (Public void Run()).

There are two ways to implement

1. By extending thread class — class thead1 extends Thread
2. By implementing Runnable interface — class thread1 implements Runnable.

Streams

Java Streams

if we want to represent a group of objects as a single entity then we should go for collection

if we want to process objects form the collection the we should go for Stream

Io stream always talks about data

Collection Stream always talks about collection data

Stream is a interface present in java.util.Stream pkg.

```
Stream s = c.stream();
```
(collection)

Ex!-

```
ArrayList<Integer> l = new ArrayList<Integer>();
l.add(0);
l.add(5);
l.add(10);
l.add(15);
l.add(20);
l.add(25);
List<Integer> l2 = l.stream().filter(i -> i%2==0).collect(Collectors.toList());
System.out.println(l2);
```