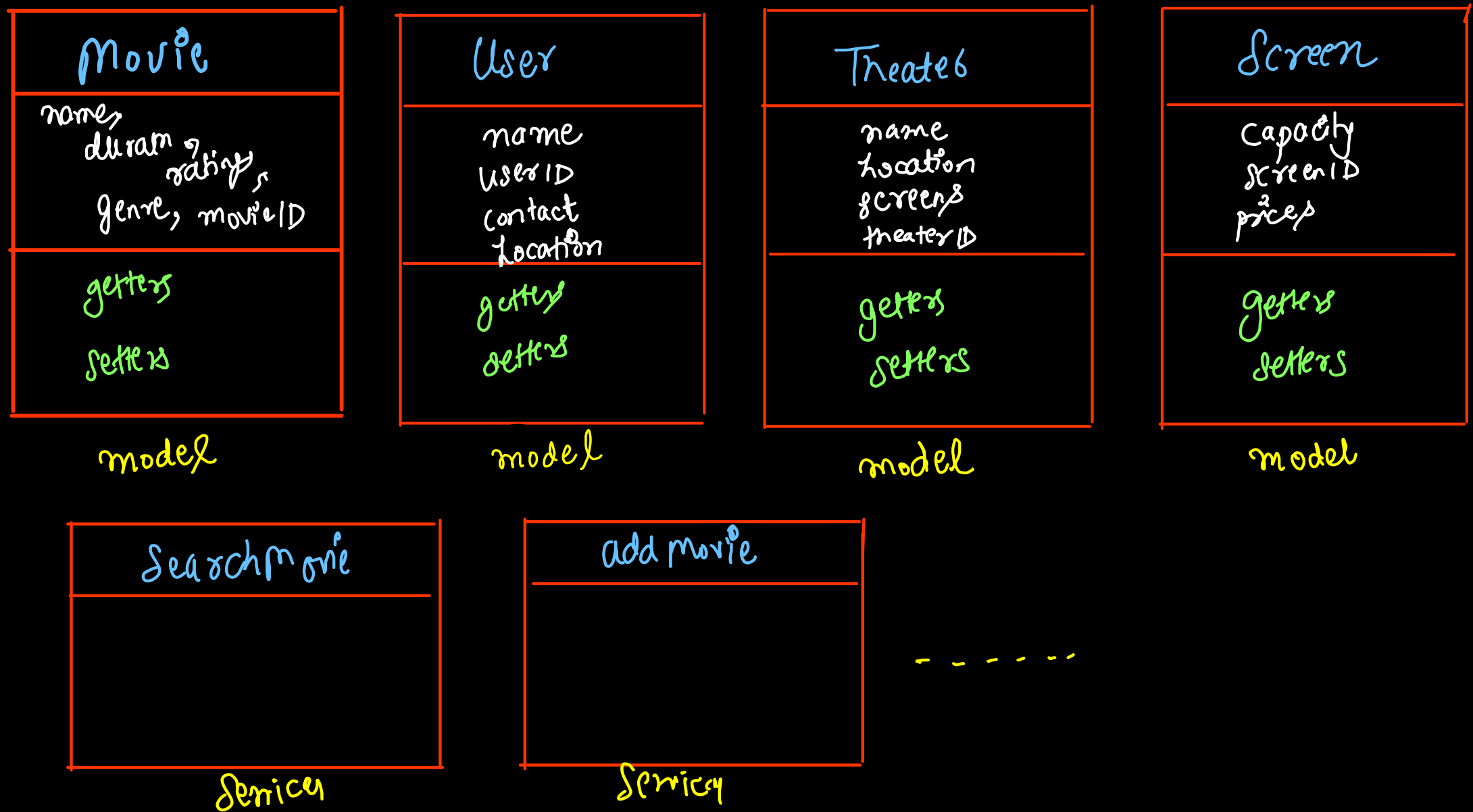
Design BookMyShow



Java Compilation & Execution using Terminal

Compilation :→ "javac filename.java"

↳ It will make .class files for each class in .java file

Execution :→ "java classname"

↳ class should have public static void main(String [] args) method, otherwise no main method found runtime error

↳ To run a program using a particular class, it is not necessary that class should be public.

Important

Public class must have same name as the .java filename. because JVM will look for an entry point, ie. class having main() method, So we are telling JVM that class which is public will be entry point.

Interview Questions

Q) Can classes or Interfaces be "private"?

A) Only inner classes/interfaces can be made private. But outermost (top level) class/interface are not allowed to be private, because if allowed, "no one" will be able to access them & it will become useless.

Q) Can we have no (zero) public classes in .java file?

A) Yes, it is possible but in that case, .java filename should not be same as any classname. Also to run the program, you should use "java classname" containing main method.

Q) Can we have more than one (>1) classes in same .java file?

Yes, we can have more than one classes in same .java file. But the constraint is there: atmost one class can be public. Rest all classes should be protected / default.

Q) Is it possible that no class have the main method?

There will be no compilation error, but there will be "no main method found" run-time error. (whichever class is used as entry point (classname used in "java classname") must have the main method in it.

Q) Is it possible that more than 1 classes have main method defined?

Yes, it is possible that multiple classes can have main method. But again, class used as entry point must have the main method defined.

Packages → used to implement "Data Hiding"

Advantages

- used to group related classes
- used to avoid name space conflicts
- It improves readability, maintainability & easy debugging.
- It is similar to "folder or directory" of Java classes.

Types of Package

- Built in Packages
  - java.lang → String, StringBuffer, Object, System, Math, Integer, etc
  - java.util → Collection frameworks (Arraylist, Hashmap, linkedlist, etc), Scanner
  - java.io → Input/Output (BufferedReader, BufferedWriter, File, InputStream, OutputStream, Serializable, etc)
- User Defined Packages
  - "package" keyword
  - "import" keyword

User Defined Packages

Note → We can have subpackages (nested directories)

→ Folder name is equivalent to package name

→ All java files in the package must have first line as "package packagename;"

→ All .class files (byte code) must be saved in the corresponding package directory / folder.

→ But java files (source code) can be saved anywhere, not necessarily within the same folder/package.

Compilation → "javac full path or relative path / javafilename.java"

Execution → "java fully qualified classname"

↳ packagename.subpackagename.classname

Three ways to import Packages

(1) Using Fully classified classname

eg `java.util.Scanner = new java.util.Scanner(System.in);`

(2) `import packagename.subpackagename.*;`

eg `import java.util.*;`

↳ Import everything from that subpackage

(3) `import packagename.subpackagename.classname.methodname;`

↳ optional (Import specific subpackage/class/method)

eg `import java.util.Scanner;`

↓ to import static method use "`import static ....`";

Note # If package is within the same working directory from where it is being used/imported, then we do not need to set class path environment variables

Class Paths (Required when driver code file & package dependency are in different directories)

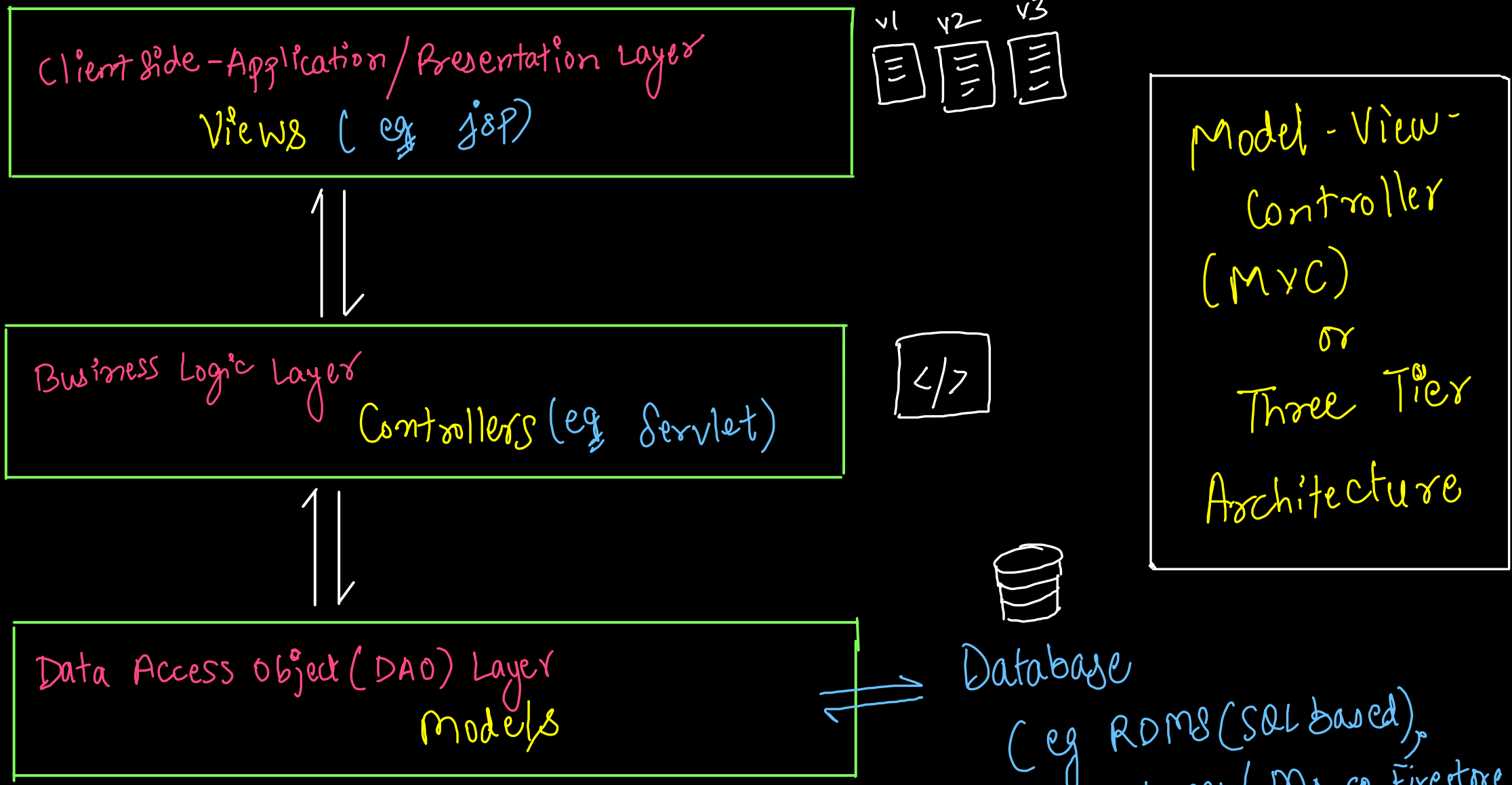
- Temporary Class Path Environment Variables
- Permanent class Path Environment Variables

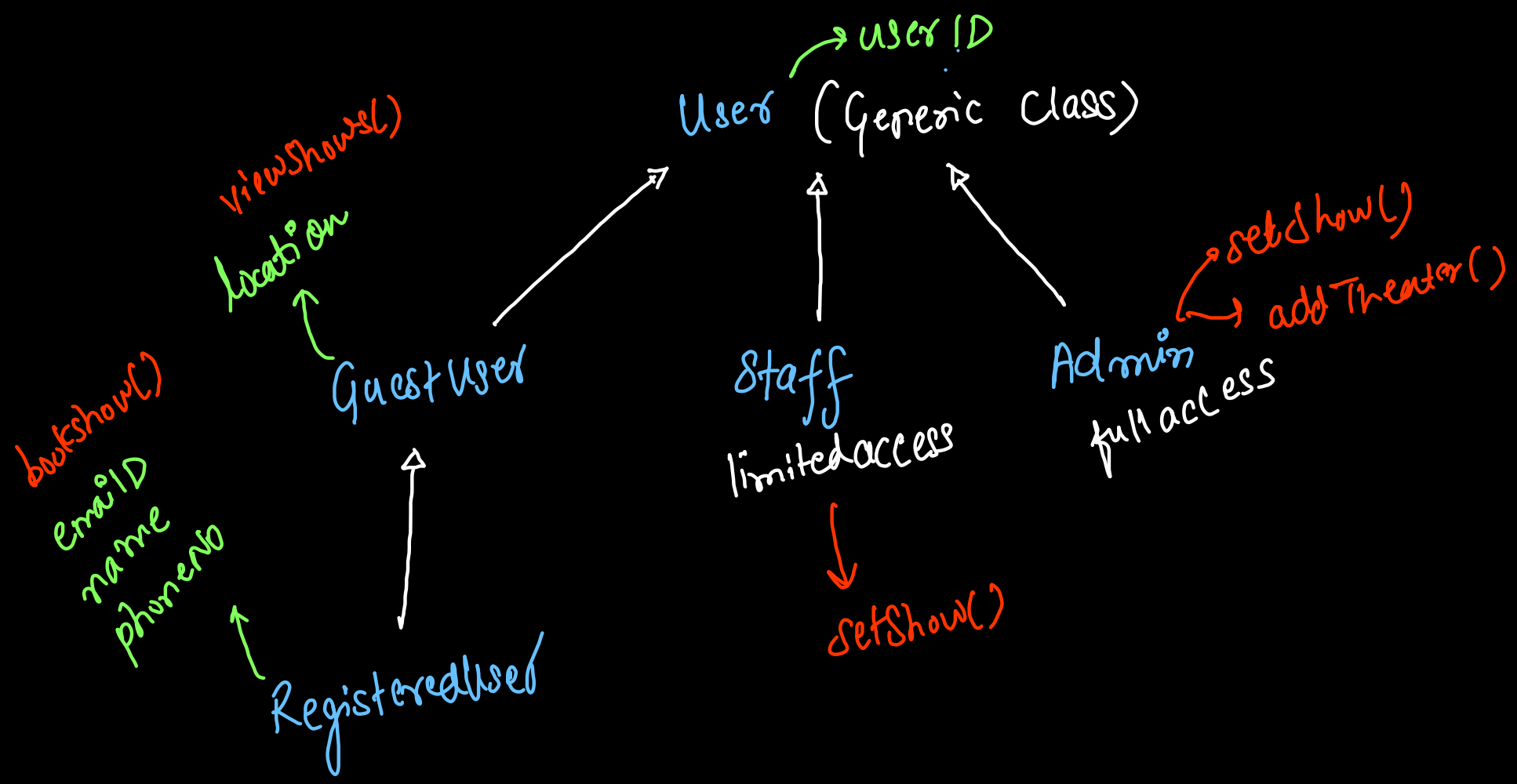
] Explore Yourself.

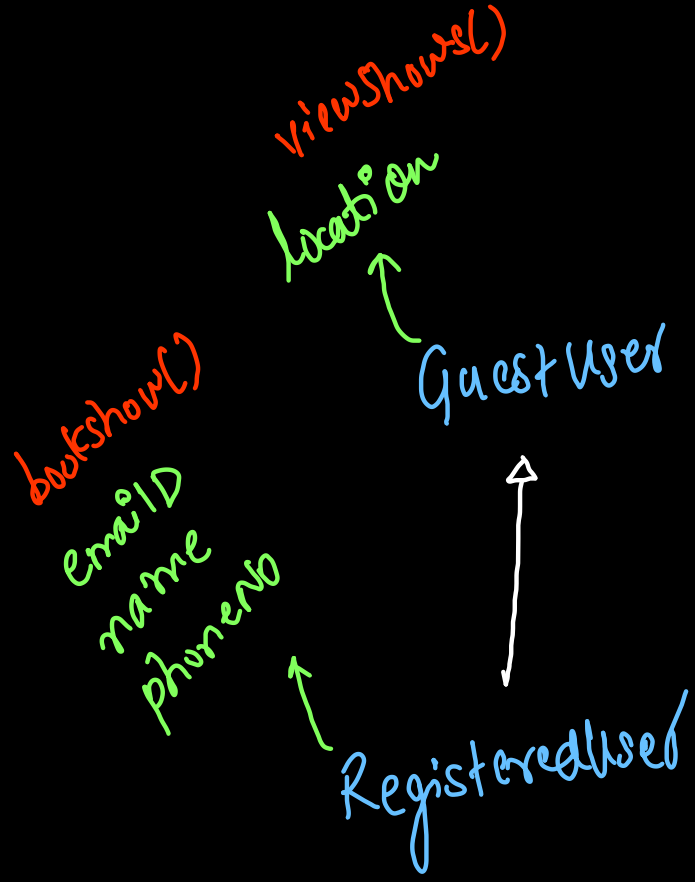
Jar file → Executable file for Distribution of Java programs

Scope of Different Access Specifiers

Access Specifier/ Modifier	Within Same class	Outside class within same package	Different Package Subclass	Different Package non-subclass	
public	Yes	Yes	Yes	Yes	everywhere
protected	Yes	Yes	Yes	No	everywhere except diff package non-subclass
default	Yes	Yes	No	No	same package
private	Yes	No	No	No	within class only


Client side - Application / Presentation Layer
Views (eg jsp)
v1
v2
v3
Model - View - Controller (MVC) or Three Tier Architecture
Business Logic Layer
Controllers (eg Servlet)
</>
Data Access Object (DAO) Layer
Models
Database
(eg RDMS (SQL based), NonSQL (Mongo, Firestore, etc)


User
(Generic Class)
userID
viewShows()
location
GuestUser
bookShow()
emailID
name
phoneNo
RegisteredUser
Staff
limitedaccess
setShow()
Admin
fullaccess
setShow()
addTheater()



RUser User = new RUser();

child of parent class object

child RUser() call ① Execute emailID name phoneNo

super() 1st line

parent GUser() call ② Execute locatn = 'India'

back track

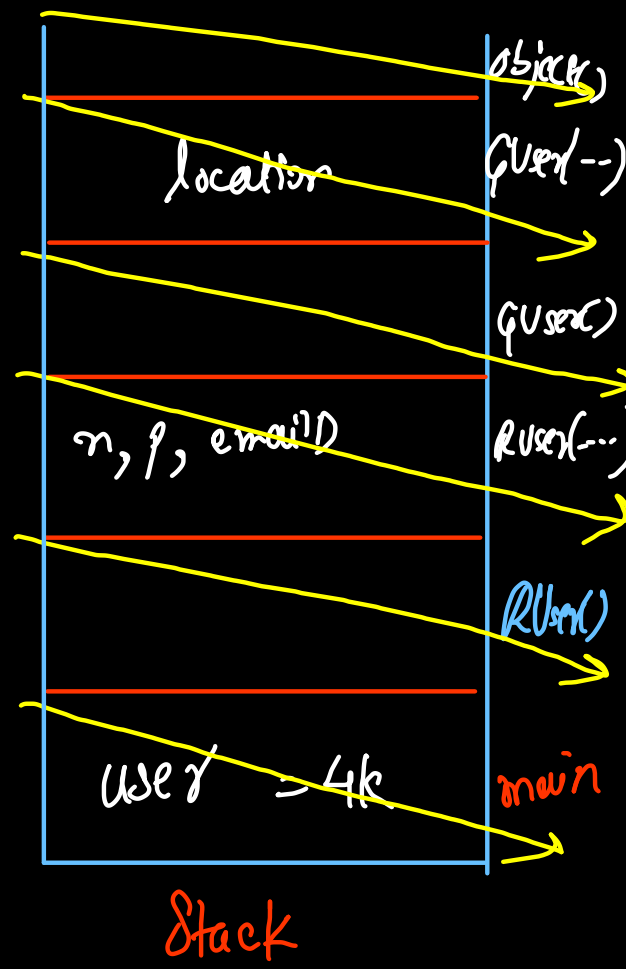
RUser() → RUser(...) → super() → super(...) → Super() object

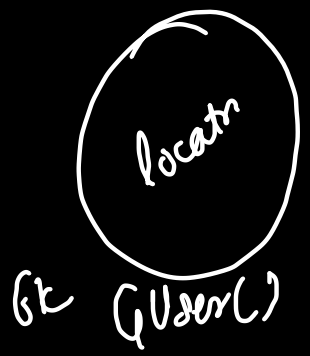
GU() GU(..)

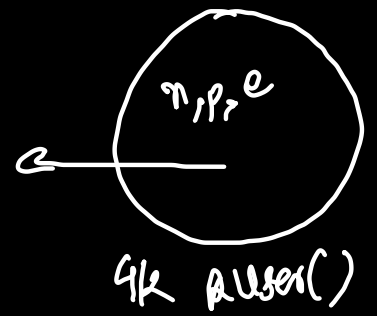
```java
public class RegisteredUser extends GuestUser{
    public String name;
    public long phoneNo;
    public String emailID;

    public RegisteredUser() {
        this(name: "Anonymous", phoneNo: 0l,
        emailID: "anonymous@gmail.com");

        System.out.println(x: "RUser Empty Constructor");
    }

    RegisteredUser(String name, long phoneNo, String emailID) {
        super();
        System.out.println(x: "RUser Parameter Constructor");
        this.name = name;
        this.phoneNo = phoneNo;
        this.emailID = emailID;
    }

    public void bookShow() {…
}
```

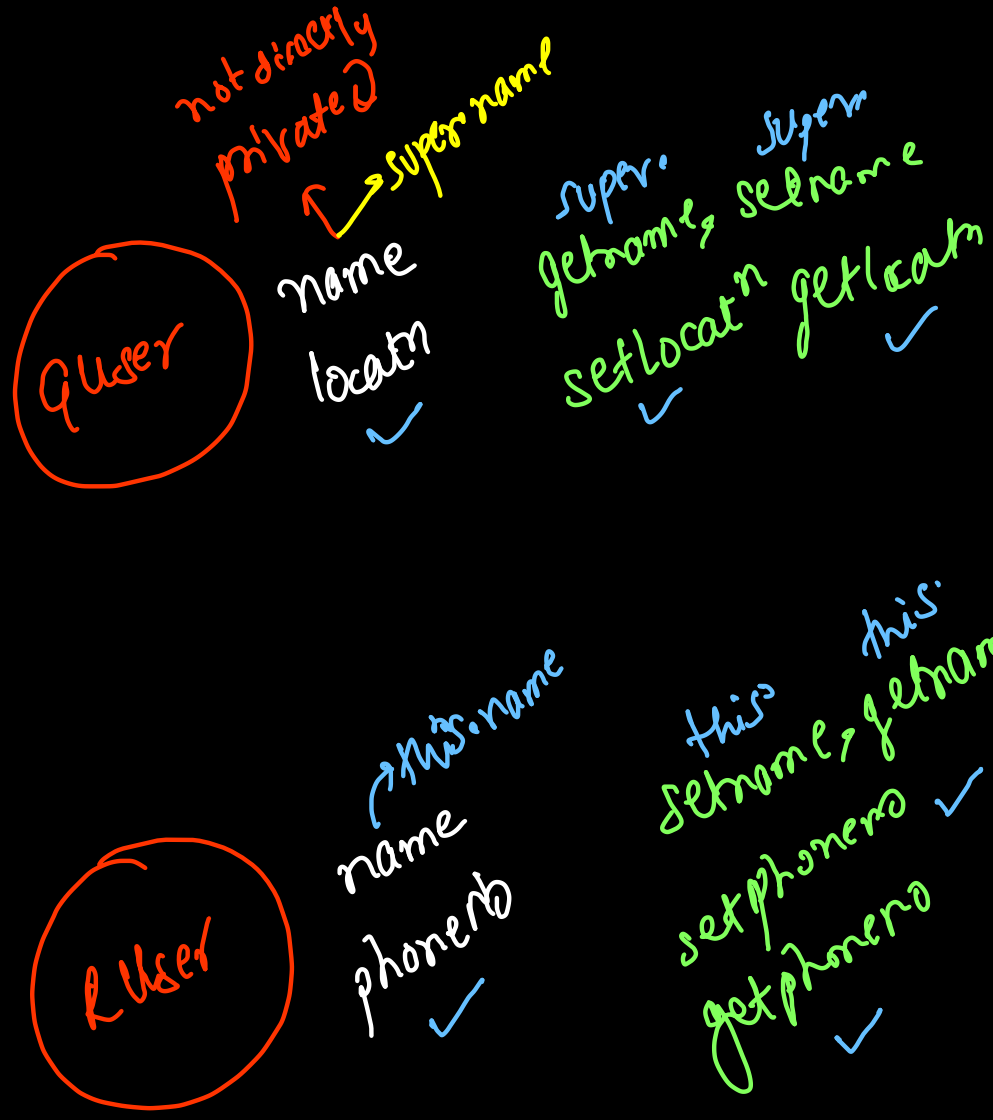
```java
package BookMyShow.users;

public class GuestUser {
    public String location;

    GuestUser() {
        this(location: "India");
        System.out.println(x: "GUser Empty
        Constructor");
    }

    GuestUser(String location) {
        this.location = location;
        System.out.println(x: "GUser Parameter
        Constructor");
    }

    public void viewShow() {…
}
```

```java
public class App {
    Run | Debug
    public static void main(String[] args) {
        RegisteredUser user = new RegisteredUser();
```







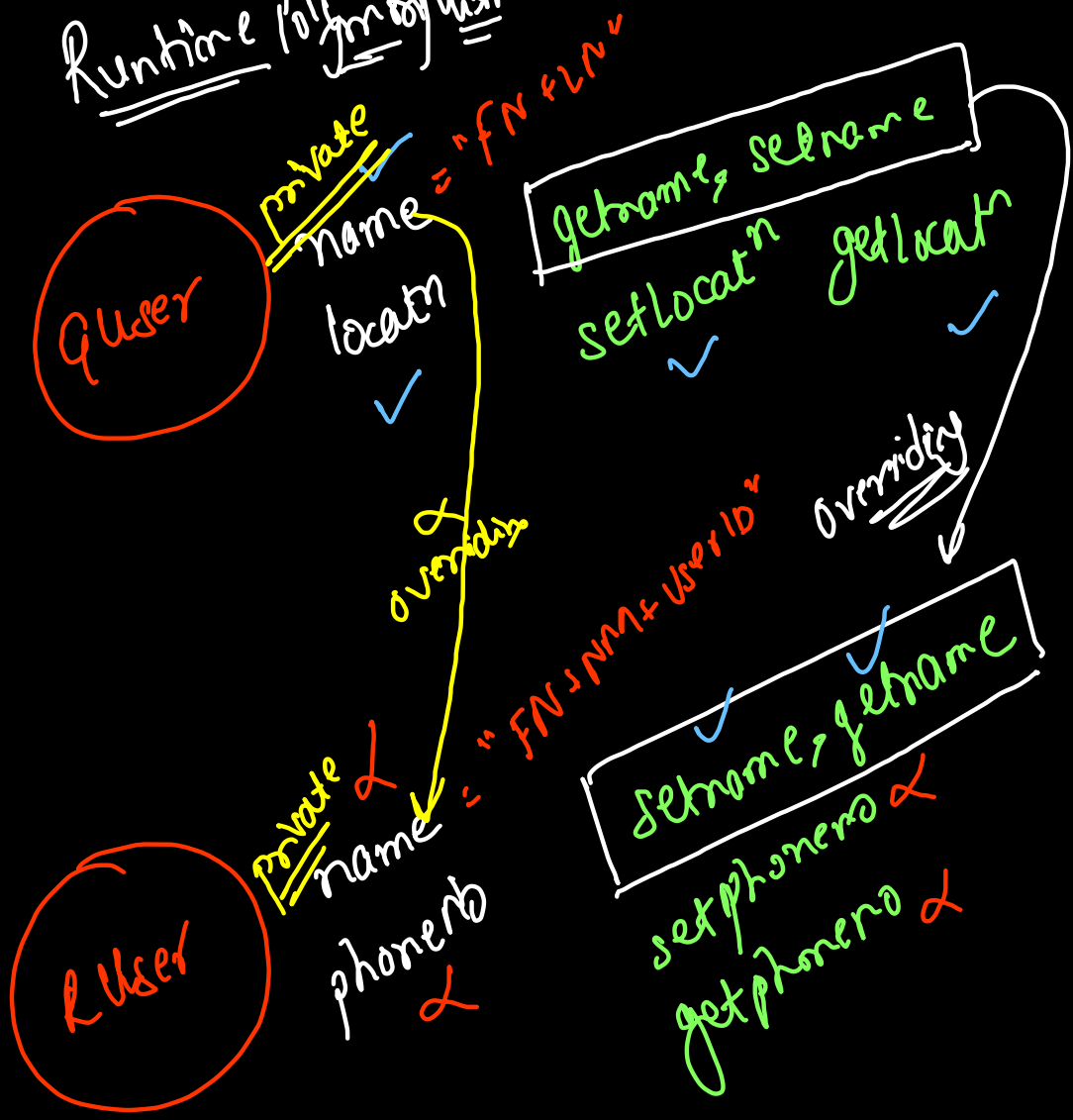


(2) Child obj2 = new Child();

reference

object

Runtime Polymorphism



properties(own)
functions (accessible)

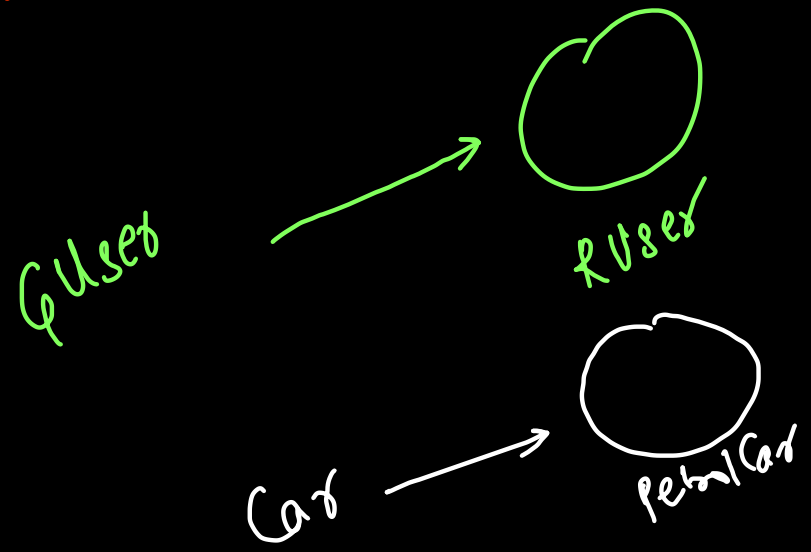
functn (inhrte)
call

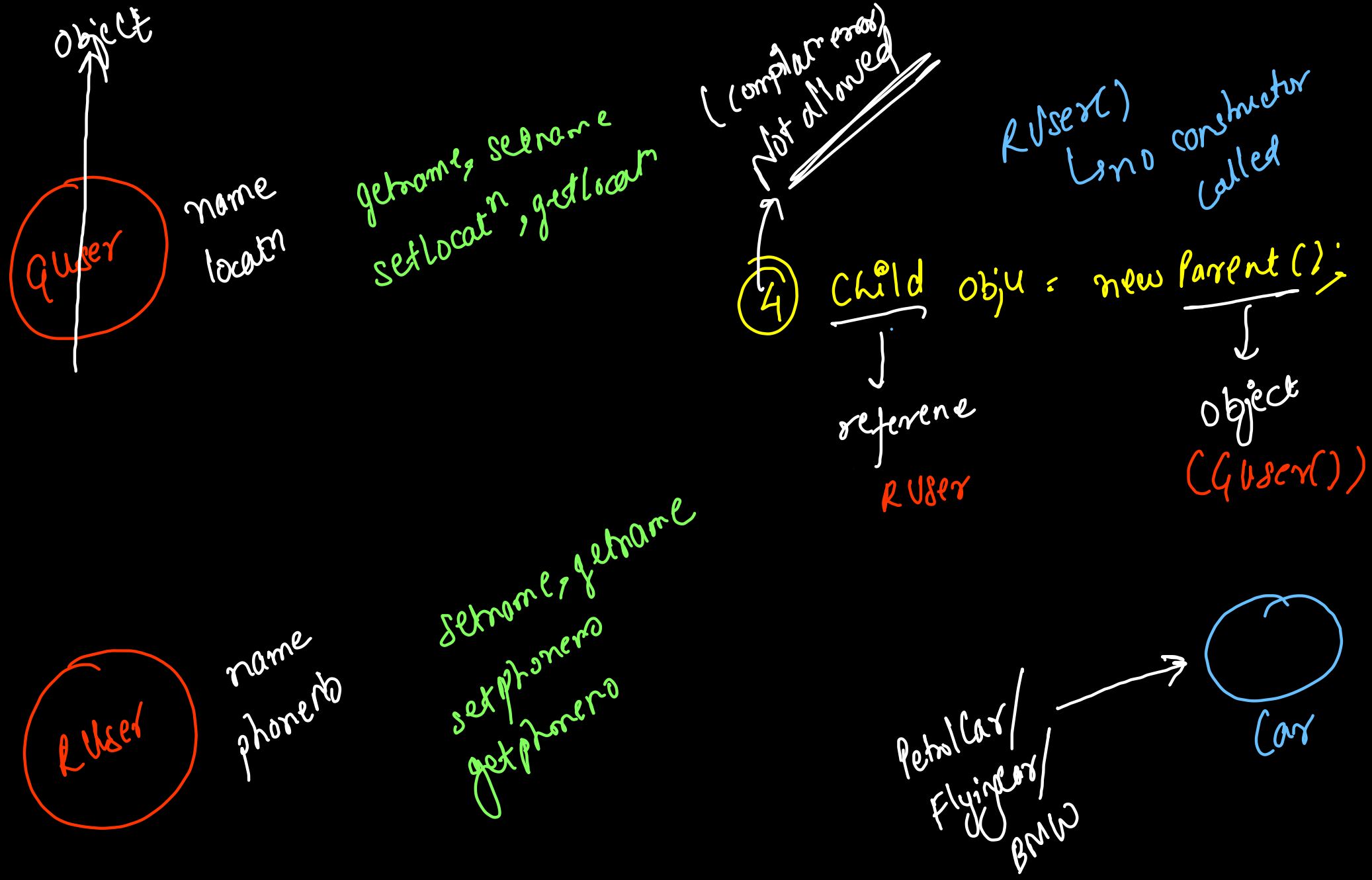
③ `Parent obj3 = new Child();`

reference

Object

`GUser obj3 = new RUser();`




Object
GUser
name
locatn
getname, setname
setlocatn, getlocatn
((compiler error)
Not allowed
4
Child obju = new Parent();
reference
RUser
RUser()
Lno constructor
called
Object
(GUser())
RUser
name
phoneNo
setname, getname
setphoneno
getphoneno
PetrolCar/
Flyingcar/
BMW
Car

Interview Question

```java
public class Class1 {

    public void method(Object obj){
        System.out.println("Object");
    }

    public void method(String str){
        System.out.println("String");
    }

    public static void main(String... arg){
        new Class1().method(null);
    }

}
```

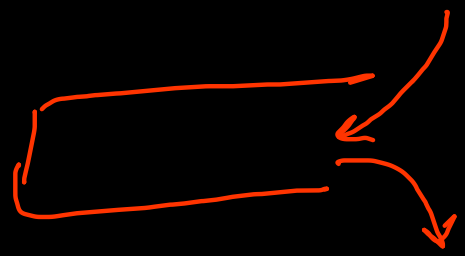
generic

specific

answer

"String" → specific class

Stack (LIFO)



ArrayList

addLast()

removeLast()

LinkedList

addFirst()

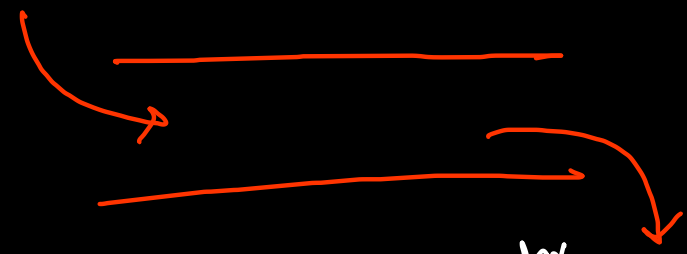
removeFirst()

Queue (FIFO)

LinkedList

addLast()

removeFirst()

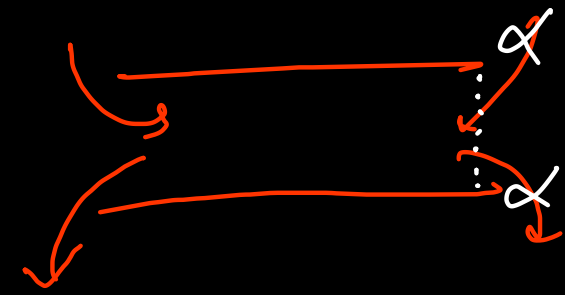


addFirst()

removeLast()

Circular Array

Deque



addFirst()

removeFirst()

addLast()

removeLast()

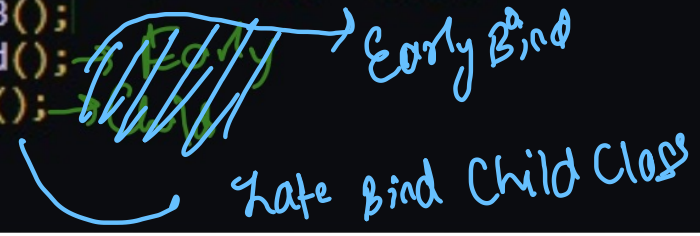
```java
static class Stack{
    ArrayDeque<Integer> q = new ArrayDeque<>();

    public void addLast(int data){
        System.out.println("Add Last of Stack");
        q.addLast(data);
    }

    public int removeLast(){
        System.out.println("Remove Last of Stack");
        return q.removeLast();
    }
}
```

```java
static class Queue{
    ArrayDeque<Integer> q = new ArrayDeque<>();

    public void addFirst(int data){
        System.out.println("Add First of Queue");
        q.addFirst(data);
    }

    public int removeLast(){
        System.out.println("Remove Last of Queue");
        return q.removeLast();
    }
}
```

```java
static class Deque extends Queue{
    ArrayDeque<Integer> q = new ArrayDeque<>();

    public void addLast(int data){
        System.out.println("Add Last of Deque");
        q.addLast(data);
    }
    public void addFirst(int data){
        System.out.println("Add First of Deque");
        q.addFirst(data);
    }
    public int removeLast(){
        System.out.println("Remove Last of Deque");
        return q.removeLast();
    }
    public int removeFirst(){
        System.out.println("Remove First of Deque");
        return q.removeFirst();
    }
}
```

```java
public static void main(String[] args){
    Queue q = new Deque();
    q.addFirst(10); // Allowed
    // q.addLast(20); Not Allowed
    System.out.println(q.removeLast()); // Allowed
    // System.out.println(q.removeFirst()); // Not Allowed

}
```

```java
class A{
    public void earlyBind(){
        System.out.println("Early Bind");
    }

    public void lateBind(){
        System.out.println("Late Bind in Parent Class");
    }
}

class B extends A{
    @Override
    public void lateBind(){
        System.out.println("Late Bind in Child Class");
    }
}

public class Main{
    public static void main(String[] args){
        A obj = new B();
        obj.earlyBind();
        obj.lateBind();
    }
}
```



Overloaded

↳ Same class a functions

Overided class

Parent - child class

↓ ref ↓ object

Abstract Datatype

abstract

Stack, Queue

af rf af rl

Priority queue

add remove

Concrete Data Structure

Implementation

Deque

Binary Heap

```java
static abstract class Stack{
    ArrayDeque<Integer> q = new ArrayDeque<>();

    public void addLast(int data){
        System.out.println("Add Last of Stack");
        q.addLast(data);
    }

    public int removeLast(){
        System.out.println("Remove Last of Stack");
        return q.removeLast();
    }
}
```

```java
public static void main(String[] args){
    Stack stk = new Stack();
}
```

```
Finished in N/A
Line 54: error: Stack is abstract; cannot be instantiated
[in Main.java]
        Stack stk = new Stack();
                    ^
```

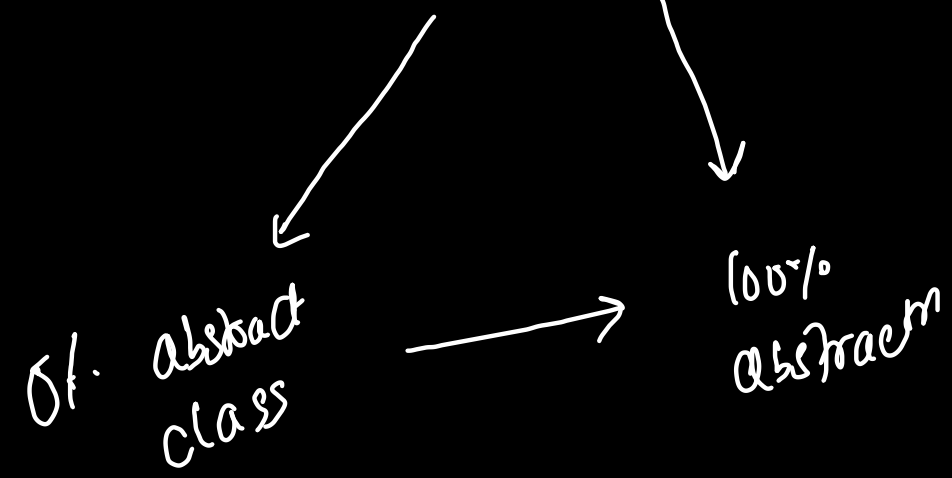
```java
static abstract class Stack{
    public abstract void addLast(int data);
    public abstract int removeLast();
}
```

```java
static class Deque extends Stack{
    static ArrayDeque<Integer> q = new ArrayDeque<>();

    public void addLast(int data){
        System.out.println("Add Last of Deque");
        q.addLast(data);
    }
    public void addFirst(int data){
        System.out.println("Add First of Deque");
        q.addFirst(data);
    }
```

```java
public static void main(String[] args){
    Deque q = new Deque();
    q.removeLast();
}
```

```
Finished in N/A
Line 23: error: Deque is not abstract and does not override
abstract method removeLast() in Stack [in Main.java]
    static class Deque extends Stack{
           ^
```

Abstract class



1 abstract method (no body) → mandatory → class should be abstract !!

```java
static abstract class Stack{
    public abstract void addLast(int data);
    public abstract int removeLast();
}
```

```java
public static void main(String[] args){
    Stack stk = new Deque();
    stk.addLast(10);
    stk.removeLast();
}
```

```
Finished in 74 ms
Add Last of Deque
Remove Last of Deque
```

```java
static class Deque extends Stack{
    static ArrayDeque<Integer> q = new ArrayDeque<>();

    public void addLast(int data){
        System.out.println("Add Last of Deque");
        q.addLast(data);
    }
    public void addFirst(int data){
        System.out.println("Add First of Deque");
        q.addFirst(data);
    }

    @Override
    public int removeLast(){
        System.out.println("Remove Last of Deque");
        return q.removeLast();
    }

    public int removeFirst(){
        System.out.println("Remove First of Deque");
        return q.removeFirst();
    }
}
```

```java
static abstract class Stack{
    int nonStaticData = 0;
    Stack(){
        System.out.println("Abstract Class Constructor");
        nonStaticData = 100;
        System.out.println(this.nonStaticData);
    }
    public abstract void addLast(int data);
    public abstract int removeLast();
}
```

```
Finished in 89 ms
Abstract Class Constructor
100
Add Last of Deque
Remove Last of Deque
```

Abstract class will have

↓

1) Constructors Yes

2) Non static data (Object Data members)

3) This pointer

Constructor — No

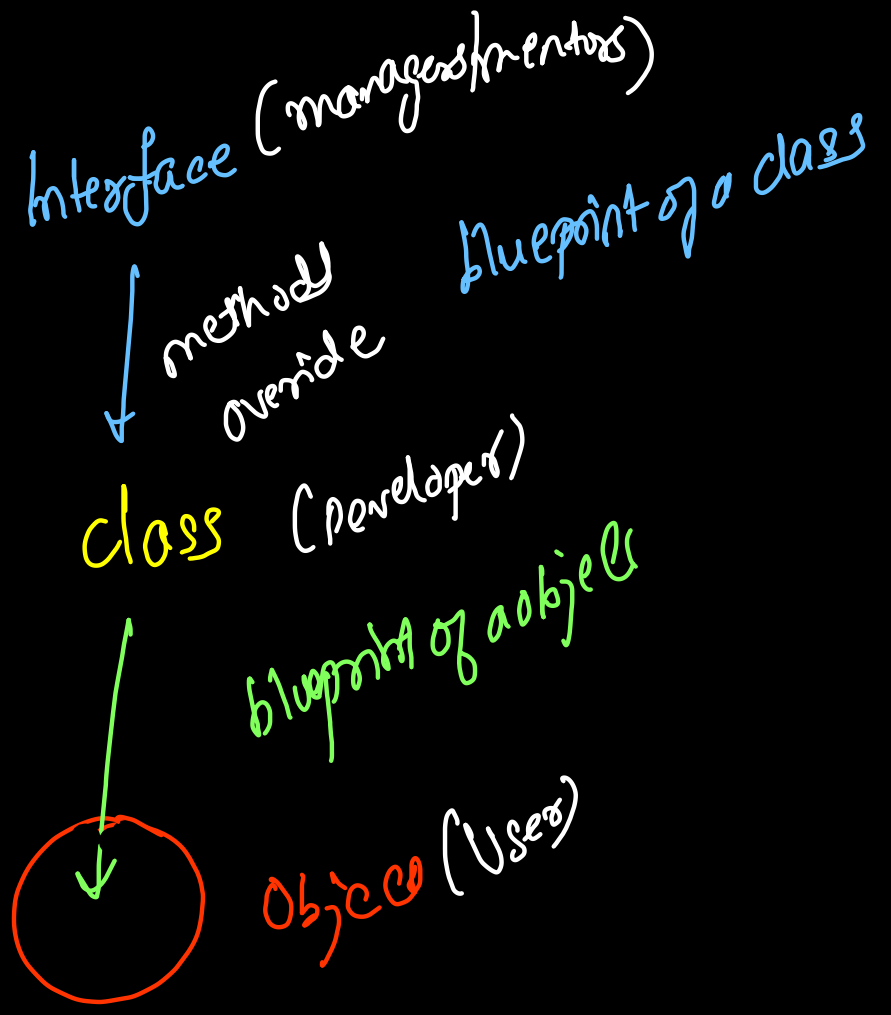
1) super()
2) override

Static methods — No

abstract

Static methods cannot be overrided.

& Abstract method must be overrided



Interface

1. not in class hierarchy
2. Constructor ✗
3. non static data ✗
4. this ✗
5. multiple interfaces can be implemented

```java
static interface Stack{
    void addLast(int data);
    int removeLast();
}

static interface Queue{
    void addFirst(int data);
    int removeLast();
}
```

```java
public static void main(String[] args){
    Stack stk = new Deque();
    stk.addLast(10);
    stk.removeLast();

    Queue q = new Deque();
    q.addFirst(10);
    q.removeLast();
}
```

```java
static class Deque implements Stack, Queue{
    ArrayDeque<Integer> q = new ArrayDeque<>();

    public void addLast(int data){
        System.out.println("Add Last of Deque");
        q.addLast(data);
    }
    public void addFirst(int data){
        System.out.println("Add First of Deque");
        q.addFirst(data);
    }
    public int removeLast(){
        System.out.println("Remove Last of Deque");
        return q.removeLast();
    }
    public int removeFirst(){
        System.out.println("Remove First of Deque");
        return q.removeFirst();
    }
}
```

```
Finished in 94 ms
Add Last of Deque
Remove Last of Deque
Add First of Deque
Remove Last of Deque
```

final keyword

1. Data Members → constant (can be static as well as non-static)
2. Reference Variables → Reassignment not allowed, data can be changed
3. Classes → cannot be extended / inherited! = all final methods!
4. methods → cannot be overided! = early Binded methods!

① Change in return type

```java
class Parent {
    int parentData;

    public void getObject() {
        System.out.println(x: "Parent getObject");
        // return this;
    }
}

class Child extends Parent {
    int childData;

    public int getObject() {
        System.out.println(x: "Overrided");
        return 0;
    }
}
```

same return type

compiler error ⇒ Ambiguous function!

Overriding is not possible with different return type!

Covariant type!

```java
class Parent {
    int parentData;

    public Parent getObject() {
        System.out.println(x: "Parent getObject ");
        return this;
    }
}

class Child extends Parent {
    int childData;

    public Child getObject() {
        System.out.println(x: "Child getObject ");
        return this;
    }
}

public class App {
    Run | Debug
    public static void main(String[] args) throws Exception {
        Parent obj1 = new Parent();
        System.out.println(obj1.getObject());

        Child obj2 = new Child();
        System.out.println(obj2.getObject());

        Parent obj3 = new Child();
        System.out.println(obj3.getObject());
    }
}
```

Overriding

↳ Parent getObject

↳ Child getObject

↳ Child getObject

```java
interface IParent {
    public IParent getObject();
}

class Child implements IParent {
    int childData;

    public Child getObject() {
        System.out.println(x: "Child getObject ");
        return this;
    }
}

public class App {
    Run | Debug
    public static void main(String[] args) throws Exception {
        IParent obj = new Child();
        System.out.println(obj.getObject());
    }
}
```

Interface Returning an Object!

② change in parameters

```java
class Parent {
    int parentData;

    public void getObject() {
        System.out.println(x: "Parent getObject");
    }
}

class Child extends Parent {
    int childData;

    public void getObject(int para) {
        System.out.println(x: "Overrided");
    }
}

public class App {
    Run | Debug
    public static void main(String[] args) throws Exception {
        Parent obj = new Child();
        obj.getObject();
    }
}
```

not overrided !

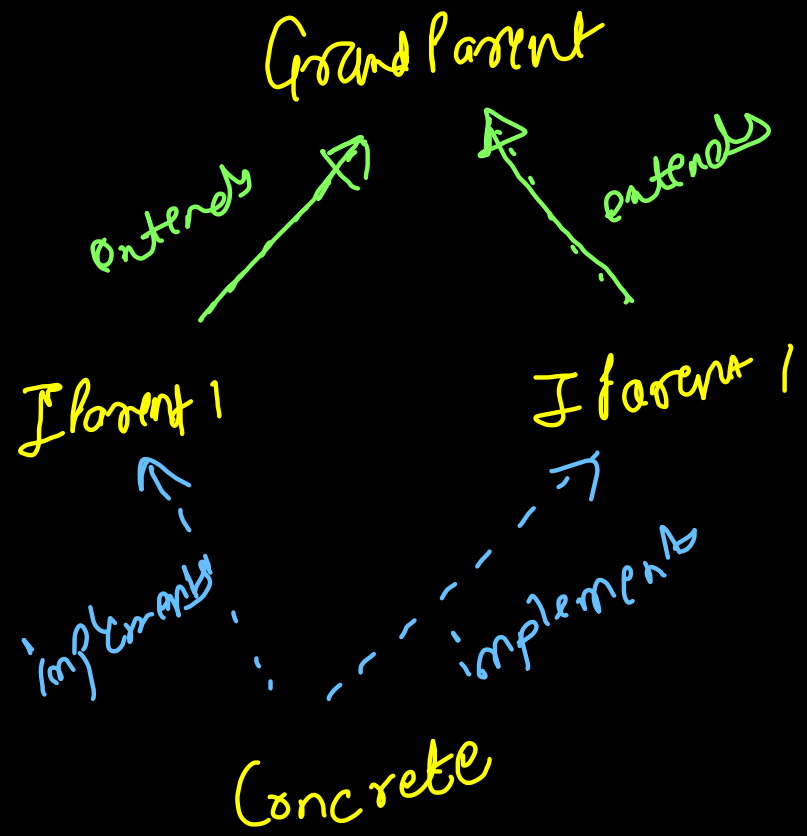
```
architaggarwal@Archits-MacBook-Air Java OOPS % javac BookMyShow/App.java
architaggarwal@Archits-MacBook-Air Java OOPS % java BookMyShow.App
Parent getObject
```

```java
class Parent {
    int parentData;

    public void getObject() {
        System.out.println(x: "Parent getObject");
    }
}

class Child extends Parent {
    int childData;

    public void getObject(int para) {
        System.out.println(x: "Overrided");
    }
}

public class App {
    Run | Debug
    public static void main(String[] args) throws Exception {
        Parent obj = new Child();
        obj.getObject(5);
    }
}
```

not overloaded also

Compilation error

```java
interface IGrandParent {
    public void ambiguous();
}

interface IParent1 extends IGrandParent {
    public void a();
}

interface IParent2 extends IGrandParent {
    public void b();
}

class Concrete implements IParent1, IParent2 {
    public void a() {
        System.out.println(x: "A Overrided");
    };

    public void b() {
        System.out.println(x: "B Overrided");
    };

    public void ambiguous() {
        System.out.println(x: "Ambiguous Overrided");
    };
}
```



Diamond Problem Solved

```java
interface IParent {
    default void concreteMethod() {
        System.out.println(x: "Hello World");
    }
}

class Child implements IParent {
}

public class App {
    Run | Debug
    public static void main(String[] args) throws Exception {
        IParent obj = new Child();
        obj.concreteMethod();
    }
}
```

Show Code Actions (⌘.)

Java 9 Interface

↳ Concrete method

↓

default

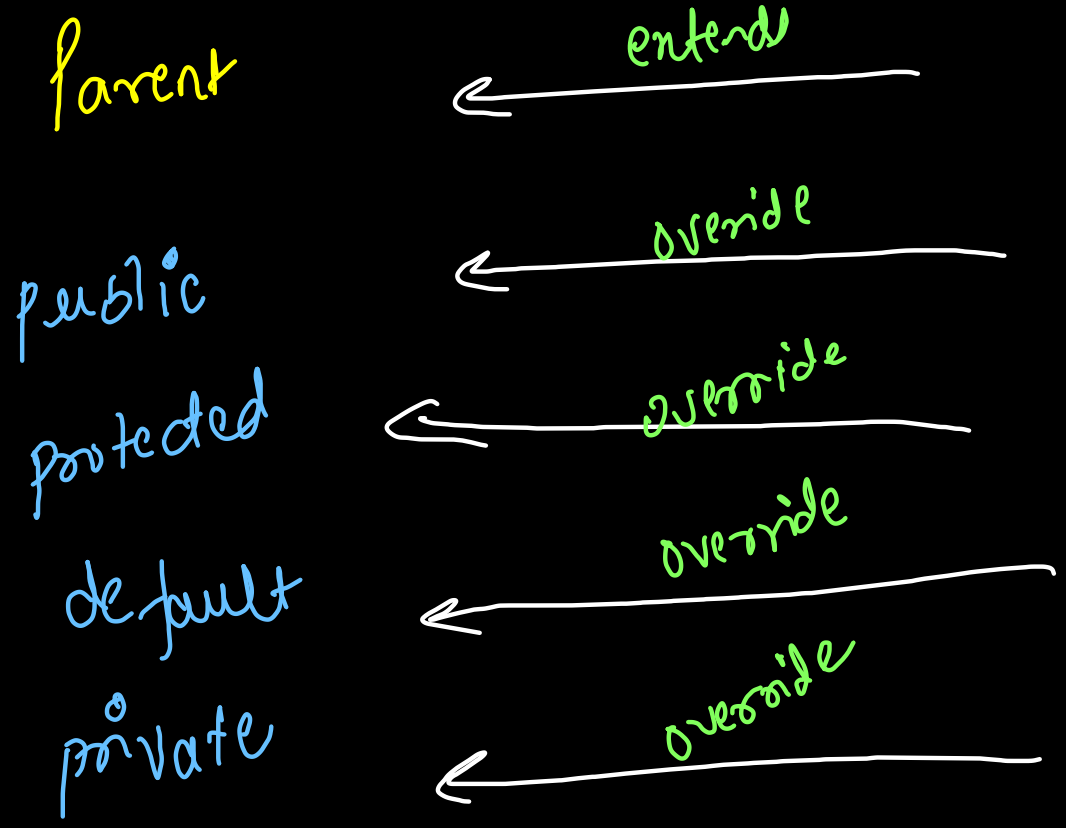
by default methods in interface are abstract !!

```
architaggarwal@Archits-MacBook-Air Java OOPS % javac BookMyShow/App.java
architaggarwal@Archits-MacBook-Air Java OOPS % java BookMyShow.App
Hello World
```

```java
interface IParent {
    int staticProp = 100;

    static void concreteMethod() {
        System.out.println("Hello World" + staticProp);
    }
}

class Child implements IParent {
}

public class App {
    Run | Debug
    public static void main(String[] args) throws Exception {
        // IParent obj = new Child();
        IParent.concreteMethod();
    }
}
```

Static concrete methods in interfaces are possible in Java 9+.

Inheritance Access modifiers



public protected default private

public, protected default private

public, protected, default private

public, protected, default, private

Upcasting vs Downcasting

User

↑ upcasting (implicit)

RegisteredUser

RUser is a User

Child class is a superclass

User obj = new RegisteredUser();

User: Parent

RegisteredUser(): Child

User

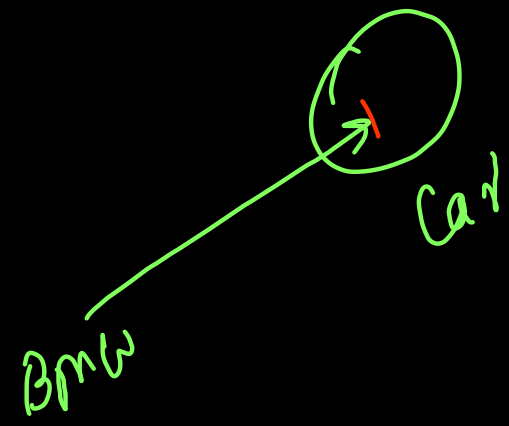
downcasting

RUser

RUser obj = new User();

child

parent



instance of

You can do downcasting explicitly by confirming the object type.

```java
import BookMyShow.users.GuestUser;
import BookMyShow.users.RegisteredUser;

public class App {
    Run | Debug
    public static void main(String[] args) throws Exception {
        GuestUser obj1 = new GuestUser();
        RegisteredUser obj2 = new RegisteredUser();
        GuestUser obj3 = new RegisteredUser();

        RegisteredUser obj4 = obj1;
        RegisteredUser obj5 = obj2;
        RegisteredUser obj6 = obj3;
    }
}
```

Not Allowed Implicitly and it is wrong

shallow copy no problem

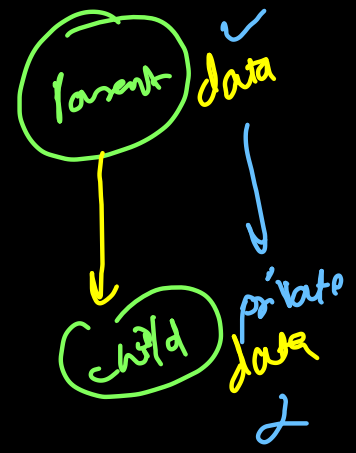
implicitly not allowed, but it should be allowed explicitly! (compilation Error)

```java
GuestUser obj1 = new GuestUser();
RegisteredUser obj4 = (RegisteredUser) obj1;
```

} Runtime Error
↳ classCast Exception!

```java
public static void main(String[] args) throws Exception {
    GuestUser obj1 = new GuestUser();
    if (obj1 instanceof RegisteredUser) {
        RegisteredUser obj4 = (RegisteredUser) obj1;
        System.out.println(x: "Obj1 is Now Registered");
    } else {
        System.out.println(x: "Obj1 is not Registered");
    }

    RegisteredUser obj2 = new RegisteredUser();
    RegisteredUser obj5 = obj2;
    System.out.println(x: "This is directly allowed implicitly");

    GuestUser obj3 = new RegisteredUser();

    if (obj3 instanceof RegisteredUser) {
        RegisteredUser obj6 = (RegisteredUser) obj3;
        System.out.println(x: "Obj3 is Now Registered but you can downcast explicitly");
    } else {
        System.out.println(x: "Obj3 is not Registered");
    }

}
```

Be sure that User is registered
⇒ User is a Registered User (Downcasting) explicitly



Generics in Java

↳ Parameterized Types → Reusability & Generalization
→ Type Safety (no need of typecasting)

① Inbuilt Arraylist
- ↳ of Generic Type (Object type)
- ↳ of Specific Type

② Custom Arraylist → Q) why collection framework of generic is not possible?
- ↳ of Integer Type
- ↳ of Generic Type (using Object)

Generics using object class

```java
Object[] arr = new Object[10];

arr[0] = new Integer(value: 10); // Integer Wrapper Class
arr[1] = 10; // autoboxing

arr[2] = new StringBuilder(str: "Hello"); // StringBuilder
arr[3] = "Hello"; // String

arr[4] = new Character(value: 'A');
arr[5] = 'A';

for (int i = 0; i < 10; i++) {
    System.out.print(arr[i] + " ");
}

((StringBuilder) arr[2]).append(str: " World");

for (int i = 0; i < 10; i++) {
    System.out.print(arr[i] + " ");
}

((StringBuilder) arr[0]).append(str: " World"); // Run-time error
```

Type checking

```java
for (int i = 0; i < 10; i++) {
    System.out.print(arr[i] + " ");
}

if (arr[2] instanceof StringBuilder)
    ((StringBuilder) arr[2]).append(str: " World");

for (int i = 0; i < 10; i++) {
    System.out.print(arr[i] + " ");
}

if (arr[0] instanceof StringBuilder)
    ((StringBuilder) arr[0]).append(str: " World");

for (int i = 0; i < 10; i++) {
    System.out.print(arr[i] + " ");
}
```

```java
public static void main(String[] args) {
    ArrayList<Integer> arr = new ArrayList<>();
    arr.add(e: 10);
    arr.add(e: 20);
    arr.add(-10);
    // arr.add("Hello");→ Compilation error

    ArrayList arr2 = new ArrayList();
    arr2.add(e: 10);
    arr2.add(e: 5.5);
    arr2.add(new StringBuilder(str: "Hello"));
    arr2.add(new RegisteredUser());

    ((StringBuilder) arr2.get(index: 2)).append(str: " World");

    if ((arr2.get(index: 3)) instanceof StringBuilder)
        ((StringBuilder) arr2.get(index: 3)).append(str: " World");
    System.out.println(arr2);
```

Parameter Type = Integer

Warning

Unbounded Type = Object type

```java
class Box<T> {
    private T data;

    public Box(T data) {
        this.data = data;
    }

    public T getData() {
        return data;
    }

    public void setData(T data) {
        this.data = data;
    }
}
```

```java
Run | Debug
public static void main(String[] args) {
    Box<Integer> obj = new Box<>(data: 5);
    System.out.println(obj.getData());

    Box<Double> obj2 = new Box<>(data: 3.14);
    System.out.println(obj2.getData());

    Box<String> obj3 = new Box<>(data: "Hello");
    System.out.println(obj3.getData());

    Box<Character> obj4 = new Box<>(data: 'A');
    System.out.println(obj4.getData());

}
```

bounded
T will be replaced by parameter type!

```java
Box obj5 = new Box(data: 4.5); // Object
System.out.println(obj5.getData());

Box obj6 = new Box(new StringBuilder(str: "Hello"));
System.out.println(obj6.getData());
```

unbounded
↓
T is replaced by Object

Example

Non Generic Box Class

```
public class Box {
    private Object object;

    public void set(Object object) { this.object = object; }
    public Object get() { return object; }
}
```

Generic Box class

```
/**
 * Generic version of the Box class.
 * @param <T> the type of the value being boxed
 */
public class Box<T> {
    // T stands for "Type"
    private T t;

    public void set(T t) { this.t = t; }
    public T get() { return t; }
}
```

Instantiation of Generic Type class

Unbounded type or raw type

1) `Box obj = new Box();` → Create Generic type as "object"

bounded type

2) `Box<Integer> obj = new Box<Integer>();` → redundant

↳ takes generic type as Integer

↳ If constructor can get object type during compile time

diamond syntax

3) `Box<Integer> obj = new Box<>();`

↳ constructor invokation

↳ diamond

Multiple Type Parameters

```
public interface Pair<K, V> {
    public K getKey();
    public V getValue();
}

public class OrderedPair<K, V> implements Pair<K, V> {

    private K key;
    private V value;

    public OrderedPair(K key, V value) {
        this.key = key;
        this.value = value;
    }

    public K getKey()   { return key; }
    public V getValue() { return value; }
}
```

Object Creatn

```
Pair<String, Integer> p
        = new OrderedPair<>();
```

Parameterized type as type parameter

```
Pair<String, ArrayList<Integer>>
    p = new OrderedPair<>();
```

```java
class MyHashMap<K, V> {
    K key;
    V value;

    public MyHashMap(K key, V value) {
        this.key = key;
        this.value = value;
    }

    public K getKey() {
        return key;
    }

    public void setKey(K key) {
        this.key = key;
    }

    public V getValue() {
        return value;
    }

    public void setValue(V value) {
        this.value = value;
    }

}
```

```java
Run | Debug
public static void main(String[] args) {
    MyHashMap ipl1 = new MyHashMap(key: "Delhi", value: 0);
    MyHashMap ipl2 = new MyHashMap(key: "Mumbai", value: 5);
    MyHashMap ipl3 = new MyHashMap(key: 10, value: 5.5);

    MyHashMap<String, Integer> ipl4 = new MyHashMap<>(key: "Delhi", value: 0);
    MyHashMap<String, Integer> ipl5 = new MyHashMap<>(key: "Mumbai", value: 5);
    // MyHashMap<String, Integer> ipl6 = new MyHashMap<>(10, 5.5); // compilation
    // error
}
```

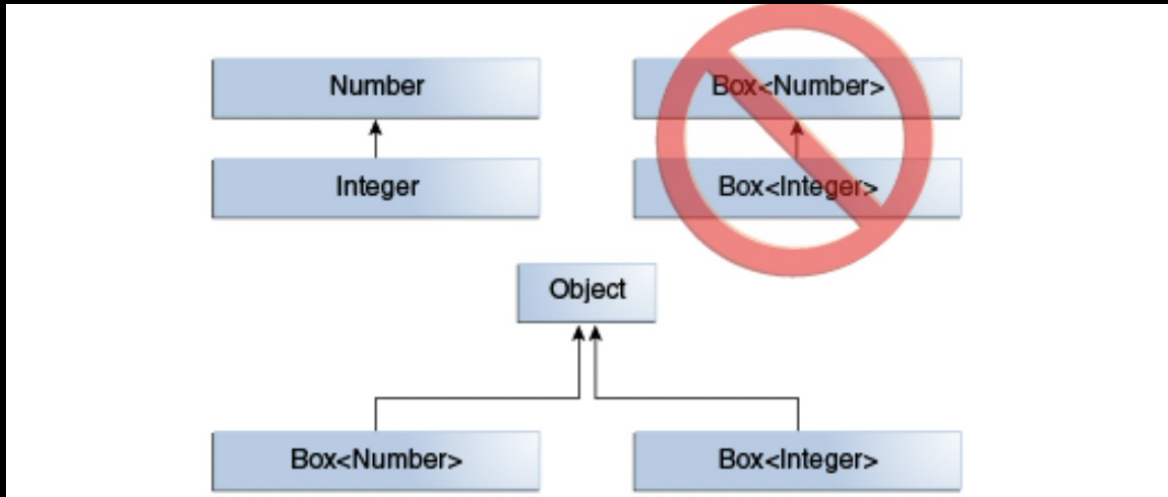
Bounded Type Parameters

```
public class Box<T> {

    private T t;

    public void set(T t) {
        this.t = t;
    }

    public T get() {
        return t;
    }

    public <U extends Number> void inspect(U u){
        System.out.println("T: " + t.getClass().getName());
        System.out.println("U: " + u.getClass().getName());
    }

    public static void main(String[] args) {
        Box<Integer> integerBox = new Box<Integer>();
        integerBox.set(new Integer(10));
        integerBox.inspect("some text"); // error: this is still String!
    }
}
```

```
class Pair<K extends Number, V> {
    K key;
    V value;

    public Pair(K key, V value) {
        this.key = key;
        this.value = value;
    }

    public K getKey() {
        return key;
    }

    public void setKey(K key) {
        this.key = key;
    }

    public V getValue() {
        return value;
    }

    public void setValue(V value) {
        this.value = value;
    }

}
```

```
Pair<Integer, Object> obj1 = new Pair<>(key: 5, value: 10);
Pair<Double, Integer> obj2 = new Pair<>(key: 5.5, value: 10);
// Pair<String, Integer> obj3 = new Pair<>("Hello", 15); // Not Possible
```

Generics & Inheritance



`Box<Integer>` is not a subtype of `Box<Number>` even though `Integer` is a subtype of `Number`.

eg:

```
Box<Number> b = new Box<>();
b.add(new Integer(10));   ✓
b.add(new Double(5.5));   ✓
```

```
Box<Integer> b = new Box<>();
b.add(new Object());      ✗
b.add(new Integer(10));   ✓
b.add(new Double(5.5));   ✗
```

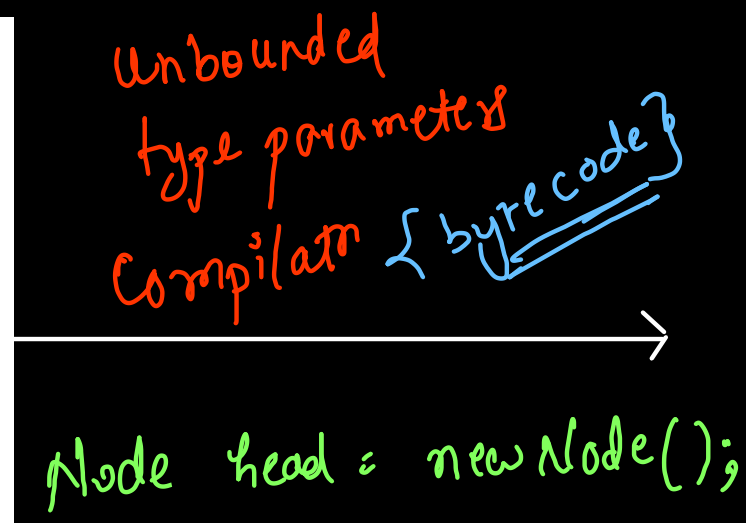
Type Erasure

Generics were introduced to the Java language to provide tighter type checks at compile time and to support generic programming. To implement generics, the Java compiler applies type erasure to:

- Replace all type parameters in generic types with their bounds or `Object` if the type parameters are unbounded. The produced bytecode, therefore, contains only ordinary classes, interfaces, and methods.
- Insert type casts if necessary to preserve type safety.
- Generate bridge methods to preserve polymorphism in extended generic types.

Type erasure ensures that no new classes are created for parameterized types; consequently, generics incur no runtime overhead.

```
public class Node<T> {

    private T data;
    private Node<T> next;

    public Node(T data, Node<T> next) {
        this.data = data;
        this.next = next;
    }

    public T getData() { return data; }
    // ...
}
```




```
public class Node {

    private Object data;
    private Node next;

    public Node(Object data, Node next) {
        this.data = data;
        this.next = next;
    }

    public Object getData() { return data; }
    // ...
}
```

Bridge Methods in Type Erasure

```
public class Node {

    public Object data;

    public Node(Object data) { this.data = data; }

    public void setData(Object data) {
        System.out.println("Node.setData");
        this.data = data;
    }
}

public class MyNode extends Node {

    public MyNode(Integer data) { super(data); }

    public void setData(Integer data) {
        System.out.println("MyNode.setData");
        super.setData(data);
    }
}
```

For the `MyNode` class, the compiler generates the following bridge method for `setData`:

```
class MyNode extends Node {

    // Bridge method generated by the compiler
    //
    public void setData(Object data) {
        setData((Integer) data);
    }

    public void setData(Integer data) {
        System.out.println("MyNode.setData");
        super.setData(data);
    }

    // ...
}
```

Restrictions on Generics #1

Cannot Instantiate Generic Types with Primitive Types

Consider the following parameterized type:

```
class Pair<K, V> {

    private K key;
    private V value;

    public Pair(K key, V value) {
        this.key = key;
        this.value = value;
    }

    // ...
}
```

When creating a `Pair` object, you cannot substitute a primitive type for the type parameter `K` or `V`:

```
Pair<int, char> p = new Pair<>(8, 'a');  // compile-time error
```

You can substitute only non-primitive types for the type parameters `K` and `V`:

```
Pair<Integer, Character> p = new Pair<>(8, 'a');
```

Note that the Java compiler autoboxes `8` to `Integer.valueOf(8)` and 'a' to `Character('a')`:

```
Pair<Integer, Character> p = new Pair<>(Integer.valueOf(8), new Character('a'));
```

Primitives as generics are not possible!

ArrayList<int> a

= new ArrayList<>();

a.add(10);

↳ Integer/wrapper class

Restrictions on Generics #2

Cannot Create Instances of Type Parameters

You cannot create an instance of a type parameter. For example, the following code causes a compile-time error:

```
public static <E> void append(List<E> list) {
    E elem = new E();  // compile-time error
    list.add(elem);
}
```

As a workaround, you can create an object of a type parameter through reflection:

```
public static <E> void append(List<E> list, Class<E> cls) throws Exception {
    E elem = cls.newInstance();   // OK
    list.add(elem);
}
```

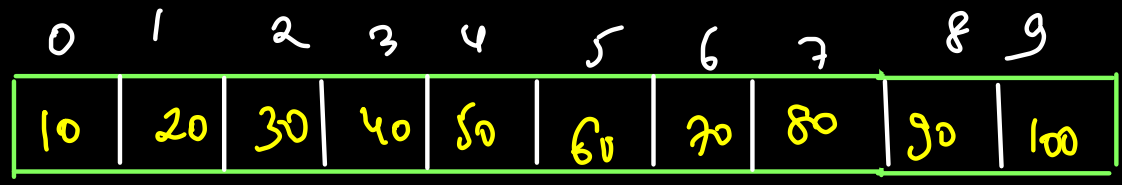
You can invoke the `append` method as follows:

```
List<String> ls = new ArrayList<>();
append(ls, String.class);
```

```
class Box<T> {
    private T data;

    public Box(){
        this.data = new T();
    }
```

Cannot create object of generic type ⇒ Compile time error !

Custom ArrayList



↓ cap + cap/2 = 10 + 10/2 = 15

↓ cap + cap/2 = 15 + 15/2 = 23

10 - ---- 150 16 0 0 0 0 0 0

CAPACITY = ~~10~~ ~~15~~ 23

size = ~~0 1 2 3 - - 9~~ 10 11 ~~12 13 14 15~~ 16

```
ArrayList<Integer>
    arr = new ArrayList<>();
for(int i=10; i<=1000; i=i+10){
    arr.add(i);
}
for(int i=
```

```java
class MyArrayList {
    private int[] data = new int[10];
    private int capacity = 10;
    private int size = 0;

    public int get(int idx) {
        return data[idx];
    }

    public void set(int val, int idx) {
        data[idx] = val;
    }

    public void add(int val) {
        if (size == capacity) {
            capacity = capacity + capacity / 2;
            int[] copy = new int[capacity];
            for (int idx = 0; idx < data.length; idx++) {
                copy[idx] = data[idx];
            }
            data = copy;
        }

        data[size] = val;
        size++;
    }
```

O(1)

O(1)

asymptotic O(N) worst

O(1) best asymptotic

Amortized (average) ⇒ O(1)

```java
public int remove() {
    int oldVal = data[size];
    data[size] = 0;
    size--;
    return oldVal;
}

@Override
public String toString() {
    StringBuilder res = new StringBuilder(str: "[");
    for (int val : data) {
        res.append(val + ",");
    }
    res.setCharAt(res.length() - 1, ch: ']');
    return res.toString();
}
```

O(1)

O(N)

```java
public static void CustomArrayList() {
    ArrayList<Integer> obj = new ArrayList<>();
    obj.add(e: 10);
    obj.add(e: 20);
    obj.add(e: 30);
    System.out.println(obj);

    MyArrayList obj2 = new MyArrayList();
    obj2.add(val: 10);
    obj2.add(val: 20);
    obj2.add(val: 30);
    System.out.println(obj2);
}
```

[10,20,30]

[10,20,30,0,0,0 ... 0]

```java
class MyArrayList<T> {
    private Object[] data = new Object[10];
    private int capacity = 10;
    private int size = 0;

    public T get(int idx) {
        return (T) data[idx];
    }

    public void set(T val, int idx) {
        data[idx] = val;
    }

    public void add(T val) {
        if (size == capacity) {
            capacity = capacity + capacity / 2;
            Object[] copy = new Object[capacity];
            for (int idx = 0; idx < data.length; idx++) {
                copy[idx] = data[idx];
            }
            data = copy;
        }

        data[size] = val; // no typecasting T is a Object (upcasting)
        size++;
    }
```

```java
    public T remove() {
        T oldVal = (T) data[size];
        // Typecasting Object is not always T (downcasting)
        data[size] = null;
        size--;
        return oldVal;
    }

    @Override
    public String toString() {
        StringBuilder res = new StringBuilder(str: "[");
        for (Object val : data) {
            res.append(val + ",");
        }
        res.setCharAt(res.length() - 1, ch: ']');
        return res.toString();
    }
}
```

Unbounded

```java
MyArrayList obj3 = new MyArrayList();
obj3.add(val: 10);
obj3.add(val: "Hello");
obj3.add(val: 5.5);
System.out.println(obj3);
```

↳ [10, "Hello", 5.5, N, N, N, ... N]

bounded

```java
MyArrayList<Integer> obj4 = new MyArrayList<>();
obj4.add(val: 10);
obj4.add(val: 20);
obj4.add(val: 30);
// obj4.add("ehl"); // not possible
System.out.println(obj4);
```

↓

[10, 20, 30, N, N, N ... N]

Restrictions on Generics #3

Cannot Declare Static Fields Whose Types are Type Parameters

A class's static field is a class-level variable shared by all non-static objects of the class. Hence, static fields of type parameters are not allowed. Consider the following class:

```
public class MobileDevice<T> {
    private static T os;

    // ...
}
```

If static fields of type parameters were allowed, then the following code would be confused:

```
MobileDevice<Smartphone> phone = new MobileDevice<>();
MobileDevice<Pager> pager = new MobileDevice<>();
MobileDevice<TabletPC> pc = new MobileDevice<>();
```

Because the static field `os` is shared by `phone`, `pager`, and `pc`, what is the actual type of `os`? It cannot be `Smartphone`, `Pager`, and `TabletPC` at the same time. You cannot, therefore, create static fields of type parameters.

Restrictions on Generics #4

Cannot Create Arrays of Parameterized Types

You cannot create arrays of parameterized types. For example, the following code does not compile:

```
List<Integer>[] arrayOfLists = new List<Integer>[2];  // compile-time error
```

The following code illustrates what happens when different types are inserted into an array:

```
Object[] strings = new String[2];
strings[0] = "hi";   // OK
strings[1] = 100;    // An ArrayStoreException is thrown.
```

If you try the same thing with a generic list, there would be a problem:

```
Object[] stringLists = new List<String>[2];  // compiler error, but pretend it's allowed
stringLists[0] = new ArrayList<String>();   // OK
stringLists[1] = new ArrayList<Integer>();  // An ArrayStoreException should be thrown,
                                            // but the runtime can't detect it.
```

If arrays of parameterized lists were allowed, the previous code would fail to throw the desired `ArrayStoreException`.

Restrictions on Generics #5

Cannot Overload a Method Where the Formal Parameter Types of Each Overload Erase to the Same Raw Type

A class cannot have two overloaded methods that will have the same signature after type erasure.

```
public class Example {
    public void print(Set<String> strSet) { }
    public void print(Set<Integer> intSet) { }
}
```

The overloads would all share the same classfile representation and will generate a compile-time error.

Lambda Expression

- One-line or inline functions
  - code will be a one line anonymous function
    - no name, no modifier, no return type, no datatypes of parameter
- Benefits :→ (1) reduce code lines, makes code readable/maintainable/concise
  (2) Sequential & parallel execution support → passing functions as parameters
  (3) calls API very efficiently

→ Inbuilt Functions using Lambda Expression ⇒ forEach method

→ Custom Lamba Expression
- Priority Queue
  - Comparable
  - Comparator
- Arrays.sort or Collections.sort
  - Reverse Order
  - Custom order (Pairs)

```
public String concatenate(String a, String b) {
    return a+b;
}
```

⇓ Type Infer at Compile time

```
(a, b) → {
    return a+b;
}
```

```
Run | Debug
public static void main(String[] args) {
    int[] arr = { 10, 20, 30, 40, 50 };

    // FOR EACH LOOP (Iterable)
    for (int val : arr) {
        System.out.print(val + " ");
    }
    System.out.println();

    // FOR EACH METHOD
    ArrayList<Integer> al = new ArrayList<>();
    al.add(e: 10);
    al.add(e: 20);
    al.add(e: 30);
    al.add(e: 40);

    al.forEach((val) -> System.out.print(val + " "));
    System.out.println();
}
```

anonymous function or arrow func = lambda Expression

Syntax ②

Curly braces can be omitted only if one line instruction is there

Syntax ①

```
al.forEach((val) -> {
    System.out.print(val + " ");
});
System.out.println();
```

```
al.forEach((val) -> {
    if (val % 2 == 0)
        System.out.print(s: "Even ");
    else
        System.out.print(s: "Odd ");
});
```

Comparator

(Binary)

```
public int compare(Object a, Object b){
    ① return a.val - b.val;
    ② return b.val - a.val
}
```

① `a.val - b.val` → a-b = (-ve) → smaller will be placed first → ⇒ Increasing order or min Heap

② `b.val - a.val` → b-a = (+ve) → larger will be placed first → ⇒ Decreasing order or max Heap

- Same class whose objects to be compared

Comparable

(By default sorting)

(Unique)

```
public int compareTo(Object other){
    ① return this.val - other.val;
    ② return other.val - this.val;
}
```

① → Increasing order

② → Decreasing order

Arrays.sort

```java
public static void customLambdaExpression() {
    int[] arr = { 50, 30, 80, 90, 10, 20, 70, 40, 100, 60 };
    Arrays.sort(arr);
    // Arrays.sort(arr, comparator); INVALID FOR PRIMITIVES

    // Increasing Order : Default
    // System.out.println(arr); // HashCode

    for (int val : arr)
        System.out.print(val + " ");
    System.out.println();

    Integer[] copy = new Integer[arr.length];
    for (int idx = 0; idx < arr.length; idx++)
        copy[idx] = arr[idx];

    Arrays.sort(copy, (a, b) -> a - b); // Increasing Order
    for (int val : copy)
        System.out.print(val + " ");
    System.out.println();

    Arrays.sort(copy, (a, b) -> b - a); // Decreasing Order
    for (int val : copy)
        System.out.print(val + " ");
    System.out.println();
}
```

Integer

Integer

Approach (1) Custom Lambda Expression

Marks of PCM (GFG)

```java
class Student{
    int phy, chem, maths;
    Student(int phy, int chem, int maths){
        this.phy = phy;
        this.chem = chem;
        this.maths = maths;
    }
}
public void customSort (int phy[], int chem[], int math[], int N)
{
    Student[] stud = new Student[N];
    for(int idx = 0; idx < N; idx++){
        stud[idx] = new Student(phy[idx], chem[idx], math[idx]);
    }

    Arrays.sort(stud, (a, b) -> {
        if(a.phy != b.phy) return a.phy - b.phy;
        // Increasing order of physics
        if(a.chem != b.chem) return b.chem - a.chem;
        // Same in Physics, Decreasing Order of Chemisty
        return a.maths - b.maths;
        // Same in Phy & Chem, Increasing order in maths
    });

    for(int idx = 0; idx < N; idx++){
        phy[idx] = stud[idx].phy;
        chem[idx] = stud[idx].chem;
        math[idx] = stud[idx].maths;
    }
}
```

Students Type

```java
String[] names = { "Guneet", "Vrushabh", "Chinmay", "Raghav", "Hardik", "Archit" };
Arrays.sort(names);
for (String val : names)
    System.out.print(val + " ");
System.out.println();
```

Archit < Chinmay < Guneet < Hardik < Raghav < Vrushabh → Lexicographically increasing order

Lexicographical:

- Chinmay < Chirag
- Chin < Chinmay

"g21211111" < "g2129"

"g212" < "g21200000"

largest no from Array

52 90 6 92 5 97 50 9 59 9927

"9 97 92 9927 90 6 59 5 52 50"

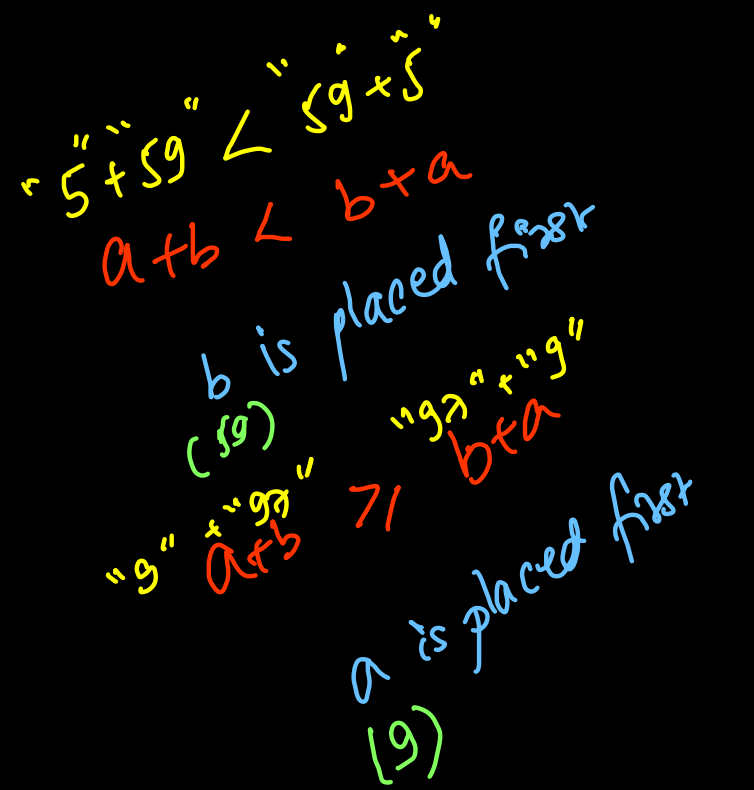
9297

5952

9 + 97 > 97 + 9

59 + 5 > 5 + 59

```java
public String largestNumber(int[] arr) {
    String[] copy = new String[arr.length];
    boolean allZero = true;
    for(int idx = 0; idx < arr.length; idx++)
    {
        if(arr[idx] != 0) allZero = false;
        copy[idx] = Integer.toString(arr[idx]);
    }

    if(allZero == true) return "0";

    Arrays.sort(copy, (a, b) -> {
        if((a + b).compareTo(b + a) < 0) return +1;
        return -1;
    });

    StringBuilder res = new StringBuilder("");
    for(String val: copy) res.append(val);
    return res.toString();
}
```



Time → O(n log r * l)

Space → O(n)

```java
class Movie{
    int duration;
    double ratings;
    String name;

    public Movie(int duration, double ratings, String name) {
        this.duration = duration;
        this.ratings = ratings;
        this.name = name;
    }
}
```

Not possible

```java
public static void comparableVSComparator() {
    Movie[] arr = new Movie[5];

    arr[0] = new Movie(duration: 180, ratings: 4.5, name: "Avengers");
    arr[1] = new Movie(duration: 150, ratings: 5.0, name: "Titanic");
    arr[2] = new Movie(duration: 100, ratings: 3.0, name: "Spiderman");
    arr[3] = new Movie(duration: 200, ratings: 5.0, name: "Avatar");
    arr[4] = new Movie(duration: 50, ratings: 1.0, name: "Thor");

    Arrays.sort(arr);
}
```

Runtime Error : How to compare 2 movie objects?

Approach #2: Using Comparable

```java
class Movie implements Comparable<Movie> {
    int duration;
    double ratings;
    String name;

    public Movie(int duration, double ratings, String name) {
        this.duration = duration;
        this.ratings = ratings;
        this.name = name;
    }

    public int compareTo(Movie other) {
        return this.duration - other.duration;
        // Default Sorting: Based on Increasing Order of Duration
    }

    public String toString() {
        return "Name : " + this.name + " of " + this.duration
                + " Minutes " + " with " + ratings + " ratings";
    }
}
```

```java
Movie[] arr = new Movie[5];

arr[0] = new Movie(duration: 180, ratings: 4.5, name: "Avengers");
arr[1] = new Movie(duration: 150, ratings: 5.0, name: "Titanic");
arr[2] = new Movie(duration: 100, ratings: 3.0, name: "Spiderman");
arr[3] = new Movie(duration: 200, ratings: 5.0, name: "Avatar");
arr[4] = new Movie(duration: 50, ratings: 1.0, name: "Thor");

Arrays.sort(arr);

for (Movie val : arr)
    System.out.println(val);
```

```
architaggarwal@Archits-MacBook-Air Java OOPS % java OOPS_Codes.LambdaExpression
Name : Thor of 50 Minutes  with 1.0 ratings
Name : Spiderman of 100 Minutes  with 3.0 ratings
Name : Titanic of 150 Minutes  with 5.0 ratings
Name : Avengers of 180 Minutes  with 4.5 ratings
Name : Avatar of 200 Minutes  with 5.0 ratings
```

```java
class MovieDurationIncreasingComparator implements Comparator<Movie> {
    public int compare(Movie a, Movie b) {
        return a.duration - b.duration;
    }
}

class MovieDurationDecreasingComparator implements Comparator<Movie> {
    public int compare(Movie a, Movie b) {
        return b.duration - a.duration;
    }
}

class MovieLexicographicalComparator implements Comparator<Movie> {
    public int compare(Movie a, Movie b) {
        return a.name.compareTo(b.name);
    }
}

class MovieRatingIncreasingComparator implements Comparator<Movie> {
    public int compare(Movie a, Movie b) {
        if (a.ratings - b.ratings < 0)
            return -1;
        return +1;
    }
}
```

```java
Arrays.sort(arr, new MovieDurationIncreasingComparator());

for (Movie val : arr)
    System.out.println(val);

System.out.println(x: " ----------  ");

Arrays.sort(arr, new MovieDurationDecreasingComparator());

for (Movie val : arr)
    System.out.println(val);
System.out.println(x: " ----------  ");

Arrays.sort(arr, new MovieRatingIncreasingComparator());

for (Movie val : arr)
    System.out.println(val);
System.out.println(x: " ----------  ");

Arrays.sort(arr, new MovieLexicographicalComparator());

for (Movie val : arr)
    System.out.println(val);
System.out.println(x: " ----------  ");
```

Approach ③ Using Comparator

```
Name : Thor of 50 Minutes  with 1.0 ratings
Name : Spiderman of 100 Minutes  with 3.0 ratings
Name : Titanic of 150 Minutes  with 5.0 ratings
Name : Avengers of 180 Minutes  with 4.5 ratings
Name : Avatar of 200 Minutes  with 5.0 ratings
 ----------
Name : Avatar of 200 Minutes  with 5.0 ratings
Name : Avengers of 180 Minutes  with 4.5 ratings
Name : Titanic of 150 Minutes  with 5.0 ratings
Name : Spiderman of 100 Minutes  with 3.0 ratings
Name : Thor of 50 Minutes  with 1.0 ratings
 ----------
```

```
Name : Thor of 50 Minutes  with 1.0 ratings
Name : Spiderman of 100 Minutes  with 3.0 ratings
Name : Avengers of 180 Minutes  with 4.5 ratings
Name : Avatar of 200 Minutes  with 5.0 ratings
Name : Titanic of 150 Minutes  with 5.0 ratings
 ----------
Name : Avatar of 200 Minutes  with 5.0 ratings
Name : Avengers of 180 Minutes  with 4.5 ratings
Name : Spiderman of 100 Minutes  with 3.0 ratings
Name : Thor of 50 Minutes  with 1.0 ratings
Name : Titanic of 150 Minutes  with 5.0 ratings
 ----------
```

→ functional interfaces in Java

eg Runnable, Comparable, Comparator, etc

↳ Anonymous classes / Two name

Interface with only one abstract function

```
interface Operation {
    int operation(int a, int b);
}
```

```
Operation sum = (a, b) -> a + b;
Operation prod = (a, b) -> a * b;
Operation sub = (a, b) -> a - b;
```

} Implementing Interface or overriding function in interface

```
private int operate(int a, int b, Operation op) {
    return op.operation(a, b);
}
```

```
LambdaFunctions myCalculator = new LambdaFunctions();
System.out.println(myCalculator.operate( a: 5,  b: 3, sum));
System.out.println(myCalculator.operate( a: 5,  b: 3, prod));
System.out.println(myCalculator.operate( a: 5,  b: 3, sub));
```

@ Functional Interface

```java
interface Operation {
    public int applyOp(int a, int b);
}

class Sum implements Operation {
    public int applyOp(int a, int b) {
        return a + b;
    }
}

class Difference implements Operation {
    public int applyOp(int a, int b) {
        return a - b;
    }
}

class Product implements Operation {
    public int applyOp(int a, int b) {
        return a * b;
    }
}

class Division implements Operation {
    public int applyOp(int a, int b) {
        return a / b;
    }
}
```

```java
Operation obj = new Sum();
System.out.println(obj.applyOp(a: 15, b: 10));

Operation obj2 = new Difference();
System.out.println(obj2.applyOp(a: 15, b: 10));

Operation obj3 = new Product();
System.out.println(obj3.applyOp(a: 15, b: 10));

Operation obj4 = new Division();
System.out.println(obj4.applyOp(a: 15, b: 10));
```

→ 25
→ 5
→ 150
→ 1

OR

```java
Operation sum = (a, b) -> a + b;
Operation diff = (a, b) -> a - b;
Operation prod = (a, b) -> a * b;
Operation div = (a, b) -> a / b;

System.out.println(sum.applyOp(a: 15, b: 10));
System.out.println(diff.applyOp(a: 15, b: 10));
System.out.println(prod.applyOp(a: 15, b: 10));
System.out.println(div.applyOp(a: 15, b: 10));
```

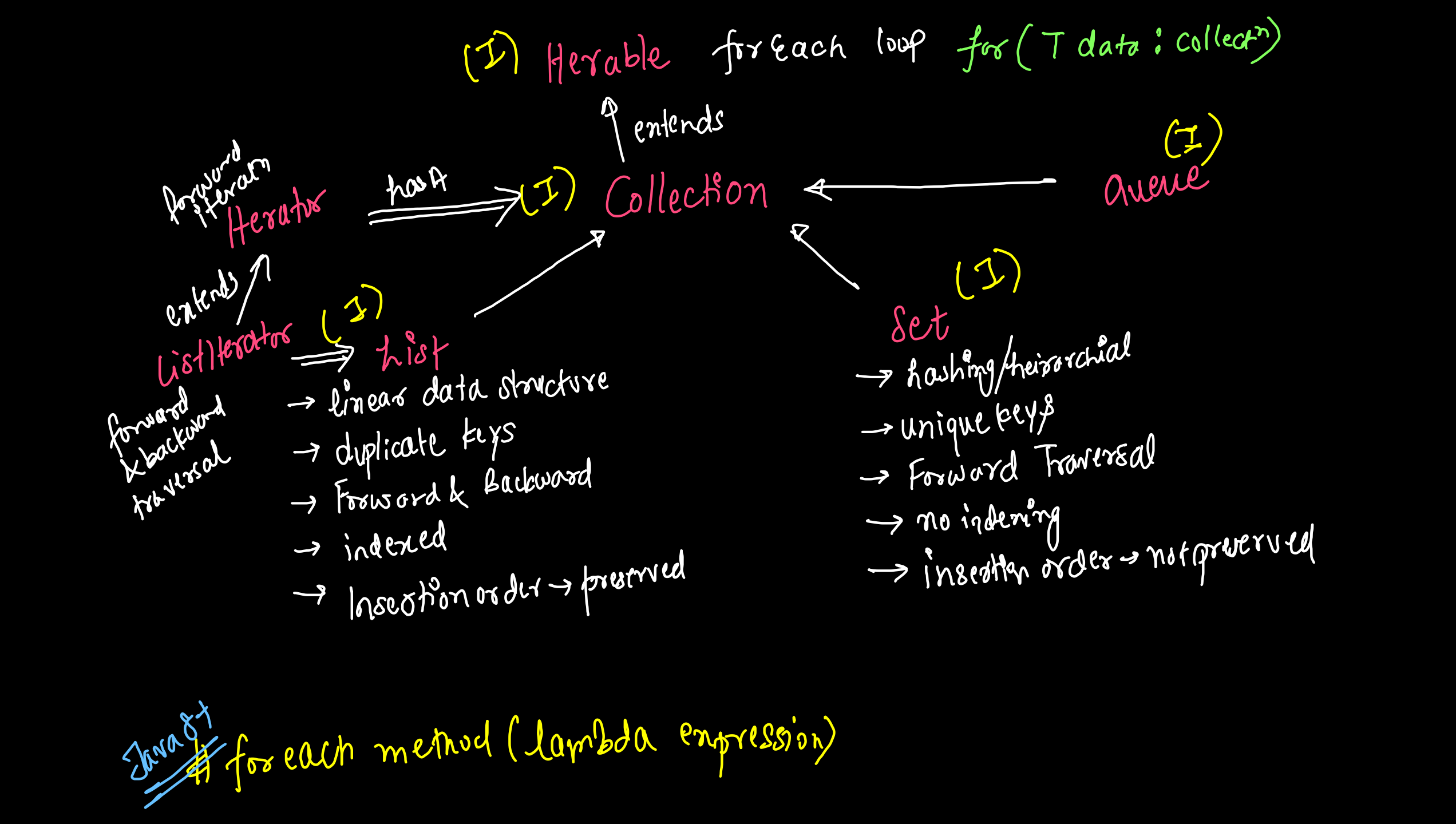
Collection

Group of Objects

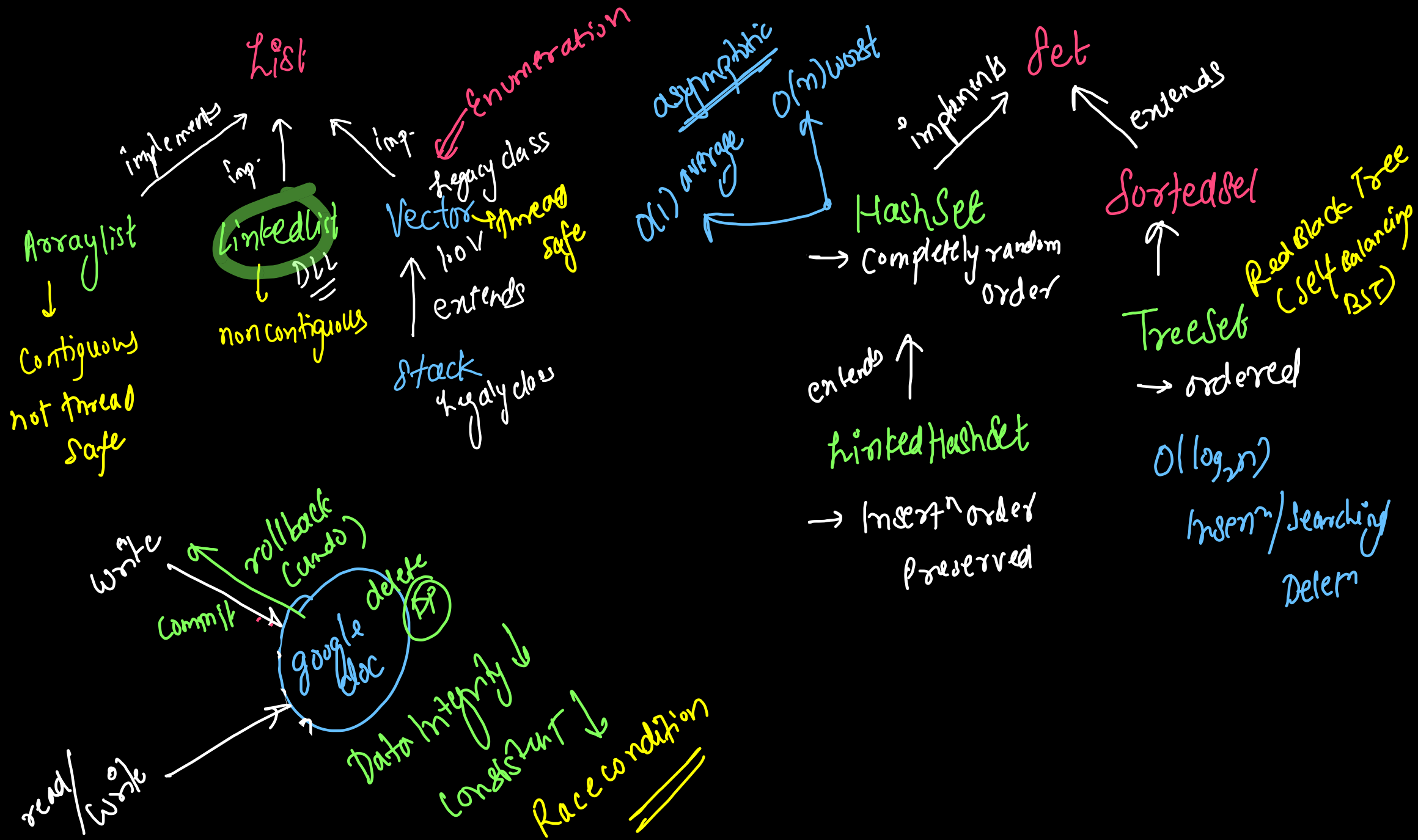
Frameworks ⇒ Java API

Package :- java.util

Set of classes & interfaces which are pre-built



Java 8 # for each method (lambda expression)


List
implements
imp.
imp.
imp.
Enumeration
asymptotic
Arraylist
Contiguous
not thread safe
LinkedList
DLL
non contiguous
Vector
legacy class
thread safe
extends
Stack
Legacy class
O(1) average
O(n) worst
Set
implements
extends
HashSet
→ Completely random order
extends
LinkedHashSet
→ Insertn order preserved
SortedSet
RedBlack Tree (self balancing BST)
TreeSet
→ ordered
O(log2n)
Insertn/Searching
Deletn
write
rollback (undo)
commit
google doc
delete
ACID
read/write
Data Integrity
Consistent
Race condition

```java
Set<Integer> s1 = new HashSet<>();
s1.add(e: 30);
s1.add(e: 10);
s1.add(e: 40);
s1.add(e: 50);
s1.add(e: 20);
s1.add(e: 10); // Ignored

Set<Integer> s2 = new LinkedHashSet<>();
s2.add(e: 30);
s2.add(e: 10);
s2.add(e: 40);
s2.add(e: 50);
s2.add(e: 20);
s2.add(e: 10); // Ignored

Set<Integer> s3 = new TreeSet<>();
s3.add(e: 30);
s3.add(e: 10);
s3.add(e: 40);
s3.add(e: 50);
s3.add(e: 20);
s3.add(e: 10); // Ignored
```

```java
for (Integer a : s1)
    System.out.print(a + " ");
System.out.println();

for (Integer a : s2)
    System.out.print(a + " ");
System.out.println();

for (Integer a : s3)
    System.out.print(a + " ");
System.out.println();
```

→ Random

→ Insertion order

→ Sorted (Inc)

```
50 20 40 10 30
30 10 40 50 20
10 20 30 40 50
```

Queue

extends

Heap

implements

duplicate can be stored

Priority Queue

Heap

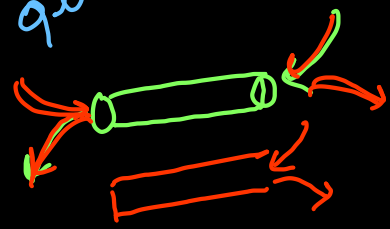
Insertn :→ O(log n)

Get Highest Priority :→ O(1)

delete Highest Priority :→ O(log n)

Searching :→ O(n)

Deque → Doubly ended queue



ArrayDeque

- addFirst - O(1)
- addLast - O(1)
- removeFirst - O(1)
- removeLast - O(1)

(addFirst, addLast, removeFirst, removeLast) → Deque

(removeFirst, removeLast) → Queue

```java
Queue<Integer> q1 = new ArrayDeque<>();
q1.add(e: 30);
q1.add(e: 10);
q1.add(e: 10);
q1.add(e: 20);
q1.add(e: 40);
q1.remove();

System.out.println(q1);

Deque<Integer> q2 = new ArrayDeque<>();
q2.addFirst(e: 30);
q2.addLast(e: 50);
q2.addLast(e: 10);
q2.add(e: 20);
q2.add(e: 30);
q2.remove();
q2.removeFirst();
q2.removeLast();

System.out.println(q2);
```

O(1) insert/removal

O(1)

O(1)

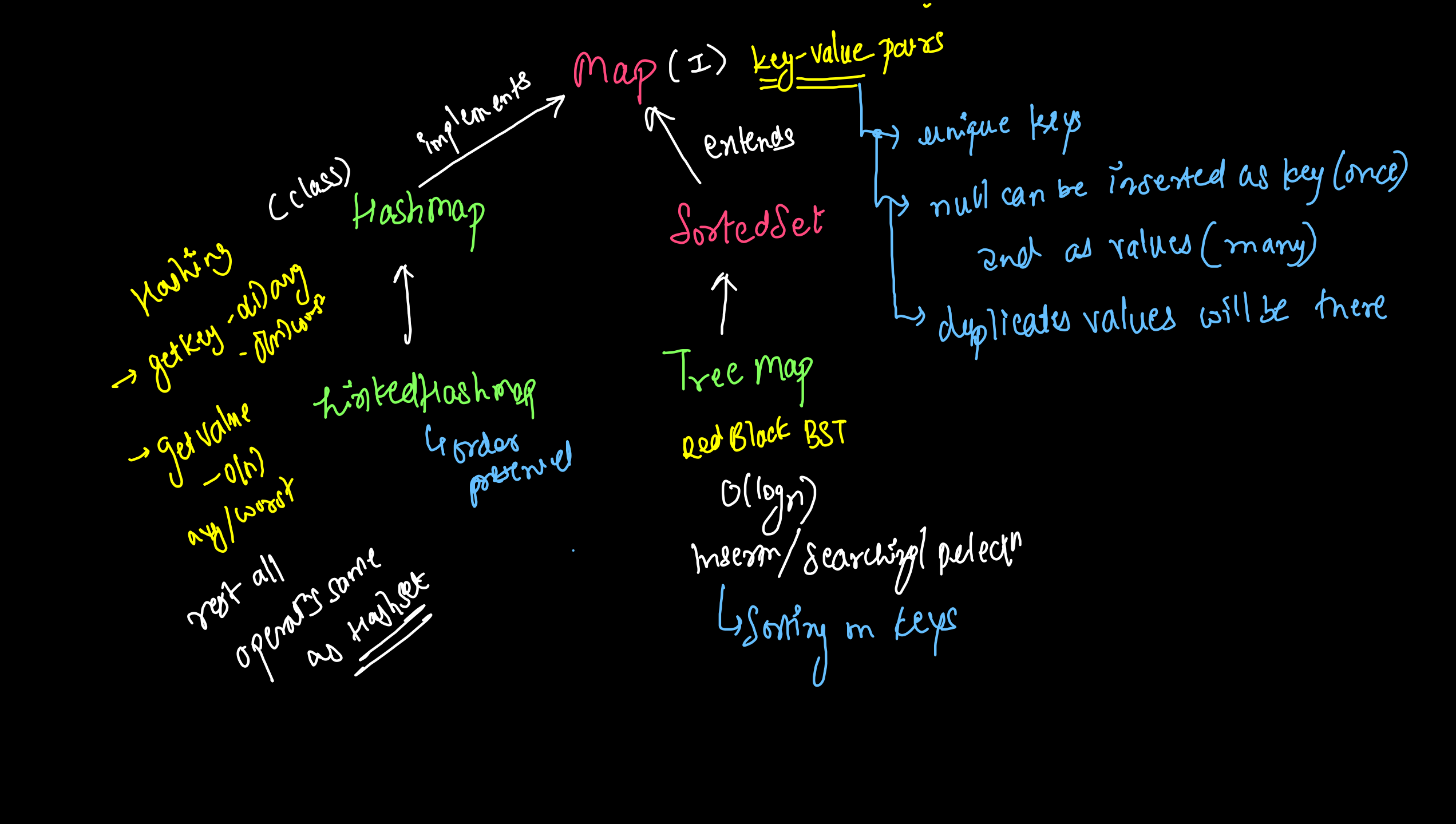
O(1)

O(1)

```java
Queue<Integer> q3 = new PriorityQueue<>();
q3.add(e: 30);
q3.add(e: 50);
q3.add(e: 10);
q3.add(e: 20);
q3.add(e: 60);
q3.add(e: 70);
q3.add(e: 90);
q3.add(e: 20);
q3.add(e: 30);

System.out.println(q3); // Not Necessarily Sorted (Heap Order Property)

// Heap Sort -> Sorted
while (q3.size() > 0) {
    System.out.print(q3.remove() + " ");
}
```

→ Min Heap

Duplicates allowed
↳ Data not stored in sorted form but deleted from HP → LP


Map (I)
key-value pairs
implements
extends
(class)
HashMap
SortedSet
unique keys
null can be inserted as key (once)
and as values (many)
duplicates values will be there
Hashing
→ getKey - O(1) avg
- O(n) worst
→ getValue
- O(n)
avg/worst
rest all operations same as HashSet
LinkedHashmap
↳ order preserved
TreeMap
Red Black BST
O(log n)
insertion/searching/deletion
↳ Sorting on keys

```java
Map<String, Integer> m1 = new HashMap<>();
m1.put(key: "Delhi", value: 30);
m1.put(key: "Delhi", value: 10);
m1.put(key: null, value: 40);
m1.put(key: null, value: 50);
m1.put(key: "Mumbai", value: null);
m1.put(key: "Kolkatta", value: null);

Map<String, Integer> m2 = new LinkedHashMap<>();
m2.put(key: "Delhi", value: 30);
m2.put(key: "Delhi", value: 10);
m2.put(key: null, value: 40);
m2.put(key: null, value: 50);
m2.put(key: "Mumbai", value: null);
m2.put(key: "Kolkatta", value: null);

Map<String, Integer> m3 = new TreeMap<>();
m3.put(key: "Delhi", value: 30);
m3.put(key: "Delhi", value: 10);
// m3.put(null, 40); Tremap key cannot be null
m3.put(key: "Mumbai", value: null);
m3.put(key: "Kolkatta", value: null);
```

```java
for (String a : m1.keySet())
    System.out.print(a + " -> " + m1.get(a) + " ");
System.out.println();

for (String a : m2.keySet())
    System.out.print(a + " -> " + m2.get(a) + " ");
System.out.println();

for (String a : m3.keySet())
    System.out.print(a + " -> " + m3.get(a) + " ");
System.out.println();
```

↳ random order

↳ insertion order

↳ sorted order (on keys)

```
null -> 50 Delhi -> 10 Kolkatta -> null Mumbai -> null
Delhi -> 10 null -> 50 Mumbai -> null Kolkatta -> null
Delhi -> 10 Kolkatta -> null Mumbai -> null
```

```java
Map<Student, Integer> m4 = new HashMap<>();

Student st1 = new Student();
st1.rollNo = 1;
Student st2 = new Student();
st2.rollNo = 2;
Student st3 = new Student();
st3.rollNo = 3;
Student st4 = new Student();
st4.rollNo = 1;
Student st5 = st2;
```

Object — Override

st1 — 4k — 500
st2 — 6k — 700
st3 — 8k — 100
st4 — 10k — 500
st5 — 6k — 700

```java
m4.put(st1, value: 10);
m4.put(st2, value: 20);
m4.put(st3, value: 30);
m4.put(st4, value: 40);
m4.put(st5, value: 50);

System.out.println(m4);
```

```java
class Student {
    int marks;
    int rollNo;
    String name;

    @Override
    public int hashCode() {
        return Integer.hashCode(rollNo);
    }

    @Override
    public boolean equals(Object other) {
        if (this.hashCode() == other.hashCode())
            return true;
        return false;
    }
}
```

custom hashing

If there hashCode will not be there then 4 objects hashing based on address

3 keys ⇒ st1 == st4, st2 == st5

```
{OOPS_Codes.Student@1=40, OOPS_Codes.Student@2=50, OOPS_Codes.Student@3=30}
```

```java
Map<ArrayList<Integer>, Integer> m5 = new HashMap<>();

ArrayList<Integer> a1 = new ArrayList<>();
a1.add(e: 10);
a1.add(e: 20);

ArrayList<Integer> a2 = new ArrayList<>();
a2.add(e: 10);
a2.add(e: 20);

ArrayList<Integer> a3 = a1;

m5.put(a1, value: 100);
m5.put(a2, value: 200);
m5.put(a3, value: 300);

System.out.println(m5);
```

hashcode() overridden & data is compared

```
{[10, 20]=300}
```

Forward Iteration

Iterable & Iterator

extended by

Collection Interface

⇓

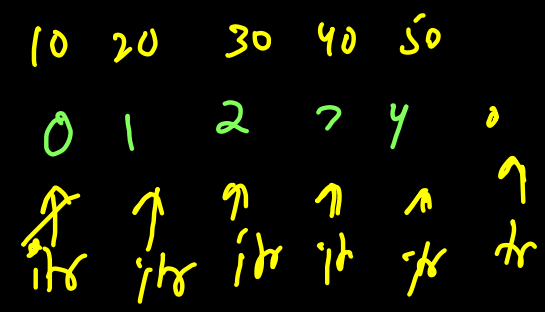
Syntatical Sugar

for each loop

```
for( object T: collection){

}
```

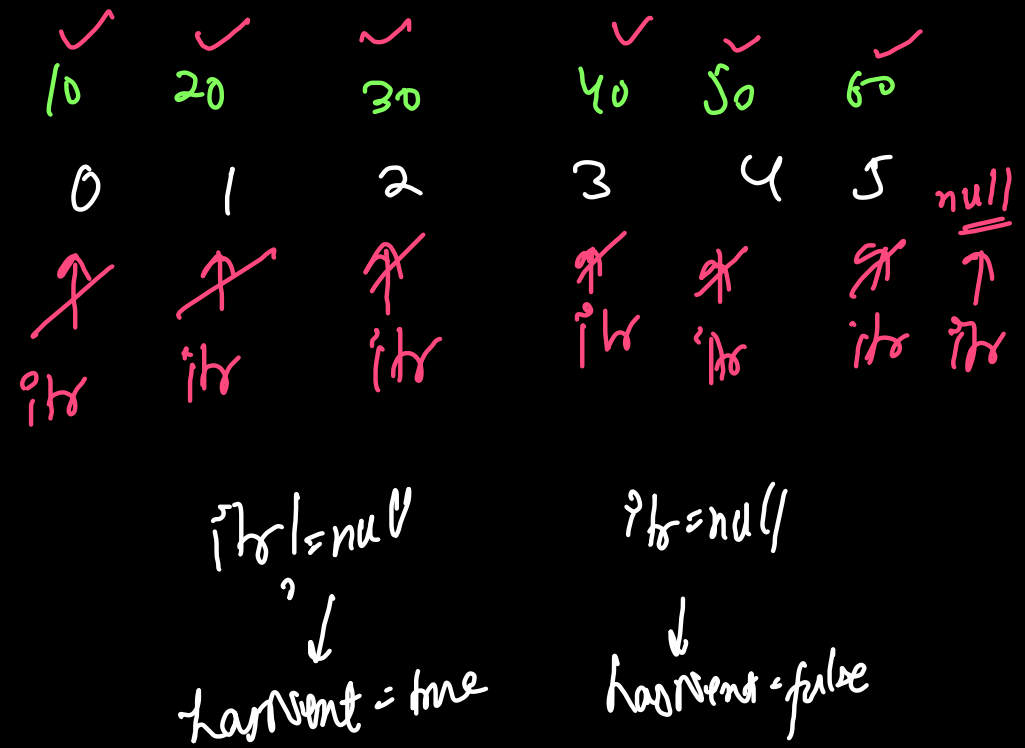
Collection HAS A iterator interface reference variable

- next() :→ return current element & setup for next element
  - If no elements found, throws Exception
- hasNext() :→ true if element(s) are remaining to be traversed otherwise false



```java
ArrayList<Integer> arr = new ArrayList<>();
arr.add(e: 10);
arr.add(e: 20);
arr.add(e: 30);
arr.add(e: 40);
arr.add(e: 50);
arr.add(e: 60);

// Iterable : For Each Loop : Syntatical Sugar
for (Integer data : arr) {
    System.out.print(data + " ");
}
System.out.println();

// For Each Method (Java 8+ Feature)
arr.forEach((data) -> System.out.print(data + " "));
System.out.println();

// Iterator:
Iterator<Integer> itr = arr.iterator();
while (itr.hasNext() == true) {
    System.out.print(itr.next() + " ");
}
```

hasA relationship



```java
// Enumeration: Iterate on Vector and Stack
Vector<Integer> v = new Vector<>();
v.add(e: 10);
v.add(e: 20);
v.add(e: 30);
v.add(e: 40);
v.add(e: 50);
v.add(e: 60);

Enumeration<Integer> e = v.elements();
while (e.hasMoreElements() == true) {
    System.out.print(e.nextElement() + " ");
}
System.out.println();

// List Iterator
ListIterator<Integer> li = arr.listIterator();
while (li.hasNext() == true) {
    System.out.print(li.next() + " ");
}
System.out.println();

ListIterator<Integer> bi = arr.listIterator(arr.size());
while (bi.hasPrevious() == true) {
    System.out.print(bi.previous() + " ");
}
System.out.println();
```

```
10 20 30 40 50 60
10 20 30 40 50 60
10 20 30 40 50 60
10 20 30 40 50 60
10 20 30 40 50 60
60 50 40 30 20 10
```

→ n th element is not included!

Custom Iterators in Java

(1) Peeking Iterator LC 284

(2) Flatten Nested List Iterator LC 341

(3) BST Iterator - I LC 173

(4) BST Iterator - II Leetcode Locked - Codestudio

(5) Two Sum in BST LC 653

Peeking Iterator

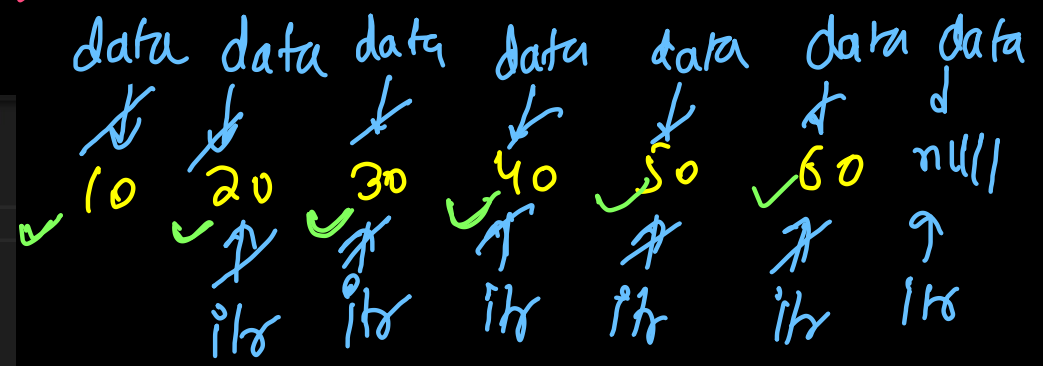
```java
class PeekingIterator implements Iterator<Integer> {
    Iterator<Integer> itr;
    Integer data;

    public PeekingIterator(Iterator<Integer> itr) {
        this.itr = itr;
        next();
    }

    public Integer peek() {
        return data;
    }

    @Override
    public Integer next() {
        Integer temp = data;

        if(itr.hasNext() == true){
            data = itr.next();
        } else {
            data = null;
        }

        return temp;
    }

    @Override
    public boolean hasNext() {
        return (data != null);
    }
}
```

1) Constructor

2) peek()

3) next()

4) hasNext()



peek() → 10

next() → 10, d=20, i=30

peek() → 20

next() → 20, d=30, i=40

peek() → 30

next() → 30, d=40, i=50

peek() → 40

next() → 40, d=50, i=60

peek() → 50

next() → 50, d=60

hasNext()
↳ data != null

peek → 60

next() → 60, d=null

hasNext()
↳ data == null

```java
class PeekingIterator implements Iterator<Integer> {
    Iterator<Integer> itr;
    Integer data;

    public PeekingIterator(Iterator<Integer> itr) {
        this.itr = itr;
        next();
    }

    public Integer peek() {
        return data;
    }

    @Override
    public Integer next() {
        Integer temp = data;
        if(itr.hasNext() == true){
            data = itr.next();
        } else {
            data = null;
        }

        return temp;
    }

    @Override
    public boolean hasNext() {
        return (data != null);
    }
}
```

Extra functionality

hasnext or collection

Iterator on collection

For List (AL, LL, Vector, Stack)
↳ peek, next, hasNext → O(1)

For Queue (ArrayDeque & PriorityQueue)
↳ peek, next, hasNext → O(1)

For Set (Hashmap)
↳ peek, next, hasNext → Θ(1) avg

ForSet (TreeSet)
↳ peek → O(1)
hasnext → O(1)
next → Θ(1) avg

Flatten Nested List Iterator

NestedList ⇒ List< NestedInteger >

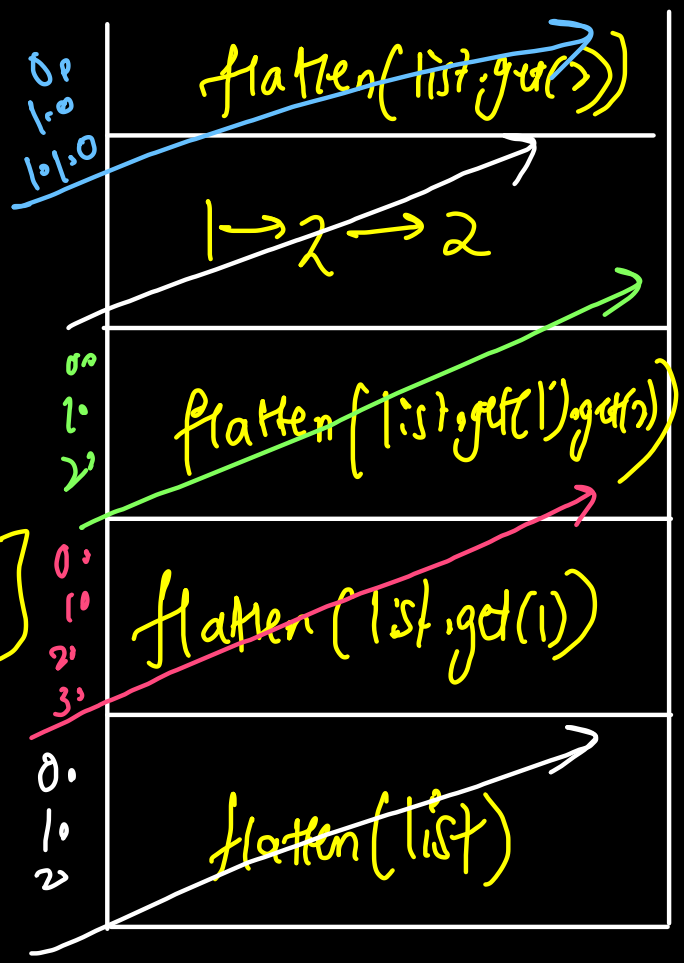
[10, [20, 30, [40, 50, [ ]], 60], [70 [80 [90]]]]

↓ convert to 1D list of integers

[10, 20, 30, 40, 50, 60, 70, 80, 90]

↓ iterator (inbuilt) next & hasNext

```java
public interface NestedInteger {

    // @return true if this NestedInteger holds a single integer, rather than a nested list.
    public boolean isInteger();

    // @return the single integer that this NestedInteger holds, if it holds a single integer
    // Return null if this NestedInteger holds a nested list
    public Integer getInteger();

    // @return the nested list that this NestedInteger holds, if it holds a nested list
    // Return empty list if this NestedInteger holds a single integer
    public List<NestedInteger> getList();
}
```



[10, 20, 30, 40, 50, 60,]

```java
public class NestedIterator implements Iterator<Integer> {
    List<Integer> arr;
    Iterator<Integer> itr;

    public NestedIterator(List<NestedInteger> nestedList) {
        arr = new ArrayList<>();
        flatten(nestedList);
        itr = arr.iterator();
    }

    public void flatten(List<NestedInteger> nestedList){
        for(NestedInteger data: nestedList){
            if(data.isInteger() == true){
                arr.add(data.getInteger());
            } else {
                flatten(data.getList());
            }
        }
    }

    @Override
    public Integer next() {
        return itr.next();
    }

    @Override
    public boolean hasNext() {
        return itr.hasNext();
    }
}
```

extra space flattened list → O(n) space

preprocessing

O(n) time

O(1) time on list

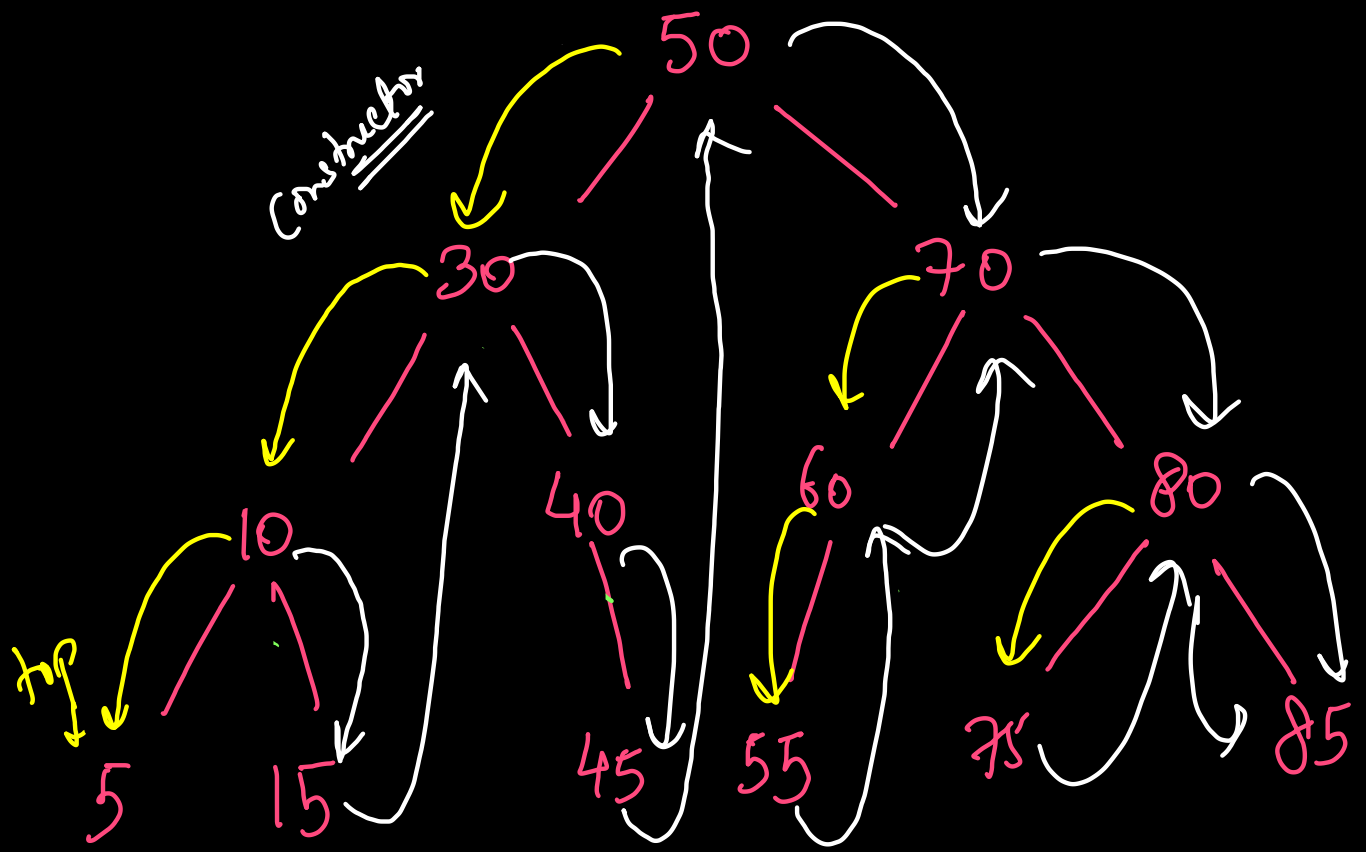
driver code

NestedIterator obj = new NestedIterator(trav);

① Constructor call

```
while(obj.hasNext())
    Syso(obj.next());
```

Binary Search Tree Iterator



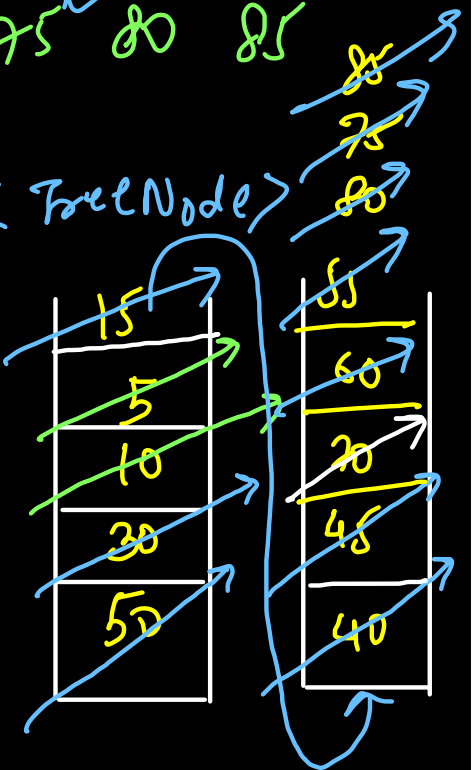
root

5 10 15 30 40

45 50 55 60

70 75 80 85

Stack<TreeNode>



stk.size() == 0 : false

stk.size() > 0 : true

```java
class BSTIterator{
    Stack<TreeNode> stk;

    public BSTIterator(TreeNode root) {
        stk = new Stack<>();
        inorderSucc(root);
    }

    public void inorderSucc(TreeNode curr){
        while(curr != null){
            stk.push(curr);
            curr = curr.left;
        }
    }

    public int next() {
        TreeNode curr = stk.pop();
        inorderSucc(curr.right);
        return curr.val;
    }

    public boolean hasNext() {
        return (stk.size() > 0);
    }
}
```

→ preprocessing → min node is not root
↳ go to leftmost node in BST (min node)

→ $O(\log_2 N) = O(h)$ in worst case

→ current ele return

→ next ele using (ceil/next larger value in stk = top())

Asymptotic
↳ worst $O(\log_2 n)$

Amortized
↳ $O(1)$ avg case

→ $O(1)$

```java
// Single BST Node
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode() {
    }

    TreeNode(int val) {
        this.val = val;
    }
}
```

```java
// Collection of Nodes
class BinarySearchTree {
    private TreeNode root;

    public void insert(int val) {
        root = insert(root, val);
    }

    private TreeNode insert(TreeNode root, int val) {
        if (root == null)
            return new TreeNode(val);

        if (val < root.val)
            root.left = insert(root.left, val);

        else if (val > root.val)
            root.right = insert(root.right, val);

        return root;
    }
}
```

```java
public static void main(String[] args) {
    BinarySearchTree tree = new BinarySearchTree();
    tree.insert(val: 70);
    tree.insert(val: 50);
    tree.insert(val: 90);
    tree.insert(val: 30);
    tree.insert(val: 40);
    tree.insert(val: 80);
    tree.insert(val: 100);
}
```

Custom class
↳ Iterable & Iteral

BST class

```java
// Collection of Nodes
class BinarySearchTree implements Iterable<Integer> {
    TreeNode root;

    public void insert(int val) {
        root = insert(root, val);
    }

    private TreeNode insert(TreeNode root, int val) {
        if (root == null)
            return new TreeNode(val);

        if (val < root.val)
            root.left = insert(root.left, val);

        else if (val > root.val)
            root.right = insert(root.right, val);

        return root;
    }

    @Override
    public Iterator<Integer> iterator() {
        BSTIterator itr = new BSTIterator(root);
        return itr;
    }
}
```

for each method

Iterator class

```java
class BSTIterator implements Iterator<Integer> {
    Stack<TreeNode> stk;

    public BSTIterator(TreeNode root) {
        stk = new Stack<>();
        inorderSucc(root);
    }

    public void inorderSucc(TreeNode curr) {
        while (curr != null) {
            stk.push(curr);
            curr = curr.left;
        }
    }

    @Override
    public Integer next() {
        TreeNode curr = stk.pop();
        inorderSucc(curr.right);
        return curr.val;
    }

    @Override
    public boolean hasNext() {
        return (stk.size() > 0);
    }
}
```

driver code

```java
public class IteratorIterable {
    Run | Debug
    public static void main(String[] args) {
        BinarySearchTree tree = new BinarySearchTree();
        tree.insert(val: 70);
        tree.insert(val: 50);
        tree.insert(val: 90);
        tree.insert(val: 30);
        tree.insert(val: 40);
        tree.insert(val: 80);
        tree.insert(val: 100);

        // Iterable: For Each Loop
        for (Integer data : tree) {
            System.out.print(data + " ");
        }
        System.out.println();

        // Iterator
        BSTIterator itr = new BSTIterator(tree.root);
        while (itr.hasNext() == true) {
            System.out.print(itr.next() + " ");
        }
    }
}
```

root should not be private

Output (Sorted → Inorder)

30 40 50 70 80 90 100

```java
// Collection of Nodes
class BinarySearchTree implements Iterable<Integer> {
    private TreeNode root;

    public void insert(int val) {
        root = insert(root, val);
    }

    private TreeNode insert(TreeNode root, int val) {
        if (root == null)
            return new TreeNode(val);

        if (val < root.val)
            root.left = insert(root.left, val);

        else if (val > root.val)
            root.right = insert(root.right, val);

        return root;
    }

    @Override
    public Iterator<Integer> iterator() {
        BSTIterator itr = new BSTIterator(root);
        return itr;
    }
}
```

```java
public static void main(String[] args) {
    BinarySearchTree tree = new BinarySearchTree();
    tree.insert(val: 70);
    tree.insert(val: 50);
    tree.insert(val: 90);
    tree.insert(val: 30);
    tree.insert(val: 40);
    tree.insert(val: 80);
    tree.insert(val: 100);

    // Iterable: For Each Loop
    for (Integer data : tree) {
        System.out.print(data + " ");
    }
    System.out.println();

    // Iterator
    Iterator<Integer> itr = tree.iterator();
    while (itr.hasNext() == true) {
        System.out.print(itr.next() + " ");
    }
}
```

```java
static class ForwardIterator{
    Stack<TreeNode> stk;

    public ForwardIterator(TreeNode root) {
        stk = new Stack<>();
        inorderSucc(root);
    }

    public void inorderSucc(TreeNode curr){
        while(curr != null){
            stk.push(curr);
            curr = curr.left;
        }
    }

    public int peek(){
        if(hasNext() == false) return 0;
        return stk.peek().val;
    }

    public int next() {
        if(hasNext() == false) return 0;
        TreeNode curr = stk.pop();
        inorderSucc(curr.right);
        return curr.val;
    }

    public boolean hasNext() {
        return (stk.size() > 0);
    }
}
```

```java
static class BackWardIterator{
    Stack<TreeNode> stk;

    public BackWardIterator(TreeNode root) {
        stk = new Stack<>();
        inorderPred(root);
    }

    public void inorderPred(TreeNode curr){
        while(curr != null){
            stk.push(curr);
            curr = curr.right;
        }
    }

    public int peek(){
        if(hasPrev() == false) return 0;
        return stk.peek().val;
    }

    public int prev() {
        if(hasPrev() == false) return 0;
        TreeNode curr = stk.pop();
        inorderPred(curr.left);
        return curr.val;
    }

    public boolean hasPrev() {
        return (stk.size() > 0);
    }
}
```

Extra Space → $O(h) = O(\log_2 n)$

Time → $O(n/2 + n/2)$

$= O(n)$

Iterate on inorder

```java
public boolean findTarget(TreeNode root, int target) {
    if(root == null || (root.left == null && root.right == null)) return false;

    ForwardIterator left = new ForwardIterator(root);
    BackWardIterator right = new BackWardIterator(root);

    while(left.hasNext() == true && right.hasPrev() == true && left.peek() < right.peek()){
        if(left.peek() + right.peek() == target) return true;
        if(left.peek() + right.peek() < target) left.next();
        else right.prev();
    }

    return false;
}
```

Exception Handling

1) Except^n vs Error
2) Exception Hierarchy
3) Compile Time (Checked) vs RunTime (Unchecked)
4) Except^n Handling :- What? Why? How?
5) Except^n object & Default Error Handling by JVM
6) 5 keywords : try, catch, finally, throw, throws
7) valid orders of try, catch & finally
8) User Defined / Custom Exceptions
9) Differences : final, finally, finalize
10) Differences : throw vs throws

Exception :→ Any abnormal behavior in your code which occurs at runtime & disturbes the normal flow by abnormal terminat (crashing).

Exception Handling :→ Alternate sequence flow provided using 5 keywords to normally terminate the program is known as exception handling.

WHAT?

```java
public static void main(String[] args) {
    System.out.println(x: "Starting Normally");

    Scanner scn = new Scanner(System.in);
    int a = scn.nextInt();
    int b = scn.nextInt();

    char op = scn.next().charAt(index: 0);

    switch (op) {
        case '+': {
            System.out.println(a + b);
            break;
        }
        case '-': {
            System.out.println(a - b);
            break;
        }
        case '*': {
            System.out.println(a * b);
            break;
        }
        case '/': {
            System.out.println(a / b);
            break;
        }
        default: {
            System.out.println(x: "Invalid Operator");
        }
    }

    System.out.println(x: "Terminating Normally");
}
```

```
architaggarwal@Archits-MacBook-Air System Design % javac Solution.java
architaggarwal@Archits-MacBook-Air System Design % java Solution
Starting Normally
10
2
/
5
Terminating Normally
architaggarwal@Archits-MacBook-Air System Design % javac Solution.java
architaggarwal@Archits-MacBook-Air System Design % java Solution
Starting Normally
10
0
/
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Solution.main(Solution.java:27)
```

classname

message

Stack Trace

Runtime Except

Handle ✗

Abnormally terminate (Crash)

```
architaggarwal@Archits-MacBook-Air System Design % javac Solution.java
architaggarwal@Archits-MacBook-Air System Design % java Solution
Starting Normally
10
abc
Exception in thread "main" java.util.InputMismatchException
        at java.base/java.util.Scanner.throwFor(Scanner.java:939)
        at java.base/java.util.Scanner.next(Scanner.java:1594)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
        at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
        at Solution.main(Solution.java:9)
```

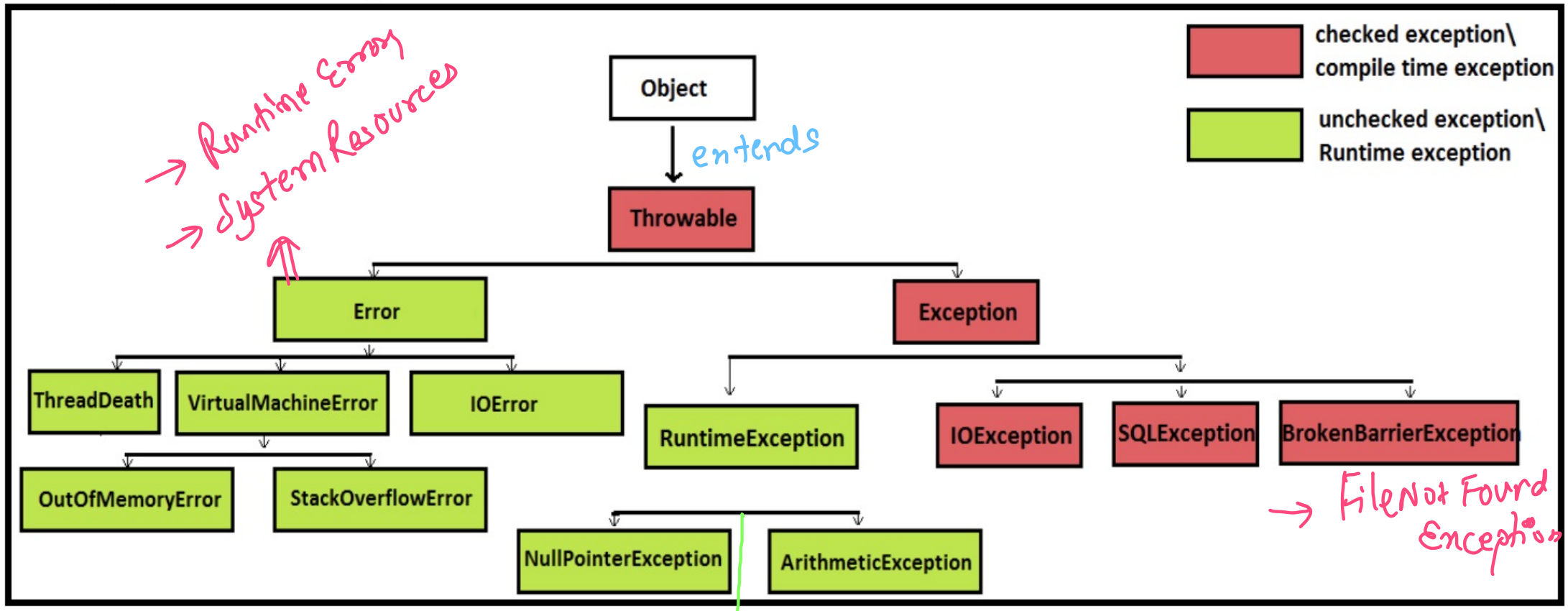
```java
String str = null;
System.out.println(str.charAt(index: 0));

System.out.println(x: "Terminating Normally");
```

```
architaggarwal@Archits-MacBook-Air System Design % javac Solution.java
architaggarwal@Archits-MacBook-Air System Design % java Solution
Starting Normally
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String.charAt(int)" because "<local1>" is null
        at Solution.main(Solution.java:36)
```

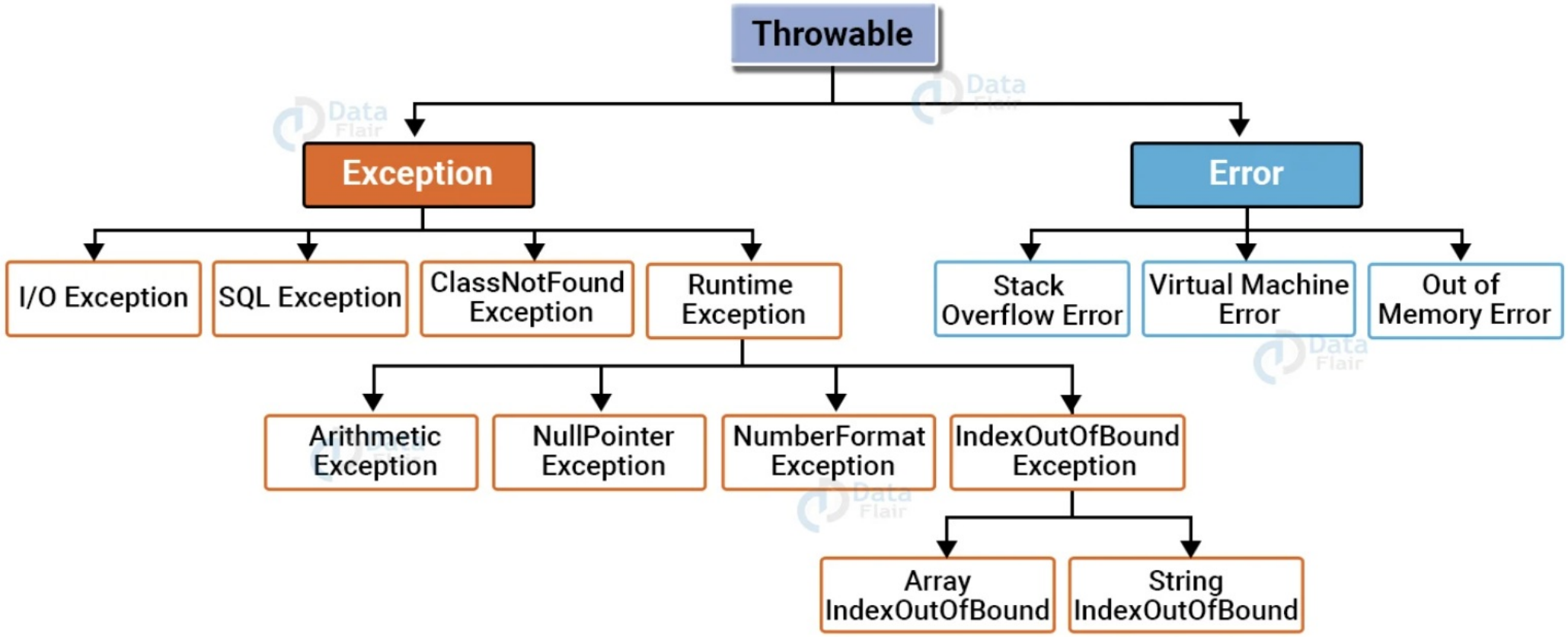
Unchecked (Runtime Excepn)

Compiler was not able to check these Exceptions

Exception hierarchy >



Hierarchy of Java Exceptions



```
FileInputStream fs = new FileInputStream(name: "d:/Archit.txt");
System.out.println(x: "Terminating Normally");
```

```
architaggarwal@Archits-MacBook-Air System Design % javac Solution.java
Solution.java:39: error: unreported exception FileNotFoundException; must be caught or declared to be thrown
        FileInputStream fs = new FileInputStream("d:/Archit.txt");
                             ^
1 error
```

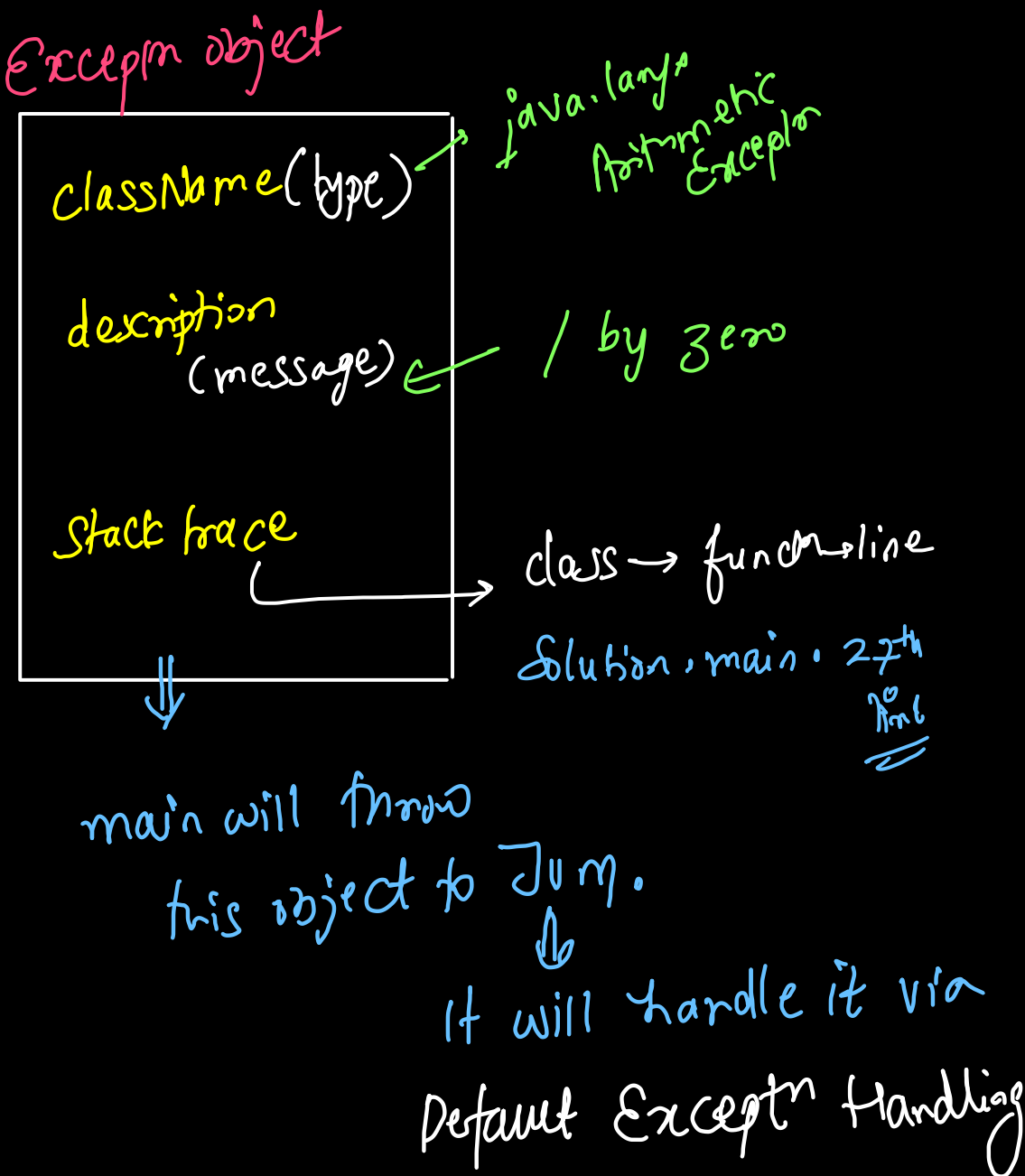
Compiler is complaining that you have not handled the FileNotFound exception so I will not compile it!

Checked / Compile Time Excepn

(Excepn will still happen at run time!)

```java
public static void main(String[] args) {
    System.out.println(x: "Starting Normally");

    Scanner scn = new Scanner(System.in);
    int a = scn.nextInt();
    int b = scn.nextInt();

    char op = scn.next().charAt(index: 0);

    switch (op) {
        case '+': {
            System.out.println(a + b);
            break;
        }
        case '-': {
            System.out.println(a - b);
            break;
        }
        case '*': {
            System.out.println(a * b);
            break;
        }
        case '/': {
            System.out.println(a / b);
            break;
        }
        default: {
            System.out.println(x: "Invalid Operator");
        }
    }

    System.out.println(x: "Terminating Normally");
}
```

← Arithmetic Exception



5 keywords

(1) try ⟶ risky code (chances of Excepn)

(2) catch ⟶ alternate flow (handling the Exception)

(3) finally ⟶ cleanup code (FileInputOutput, I/O Stream close, memory release)

(4) throw ⟶ throw custom Exception from the current fn to the calling fn!
↳ Runtime

(5) throws ⟶ calling fn gets to know called fn might throw Exception!
↳ Checked Exception

```java
public static void main(String[] args) {
    System.out.println(x: "Starting Normally");

    Scanner scn = new Scanner(System.in);
    int a = scn.nextInt();
    int b = scn.nextInt();

    char op = scn.next().charAt(index: 0);

    switch (op) {
        case '+': {
            System.out.println(a + b);
            break;
        }
        case '-': {
            System.out.println(a - b);
            break;
        }
        case '*': {
            System.out.println(a * b);
            break;
        }
        case '/': {
            try {
                System.out.println(a / b);
            } catch (ArithmeticException e) {
                System.out.println(x: "Division by Zero Not Allowed");
            }
            break;
        }
        default: {
            System.out.println(x: "Invalid Operator");
        }
    }

    System.out.println(x: "Terminating Normally");
}
}
```

```
architaggarwal@Archits-MacBook-Air System Design % java Solution
Starting Normally
10
2
/
5
Terminating Normally
architaggarwal@Archits-MacBook-Air System Design % javac Solution.java
architaggarwal@Archits-MacBook-Air System Design % java Solution
Starting Normally
10
0
/
Division by Zero Not Allowed
Terminating Normally
```

```java
System.out.println(x: "Starting Normally");

try {
    Scanner scn = new Scanner(System.in);
    int a = scn.nextInt();
    int b = scn.nextInt();
    char op = scn.next().charAt(index: 0);

    switch (op) {
        case '+': {
            System.out.println(a + b);
            break;
        }
        case '-': {
            System.out.println(a - b);
            break;
        }
        case '*': {
            System.out.println(a * b);
            break;
        }
        case '/': {
            System.out.println(a / b);
            break;
        }
        default: {
            System.out.println(x: "Invalid Operator");
        }
    }
} catch (Exception e) {
    System.out.println(e);
}

System.out.println(x: "Terminating Normally");
```

```
architaggarwal@Archits-MacBook-Air System Design % javac Solution.java
architaggarwal@Archits-MacBook-Air System Design % java Solution
Starting Normally
10
abc
java.util.InputMismatchException
Terminating Normally
```

```
architaggarwal@Archits-MacBook-Air System Design % javac Solution.java
architaggarwal@Archits-MacBook-Air System Design % java Solution
Starting Normally
10
0
/
java.lang.ArithmeticException: / by zero
Terminating Normally
```

```java
System.out.println(x: "Starting Normally");

try {
    System.out.println(x: "Inside Try Block Before Input");
    Scanner scn = new Scanner(System.in);
    int a = scn.nextInt();
    int b = scn.nextInt();
    char op = scn.next().charAt(index: 0);
    System.out.println(x: "Inside Try Block After Input");

    switch (op) {
        case '+': {
            System.out.println(a + b);
            break;
        }
        case '-': {
            System.out.println(a - b);
            break;
        }
        case '*': {
            System.out.println(a * b);
            break;
        }
        case '/': {

            System.out.println(x: "Inside Swith Case Before Division");
            System.out.println(a / b);
            System.out.println(x: "Inside Swith Case After Division");
            break;
        }
        default: {
            System.out.println(x: "Invalid Operator");
        }
    }

    System.out.println(x: "Inside Try After Switch Case");
} catch (Exception e) {
    System.out.println(x: "Inside Catch");
    System.out.println(e);
}

System.out.println(x: "Terminating Normally");
```

```
architaggarwal@Archits-MacBook-Air System Design % java Solution
Starting Normally
Inside Try Block Before Input
10
2
/
Inside Try Block After Input
Inside Swith Case Before Division
5
Inside Swith Case After Division
Inside Try After Switch Case
Terminating Normally
```

No Except | e

```
architaggarwal@Archits-MacBook-Air System Design % java Solution
Starting Normally
Inside Try Block Before Input
10
abc
Inside Catch
java.util.InputMismatchException
Terminating Normally
```

Input Except | e

```
architaggarwal@Archits-MacBook-Air System Design % java Solution
Starting Normally
Inside Try Block Before Input
10
0
/
Inside Try Block After Input
Inside Swith Case Before Division
Inside Catch
java.lang.ArithmeticException: / by zero
Terminating Normally
```

Division Except | b

```java
public static void main(String[] args) {
    try {
        Integer a = Integer.parseInt(args[0]);
        Integer b = Integer.parseInt(args[1]);

        System.out.println(a / b);
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

```
architaggarwal@Archits-MacBook-Air System Design % javac Solution.java
architaggarwal@Archits-MacBook-Air System Design % java Solution 10 2
5
architaggarwal@Archits-MacBook-Air System Design % java Solution 10
java.lang.ArrayIndexOutOfBoundsException: Index 1 out of bounds for length 1
architaggarwal@Archits-MacBook-Air System Design % java Solution 10 abc
java.lang.NumberFormatException: For input string: "abc"
architaggarwal@Archits-MacBook-Air System Design % java Solution 10 0
java.lang.ArithmeticException: / by zero
```

```java
public static void main(String[] args) {
    try {
        Integer a = Integer.parseInt(args[0]);
        Integer b = Integer.parseInt(args[1]);

        System.out.println(a / b);
    } catch (ArithmeticException e) {
        System.out.println(x: "Division by Zero Not Allowed");
    } catch (NumberFormatException e) {
        System.out.println(x: "Please pass integers only");
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println(x: "Please pass atleast 2 parameters");
    } catch (Exception e) {
        System.out.println(x: "Some Other Expection Occured");
    }
}
```

multiple catch statements!

```
architaggarwal@Archits-MacBook-Air System Design % javac Solution.java
architaggarwal@Archits-MacBook-Air System Design % java Solution 10 2
5
architaggarwal@Archits-MacBook-Air System Design % java Solution 10
Please pass atleast 2 parameters
architaggarwal@Archits-MacBook-Air System Design % java Solution 10 abc
Please pass integers only
architaggarwal@Archits-MacBook-Air System Design % java Solution 10 0
Division by Zero Not Allowed
```

Valid orders of try, catch

```
try { }
```

✗ only try not possible

```
catch { }
```

✗ only catch not possible

```
try {
    try { }
    catch (e1) { }
}
catch { }
```

```
try { }
catch (e1) { }
```

both are valid

```
try { }
catch (e1) { }
catch (e2) { }
  :
catch (ek) { }
```

✓ multiple catch

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

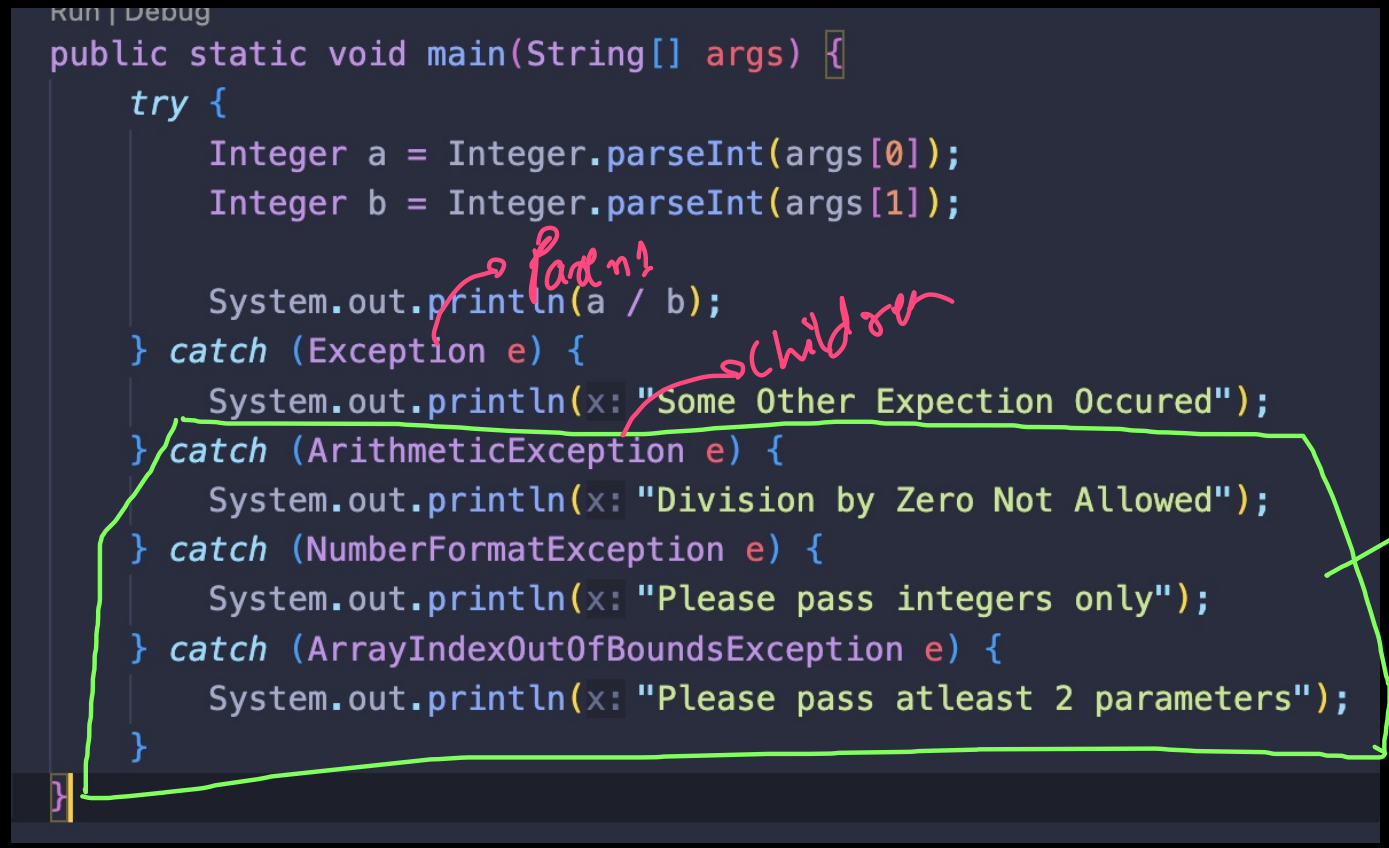
```
catch ( ) { }
    try { }
```

✗ not possible

```
try { ... }
Syso("In b/w try & catch")
catch ( ) { }
```

✗ not possible

✓ Nested Try Loop


Run | Debug
public static void main(String[] args) {
try {
Integer a = Integer.parseInt(args[0]);
Integer b = Integer.parseInt(args[1]);
System.out.println(a / b);
} catch (Exception e) {
System.out.println(x: "Some Other Expection Occured");
} catch (ArithmeticException e) {
System.out.println(x: "Division by Zero Not Allowed");
} catch (NumberFormatException e) {
System.out.println(x: "Please pass integers only");
} catch (ArrayIndexOutOfBoundsException e) {
System.out.println(x: "Please pass atleast 2 parameters");
}
}
Parent
Children
Unreachable code
↓
System/Compilation error

```java
Run | Debug
public static void main(String[] args) {
    try {
        Scanner scn = new Scanner(System.in);
        int a = scn.nextInt();
        int b = scn.nextInt();
        System.out.println(a / b);
        scn.close(); // Scanner Object -> Memory Release, Input Stream Close
    } catch (Exception e) {
        System.out.println("Exception is Handled : " + e);
    }
}
```

→ scn will only close if Except is not occured

```java
try {
    Scanner scn = new Scanner(System.in);
    int a = scn.nextInt();
    int b = scn.nextInt();
    System.out.println(a / b);
    scn.close();
} catch (Exception e) {
    System.out.println("Exception is Handled : " + e);
    scn.close();
}
```

→ Code redundancy

```java
Scanner scn = new Scanner(System.in);
try {
    int a = scn.nextInt();
    int b = scn.nextInt();
    System.out.println(a / b);

} catch (Exception e) {
    System.out.println("Exception is Handled : " + e);
    System.out.println(1 / 0);
} finally {
    System.out.println(x: "Finally Block Executed: Clean up Code");

}

scn.close();
```

This line will not execute if catch will run because catch have excepn (abnormal termination)

```java
Scanner scn = new Scanner(System.in);
try {
    int a = scn.nextInt();
    int b = scn.nextInt();
    System.out.println(a / b);

} finally {
    System.out.println(x: "Finally Block Executed: Clean up Code");
    scn.close();
}
```

```
architaggarwal@Archits-MacBook-Air System Design % javac Solution.java
architaggarwal@Archits-MacBook-Air System Design % java Solution
10 5
2
Finally Block Executed: Clean up Code
architaggarwal@Archits-MacBook-Air System Design % javac Solution.java
architaggarwal@Archits-MacBook-Air System Design % java Solution
10 0
Finally Block Executed: Clean up Code
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Solution.main(Solution.java:79)
```

→ finally executed even during abnormal terminal!

```java
Scanner scn = new Scanner(System.in);
try {
    int a = scn.nextInt();
    int b = scn.nextInt();
    System.out.println(a / b);

} catch (Exception e) {
    System.out.println("Exception is Handled : " + e);
} finally {
    System.out.println(x: "Finally Block Executed: Clean up Code");
    scn.close();
}
```

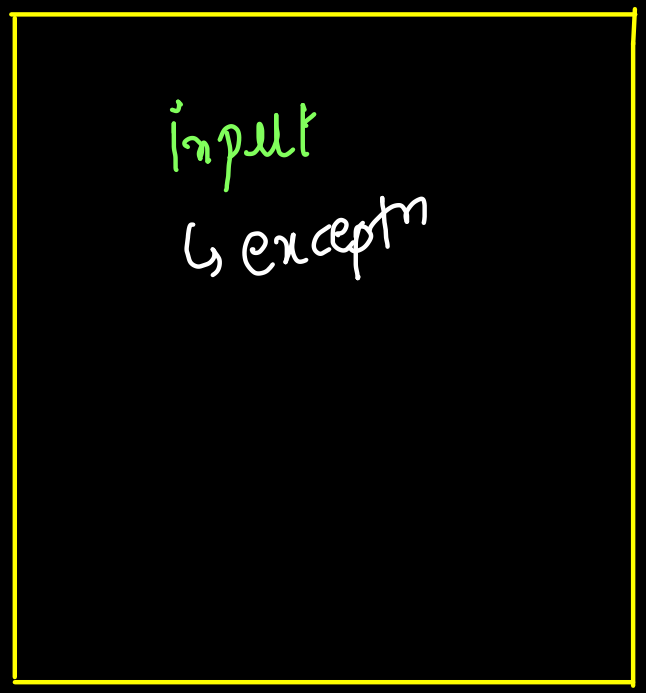
```
architaggarwal@Archits-MacBook-Air System Design % javac Solution.java
architaggarwal@Archits-MacBook-Air System Design % java Solution
10 2
5
Finally Block Executed: Clean up Code
architaggarwal@Archits-MacBook-Air System Design % java Solution
10 0
Exception is Handled : java.lang.ArithmeticException: / by zero
Finally Block Executed: Clean up Code
```

```java
Run | Debug
public static void main(String[] args) {
    try {
        FileInputStream scn = new FileInputStream(name: "d:/abc.txt");
    } catch (Exception e) {
        System.out.println("Exception Occured: " + e);
    }
}
```

```
architaggarwal@Archits-MacBook-Air System Design % javac Solution.java
architaggarwal@Archits-MacBook-Air System Design % java Solution
Exception Occured: java.io.FileNotFoundException: d:/abc.txt (No such file or directory)
```

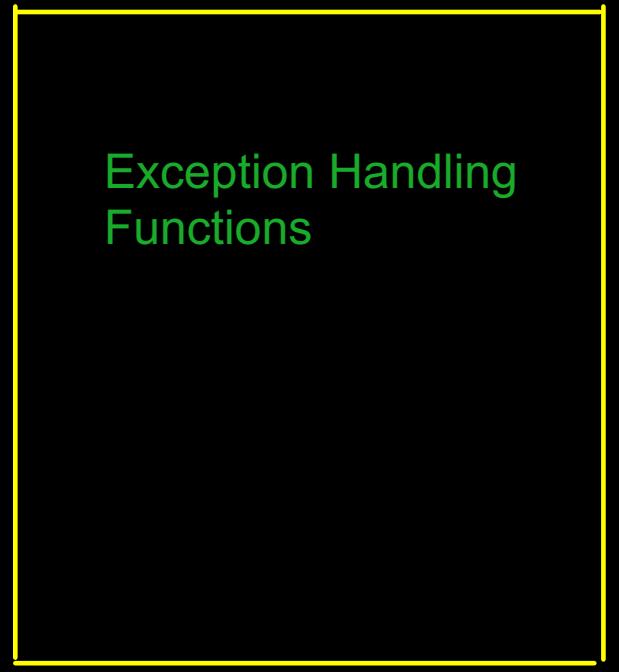
javac : accepts handled using try & catch : no problem

Checked excepn occured at runtime!



Driver
(main)

Calculator



Exception Handler

```java
class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public int subtract(int a, int b) {
        return a - b;
    }

    public int divide(int a, int b) {
        return a / b;
    }

    public int multiply(int a, int b) {
        return a * b;
    }
}
```

Business Logic (App)

throws ArithmeticException (Optional)

```java
class Driver {
    Run | Debug
    public static void main(String[] args) {
        Calculator calc = new Calculator();

        Scanner scn = new Scanner(System.in);
        int a = 0, b = 0;
        try {
            a = scn.nextInt();
            b = scn.nextInt();

            try {
                int res = calc.divide(a, b);
                System.out.println(res);
            } catch (ArithmeticException e) {
                ExceptionHandling.calculatorException();
            }

        } catch (Exception e) {
            ExceptionHandling.inputOutputException();
        }

    }
}
```

Driver Code (Client side)

```java
class ExceptionHandling {
    public static void inputOutputException() {
        System.out.println(x: "Please Provide Input Again");
    }

    public static void calculatorException() {
        System.out.println(x: "Division by 0 Not Allowed");
    }
}
```

Except Handler Servic

If your funcn where Runtime/Unchecked Excepn is occured does not handle it, it will be automatically thrown to the calling function.

If your funcn where checked/compiletime Excepn is occured does not handle it, it have to explicitly tell the caller funcn that I can throw a Excepn by using throws keyword.

checked / compile-time

```java
class FileInputOutput {
    public static void fileRead(String path) throws FileNotFoundException {
        // Checked / Compile Time Exception
        FileInputStream file = new FileInputStream(path);
        fileWrite(file);
    }

    public static void fileWrite(FileInputStream file) {
        System.out.println(x: "Performs Some Task on the File");
    }
}
```

```java
class Driver2 {
    Run | Debug
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        String path = scn.nextLine();

        try {
            FileInputOutput.fileRead(path);
        } catch (FileNotFoundException e) {
            System.out.println(x: "Wrong Path Passed");
        }
    }
}
```

```
architaggarwal@Archits-MacBook-Air Java Advanced % javac A_06_ExceptionHandlingPart2.java
architaggarwal@Archits-MacBook-Air Java Advanced % java Driver2
d://abc.txt
Wrong Path Passed
architaggarwal@Archits-MacBook-Air Java Advanced % javac A_06_ExceptionHandlingPart2.java
architaggarwal@Archits-MacBook-Air Java Advanced % java Driver2
/Users/architaggarwal/Documents/Demo.txt
Performs Some Task on the File
architaggarwal@Archits-MacBook-Air Java Advanced %
```

```
vote( int age ){
    if (age < 18)
        throw Excep^n

    Syso( " voted successfully");
}
```

Voting app

```java
// Custom Exception: Unchecked
class AgeInvalidException extends RuntimeException {
    public AgeInvalidException() {
        super(message: "Age Is Invalid");
    }

    public AgeInvalidException(String message) {
        super(message);
    }
}

class VotingApp {
    public static void vote(int age) {
        if (age < 18)
            throw new AgeInvalidException();

        System.out.println(x: "Voted Successfully");
    }
}
```

```java
class Driver3 {
    Run | Debug
    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int age = scn.nextInt();

        try {
            VotingApp.vote(age);
        } catch (AgeInvalidException e) {
            System.out.println(e);
        }
    }
}
```

```
architaggarwal@Archits-MacBook-Air Java Advanced % javac A_06_ExceptionHandlingPart2.java
architaggarwal@Archits-MacBook-Air Java Advanced % java Driver3
15
AgeInvalidException: Age Is Invalid
architaggarwal@Archits-MacBook-Air Java Advanced % java Driver3
25
Voted Successfully
```

Difference Between Checked Exception and Unchecked Exception

Checked Exception / Compile Time Exception	Unchecked Exception / Runtime Exception
1. Checked Exceptions are the exceptions that are checked and handled at compile time.	1. Unchecked Exceptions are the exceptions that are not checked at compiled time.
2. The program gives a compilation error if a method throws a checked exception.	2. The program compiles fine because the compiler is not able to check the exception.
3. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword.	3. A method is not forced by compiler to declare the unchecked exceptions thrown by its implementation. Generally, such methods almost always do not declare them, as well.
4. A checked exceptions occur when the chances of failure are too high.	4. Unchecked exception occurs mostly due to programming mistakes.
5. They are direct subclass of Exception class but do not inherit from RuntimeException.	5. They are direct subclass of RuntimeException class.

final

keyword

1. const variable
2. final class cannot be extend
3. final method cannot be overided.

finally

block

clean up code
(memory release, I/O stream close)

finalize

method (of Object class)

Destructor
(release resources from an object to be deallocate)

Difference between throw and throws keyword

throw keyword	throws keyword
1. throw keyword is used to create an exception object manually i.e. by programmer (otherwise by default method is responsible to create exception object)	1. throws keyword is used to declare the exceptions i.e. it indicate the caller method that given type of exception can occur so you have to handle it while calling.
2. throw keyword is mainly used for runtime exceptions or unchecked exceptions	2. throws keyword is mainly used for compile time exceptions or checked exceptions
3. In case of throw keyword we can throw only single exception	3. In case of throws keyword we can declare multiple exceptions i.e. `void readFile() throws FileNotFoundException, NullPointerException, etc.`
4. throw keyword is used within the method	4. throws keyword is used with method signature
5. throw keyword is followed by new instance	5. throws keyword is followed by class
6. We cannot write any statement after throw keyword and thus it can be used to break the statement	6. throws keyword does not have any such rule

Memory Allocation in C/C++

1) life & scope of primitive variables & reference variables - (same for C++ & Java)

2) life & scope of Array / String literals / objects.

3) (Dynamic Memory Allocatn) → new / malloc
& delete (dynamic memory Allocatn) → delete / free } in C++/C

4) Memory leaks in C/C++ (consumed memory & free memory)

5) Deadlocks & Starvation in C/C++

6) Constructors & Destructors in C++
(new) (delete)

Garbage Collection in Java

1) Automatic Garbage Collection → Garbage collector

2) no delete keyword, no destructors in Java!

3) Unreferenced objects → garbage collectible/dead or abandoned objects

4) JVM → daemon thread → garbage collector
- main thread (user thread)
- Thread Scheduler (daemon/background method)

5) On the spot OR cummulative ?
↳ on memory fragmentation!

6) Garbage Collector Working

finalize() method

(1) Similar to destructor in C++

(2) Runs before object is about to destroy

(3) Present in Object class → can be overided to provide clean up of System resources (eg database, file, I/Ostream, hardware, etc)

(4) Called automatically only once by JVM (Garbage Collector thread)

(5) Custom calling will only lead to clean up & not object destruction/memory allocation.

c) Exceptions in finalize method are ignored/not propogated by garbage collector thread

d) There is no chaining of finalize() method implicitly like for constructor chaining in Java and destructor chaining in C++.

Forceful execution of finalize() method

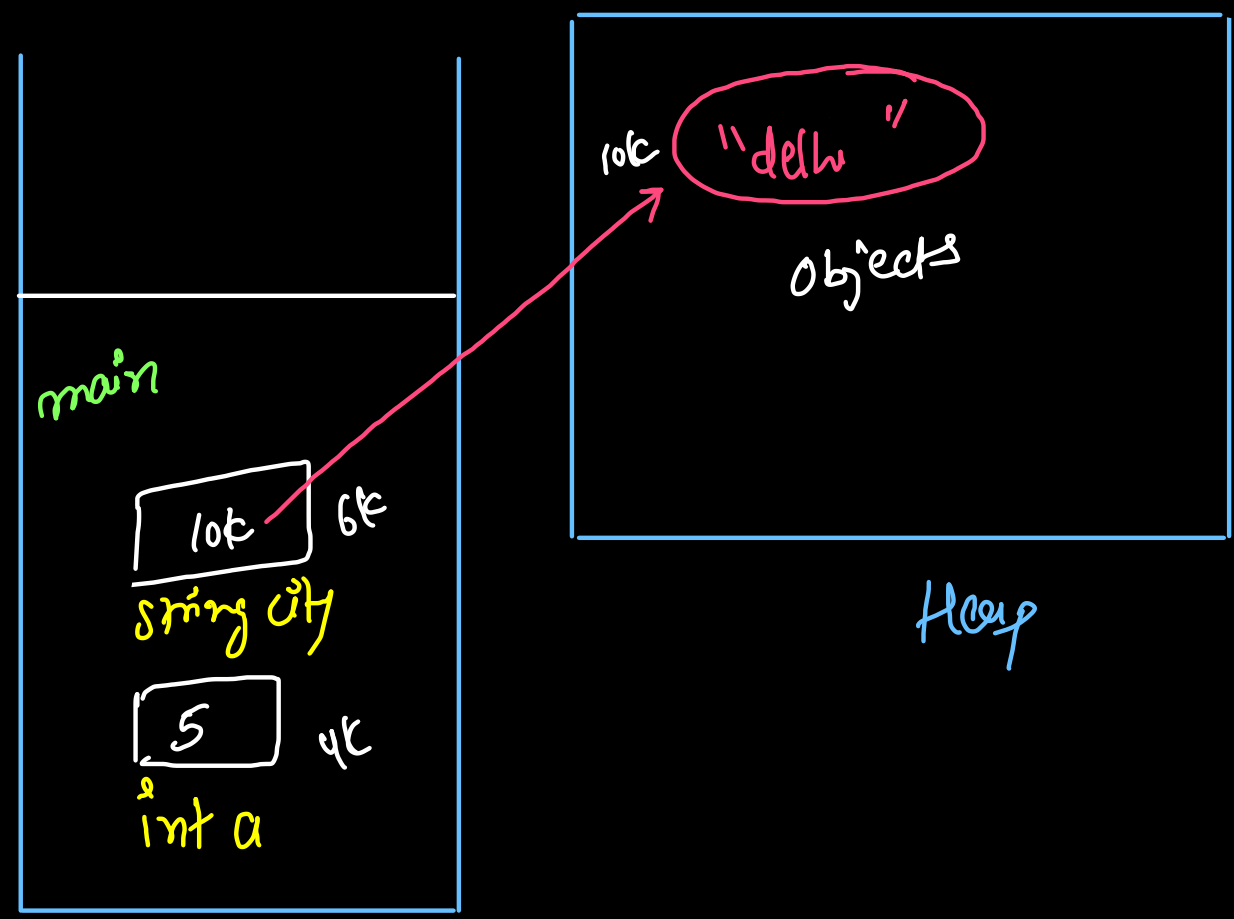
- Runtime.getRuntime().gc()
- System.gc()

Memory Allocation of Primitive / Reference Variables

* Stack
  - Primitive Variables
  - Reference Variables

Scope :→ Functn / Block scope

Lifetime :→ Function Push in Stack
↳ create / Allocate
Functn Pop in Stack
↳ Deallocate / Release



```java
public static void fun() {
    System.out.println(x: "Primitive & Reference Variables Creation / Allocation");
    // Function Scope Local Variables:
    int a = 5;
    String str = "hello";

    for (int idx = 0; idx < 5; idx++) {
        // Index is accessible within for loop only
        System.out.print(idx + " ");
    }

    if (a % 2 == 0) {
        String even = "even";
        System.out.println(even);
    } else {
        String odd = "odd";
        System.out.println(odd);
    }

    System.out.println(x: "Primitive & Reference Variables Deletion / Deallocation");
}
```

```
architaggarwal@Archits-MacBook-Air 02. Core Java Advanced % javac 07.GarbageCollection.java
architaggarwal@Archits-MacBook-Air 02. Core Java Advanced % java Solution
Primitive & Reference Variables Creation / Allocation
0 1 2 3 4 odd
Primitive & Reference Variables Deletion / Deallocation
```

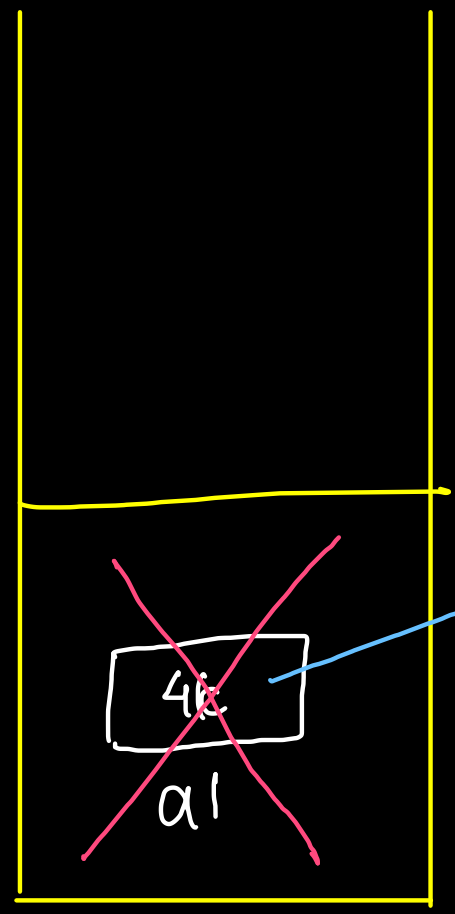
Memory Allocation of Heap Variables

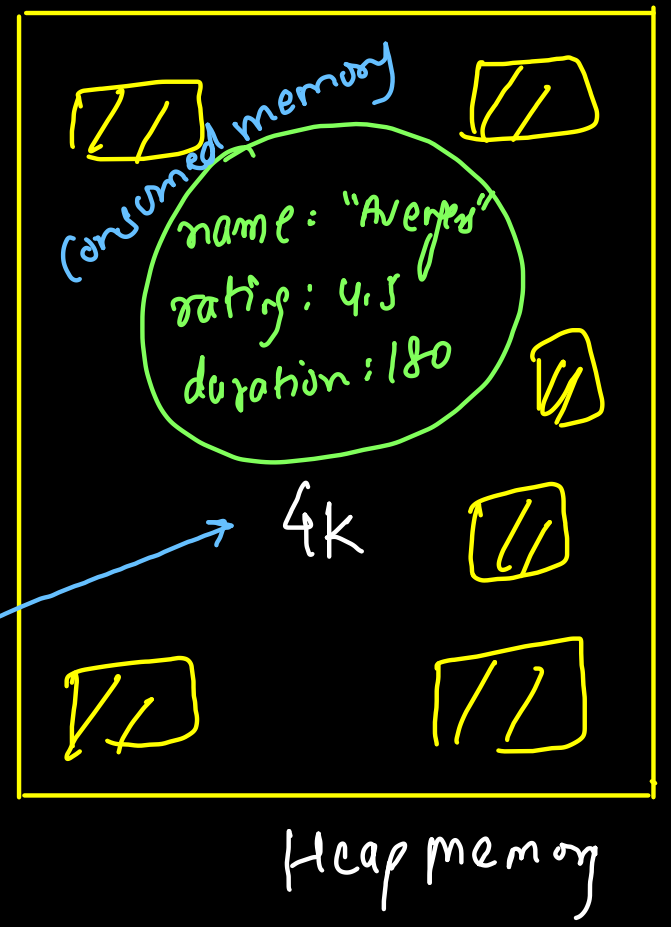
C/C++

Movie a1 = new Movie(); (Constructor)

fragmented memory
↳ memory leak → on Heap
- Developers mistakes
- Objects were not deallocated from heap

delete a1; → explicit deallocation
↳ destructor





Movie Ticketing App

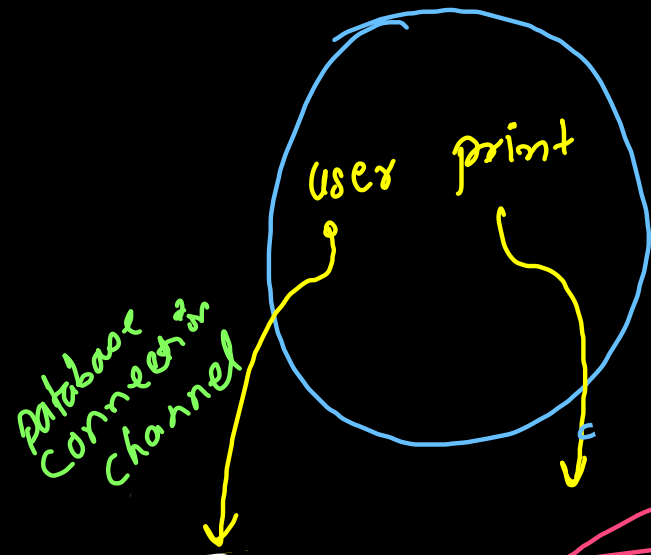
memory Deallocation (&)

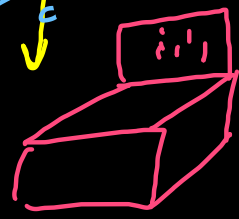
- Hardware Resources
- Database Connection

} Decoupling &

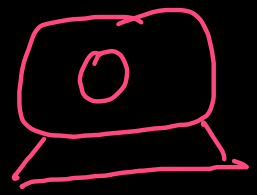
Starvation Deadlock

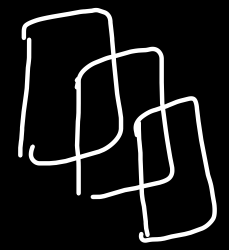
for other objects/applications!





hardware resources





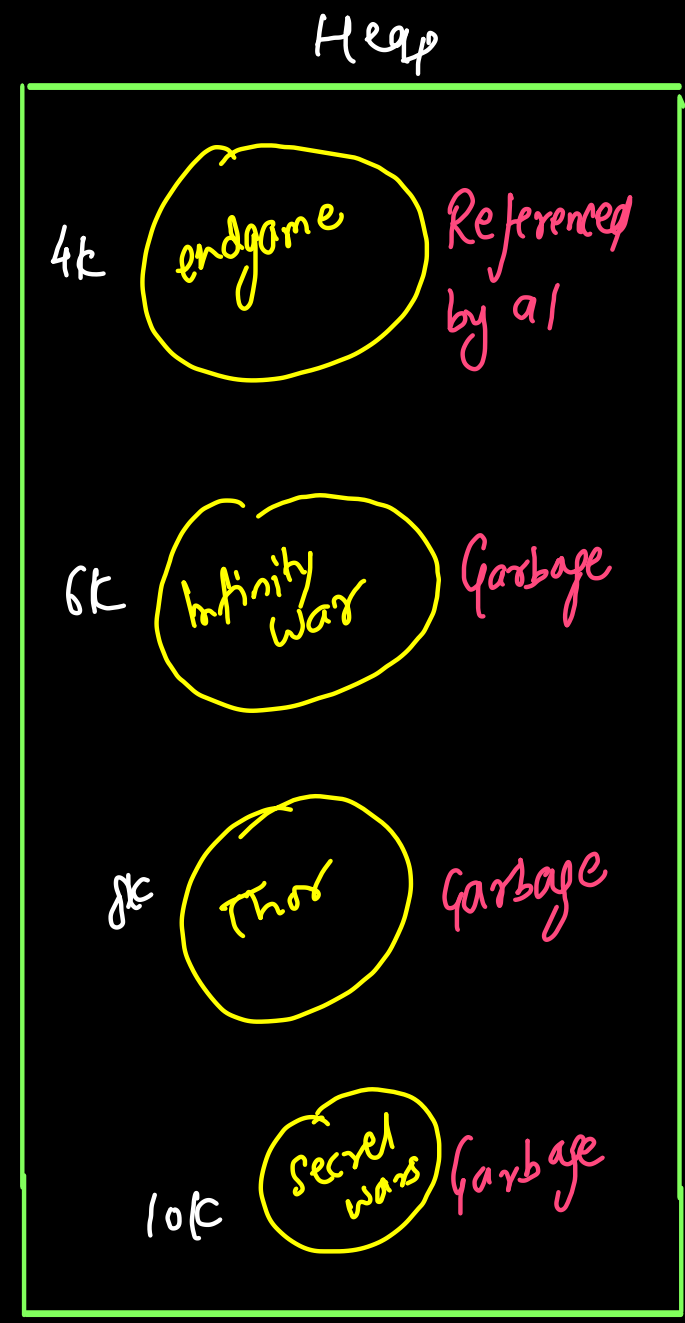
I/O stream

Automatic Garbage Collection in Java.

Stack

a1 = 4k a2 = ~~6k~~ null a3 = ~~8k~~ 9k

```java
public static void automaticGarbageCollection() {
    Movie a1 = new Movie(duration: 180, name: " Endgame", rating: 4.5);
    // Object cannot be deleted because it is referenced

    Movie a2 = new Movie(duration: 150, name: "Infinity War", rating: 4.2);
    a2 = null; // 1. Nulling the Reference

    Movie a3 = new Movie(duration: 120, name: "Thor", rating: 2.5);
    a3 = a1; // 2. Updating the Reference

    new Movie(duration: 120, name: "Secret Wars", rating: 2.5);
    // 3. Anonymous Object
}
```



```java
class Movie {
    static int countCreation = 0;
    static int countDeletion = 0;
    int duration;
    String name;
    double rating;

    public Movie(int duration, String name, double rating) {
        System.out.println(x: "Memory Allocation - Initialization of Variables");
        countCreation++;
        this.duration = duration;
        this.name = name;
        this.rating = rating;
    }

    @Override
    public void finalize() throws Throwable {
        System.out.println(x: "Memory Deallocation");
        countDeletion++;
    }
}
```

```java
public static void automaticGarbageCollection() {
    Movie a1 = new Movie(duration: 180, name: " Endgame", rating: 4.5);
    // Object cannot be deleted because it is referenced

    Movie a2 = new Movie(duration: 150, name: "Infinity War", rating: 4.2);
    a2 = null; // 1. Nulling the Reference

    Movie a3 = new Movie(duration: 120, name: "Thor", rating: 2.5);
    a3 = a1; // 2. Updating the Reference

    new Movie(duration: 120, name: "Secret Wars", rating: 2.5);
    // 3. Anonymous Object

    // Garbage Collector: Memory Deallocation:
    // Automatic, Cummulative, On Memory Leak Avoidance
    for (int idx = 0; idx < 1000000; idx++) {
        new Movie(duration: 120, name: "Temp", rating: 2.5);
    }

    System.out.println(Movie.countCreation + " " + Movie.countDeletion);
}
```

≈ 10 lakh

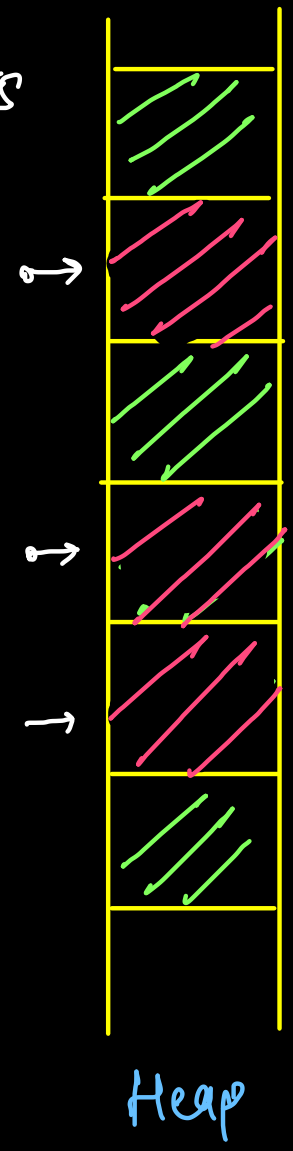
≈ 5.25 lakh

Garbage Collector Working

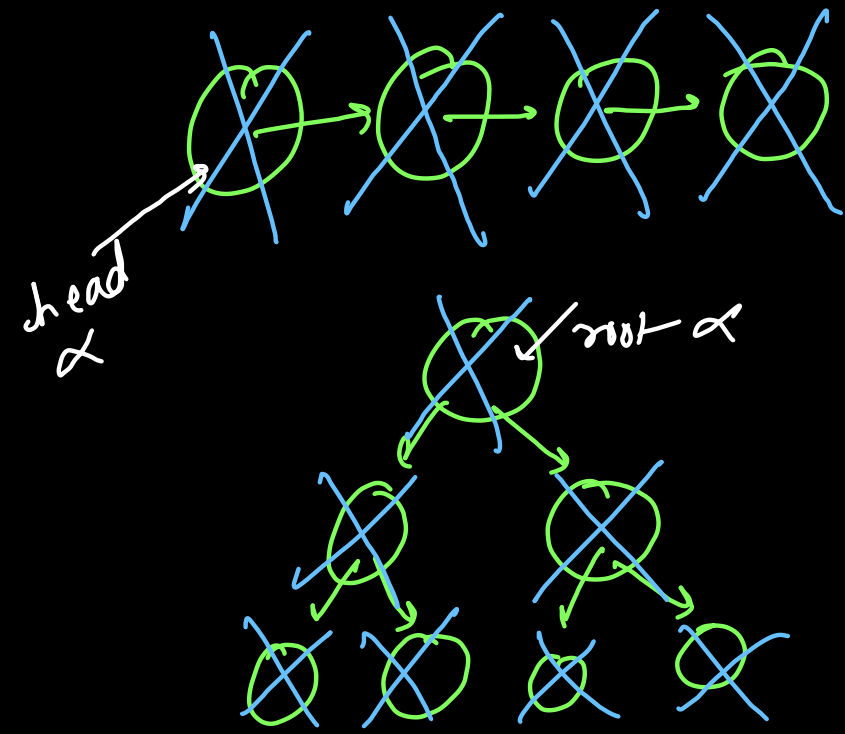
i) Marking the unreachable objects

ii) Destroying them : free the memory

iii) Compaction of memory

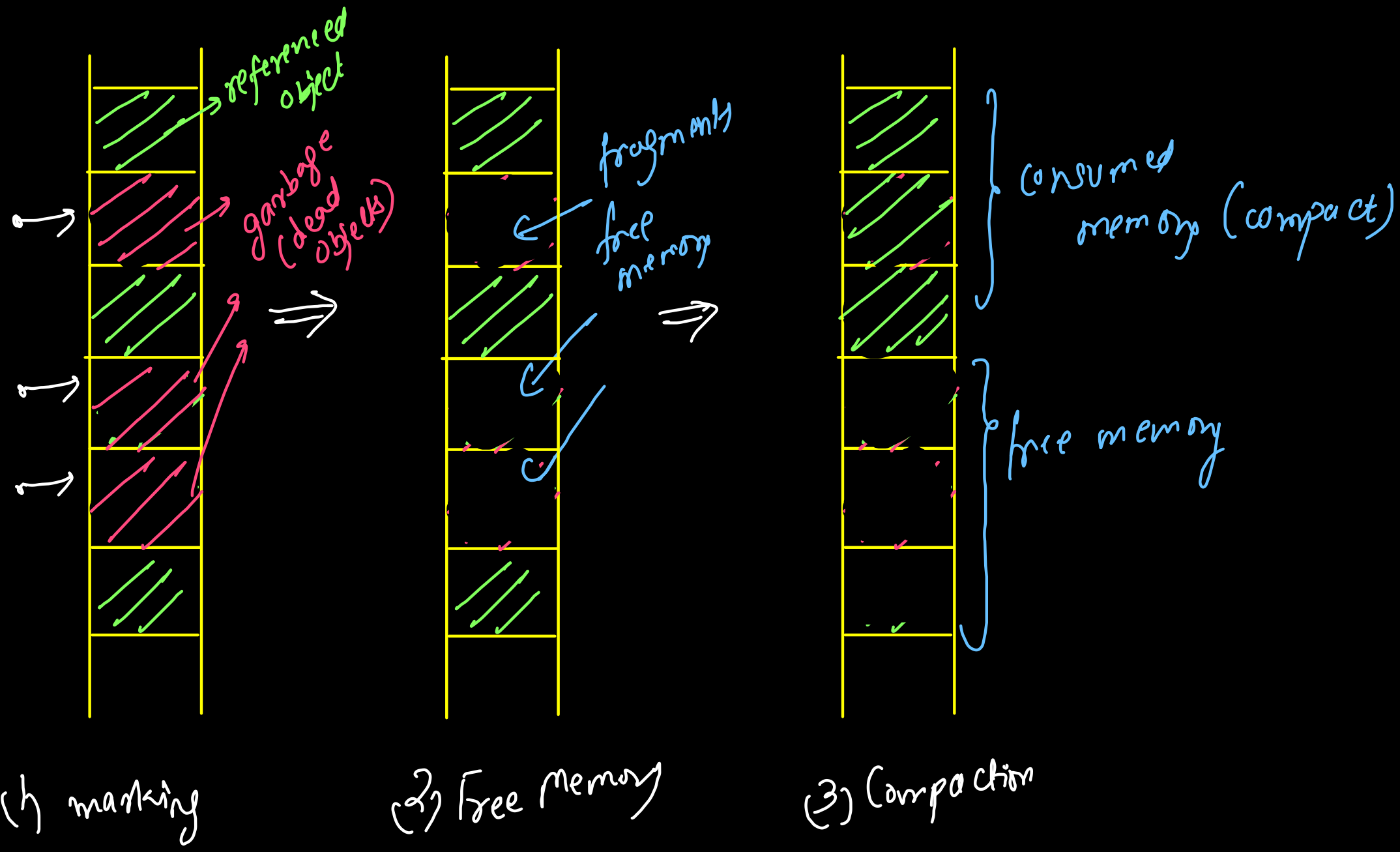


(1) GC roots



= Referenced

= Unreferenced


referenced object
garbage (dead objects)
fragmants
free memory
consumed memory (compact)
free memory
(1) marking
(2) Free memory
(3) Compaction

```java
class Movie {
    static int countCreation = 0;
    static int countDeletion = 0;
    int duration;
    String name;
    double rating;
    FileInputStream ticket;

    public Movie(int duration, String name, double rating) throws Exception {
        System.out.println(x: "Memory Allocation - Initialization of Variables");
        countCreation++;

        this.duration = duration;
        this.name = name;
        this.rating = rating;

        // System Resources Couple
        ticket = new FileInputStream(name: "/Users/architaggarwal/Documents/DSA-And-LLD/LLD or OOAD/ticekt.txt");
    }

    @Override
    public void finalize() throws Throwable {
        System.out.println(x: "Memory Deallocation");
        // System Resources Decouple: Clean Up Code
        ticket.close();
        countDeletion++;
    }
}
```

```java
public static void finalizeDemo() throws Exception {
    Movie a = new Movie(duration: 180, name: " Endgame", rating: 4.5);
    a.finalize();
}
```

↗ object Create

↳ object Deletion (assume)

```java
class Movie {
    static int countCreation = 0;
    static int countDeletion = 0;
    int duration;
    String name;
    double rating;
    Scanner scn;

    public Movie(int duration, String name, double rating) throws Exception {
        System.out.println(x: "Memory Allocation - Initialization of Variables");
        countCreation++;

        this.duration = duration;
        this.name = name;
        this.rating = rating;
      You, 1 second ago • Uncommitted changes
        // System Resources Couple
        scn = new Scanner(System.in);
    }

    @Override
    public void finalize() throws Throwable {
        System.out.println(x: "Memory Deallocation");
        // System Resources Decouple: Clean Up Code
        scn.close();
        countDeletion++;
    }
}
```

Last code before Object deallocation

↳ clean up code

↳ System resources release

```java
public static void finalizeDemo() throws Exception {
    Movie a = new Movie(duration: 180, name: " Endgame", rating: 4.5);
    try {
        a.finalize();
    } catch (Throwable e) {
    }
}
```

Driver

```
architaggarwal@Archits-MacBook-Air 02. Core Java Advanced % javac 07.GarbageCollection.java
Note: 07.GarbageCollection.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
architaggarwal@Archits-MacBook-Air 02. Core Java Advanced % java Solution
Memory Allocation - Initialization of Variables
Memory Deallocation
```

Output

```java
public static void finalizeDemo() throws Exception {
    Movie a = new Movie(duration: 180, name: " Endgame", rating: 4.5);
    try {
        a.finalize();
    } catch (Throwable e) {
    }

    System.out.println(a); // It will still print the object: Object is still there in memory
    // Object Deallocation due to Developer's finalize Call is not happening      You, 1 secon
}
```

```
architaggarwal@Archits-MacBook-Air 02. Core Java Advanced % java Solution
Memory Allocation - Initialization of Variables
Clean Up Code
Movie@5a39699c
```

Level 1 (Parent)

```
constructor() {
    Syso (Parent -> Execute)
}
```

Level 2 (Child)

```
constructor() {
    super();
    Syso(Child -> Execute);
}
```

Level 3 (Grandchild)

```
constructor(){
    super();
    Syso (Grandchild -> Execute)
}
```

```
Grandchild obj = new Grandchild();
```

Invokam(call)

GC()

↓

C()

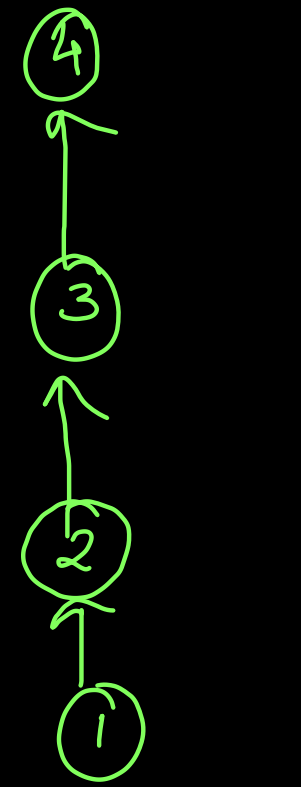
↓

Parent()

↓

Object()

Execution Postorder



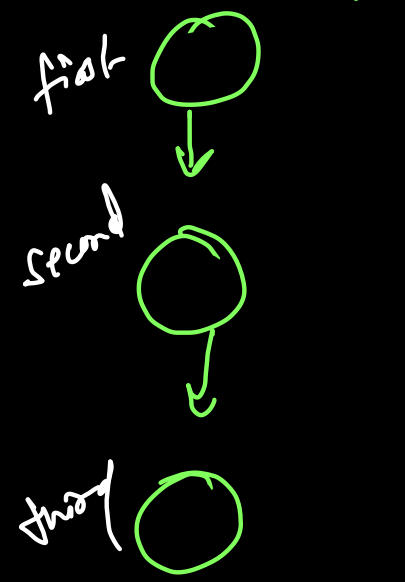
```
Level1(Parent)
finalize(){
    Syso("P -> cleanup code");
}

Level 2 (Child)
finalize(){
    Syso("C -> cleanup code");
    super.finalize();
}

Level 3 (GrandChild)
finalize(){
    Syso("GC -> cleanup code");
    super.finalize();
}
```

`GrandChildObj`.finalize();
{ Explicitly }

Invokation (Call)

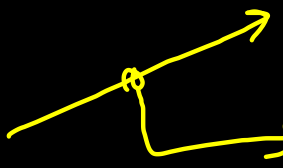
GC()
↓
C()
↓
Parent()

Execution (Preorder)



Forceful Execution of object Destructn or calling of Garbage Collectn

(1) System.gc();



(2) Runtime.getRuntime().gc();

optional for JVM to accept your GC thread. It can also refuse/decline them.

① final vs finally vs finalize

② finalize vs System.gc()

```java
public static void gcDemo() throws Exception {
    Movie a1 = new Movie(duration: 180, name: " Endgame", rating: 4.5);
    // Object cannot be deleted because it is referenced

    Movie a2 = new Movie(duration: 150, name: "Infinity War", rating: 4.2);
    a2 = null; // 1. Nulling the Reference

    Runtime.getRuntime().gc();

    Movie a3 = new Movie(duration: 120, name: "Thor", rating: 2.5);
    a3 = a1; // 2. Updating the Reference

    // Forceful Execution of Garbage Collection
    System.gc();

}
```

```
Initialization of Variables
Initialization of Variables
Initialization of Variables
Clean Up Code
Memory Deallocation
Clean Up Code
Memory Deallocation
```

Enums in Java

1) Creating interrelated constants in Java
   - old way :→ Interfaces or abstract class
   - new way :→ Enums
     - Outside any class
     - Inside class (inner enum)

2) Variables are → public, final, static, objects of same type
   - Internal implementation of Enums → class

3) Constructors
   - Empty parameter/default Constructor
   - Parameterized Constructor

4) Getters & other functions can be there

5) Enum parent class
- Extended by every userdefined enum.
- Implements Comparable & Serialize
- ordinal(), values(), name()

6) Enum cannot extend any other class
But Enum can implement other interfaces

Applications
- → Switch case
- → for each loop over enum!

eg levels, Days, Colors, Languages, Movie Type, etc

abstract / interface

```
class Genre {
    static final String ACTION = "Action";
    static final String ROMANCE = "Romance";
    static final String COMEDY = "Comedy";
}
```

Old fashioned way

```
enum ScreenType {
    TWOD, THREED, IMAX3D, FOURDX;
}

class Movie {
    String genre = Genre.ACTION;
    ScreenType type = ScreenType.THREED;
}
```

public, final, static

New way

```
class Movie {
    String genre = Genre.ACTION;
}

class Solution {
    Run | Debug
    public static void main(String[] args) {
        Movie obj = new Movie();

        switch (obj.genre) {
            case Genre.ACTION: {
                System.out.println(x: "Nerds will watch this movie");
                break;
            }
            case Genre.ROMANCE: {
                System.out.println(x: "Couples will watch this movie");
                break;
            }
            case Genre.COMEDY: {
                System.out.println(x: "Family will watch it together");
                break;
            }
            default: {
                System.out.println(x: "No Such Genre Exists");
            }
        }
    }
}
```

```java
class ScreenType{
    public static final ScreenType TWOD = new ScreenType();
    public static final ScreenType THREED = new ScreenType();
    public static final ScreenType IMAX3D = new ScreenType();
    public static final ScreenType FOURDX = new ScreenType();
}

enum ScreenTypeEnum {
    TWOD, THREED, IMAX3D, FOURDX;
}
```

↑ internally implemented

Extendy Enum

↳ Cannot extend any other class

enum extends T ✗

Although Interfaces can be implemented

enum implements T ✓

```java
class ScreenType {
    public static final ScreenType TWOD = new ScreenType();
    public static final ScreenType THREED = new ScreenType();
    public static final ScreenType IMAX3D = new ScreenType();
    public static final ScreenType FOURDX = new ScreenType();

    [redacted]

    public ScreenType() {
        System.out.println(x: "Constructor Called");
    }
}

enum ScreenTypeEnum {
    TWOD, THREED, IMAX3D, FOURDX;

    ScreenTypeEnum() {
        System.out.println(x: "Enum Constructor Called");
    }
}
```

```java
class Movie {
    String genre = Genre.ACTION;
    ScreenType stype = ScreenType.THREED;
}

class Solution {
    public static void oldFashioned() {…

    Run | Debug
    public static void main(String[] args) {
        Movie obj = new Movie();
        System.out.println(obj.stype);

        ScreenTypeEnum obj2 = ScreenTypeEnum.TWOD;
    }
}
```

```
architaggarwal@Archits-MacBook-Air 02. Core Java Advanced % java Solution
Constructor Called
Constructor Called
Constructor Called
Constructor Called
ScreenType@442d9b6e
Enum Constructor Called
Enum Constructor Called
Enum Constructor Called
Enum Constructor Called
```

```java
class ScreenType {
    public static final ScreenType TWOD = new ScreenType(price: 250);
    public static final ScreenType THREED = new ScreenType(price: 300);
    public static final ScreenType IMAX3D = new ScreenType(price: 400);
    public static final ScreenType FOURDX = new ScreenType(price: 600);

    int price;

    public ScreenType() {
        System.out.println(x: "Constructor Called");
        price = 100;
    }

    public ScreenType(int price) {
        this.price = price;
    }
}
```

```java
enum ScreenTypeEnum {
    TWOD(price: 250), THREED(price: 300), IMAX3D(price: 400), FOURDX(price: 600);

    int price;

    ScreenTypeEnum() {
        System.out.println(x: "Enum Constructor Called");
        price = 100;
    }

    ScreenTypeEnum(int price) {
        this.price = price;
    }
}
```

```java
public static void main(String[] args) {
    ScreenType obj = ScreenType.THREED;
    System.out.println(obj);→hashcode

    ScreenTypeEnum obj2 = ScreenTypeEnum.TWOD;

    System.out.println(obj.price);→ 300
    System.out.println(obj2.price);→ 250
}
```

```
System.out.println(ScreenTypeEnum.TWOD.ordinal());
System.out.println(ScreenTypeEnum.THREED.ordinal());
System.out.println(ScreenTypeEnum.IMAX3D.ordinal());
System.out.println(ScreenTypeEnum.FOURDX.ordinal());
```

→ 0
→ 1
→ 2
→ 3

ordinal property → Enum Class (Parent class)

of enum

↳ to uniquely identify each Constant

```
 */
@SuppressWarnings("serial") // No serialVersionUID needed due to
                            // special-casing of enum classes.
public abstract class Enum<E extends Enum<E>>
        implements Constable, Comparable<E>, Serializable {
    /**
     * The name of this enum constant, as declared in the enum declaration.
     * Most programmers should use the {@link #toString} method rather than
     * accessing this field.
     */
    private final String name;

    /**…
    public final String name() {
        return name;
    }

    /**…
    private final int ordinal;

    /**…
    public final int ordinal() {
        return ordinal;
    }
```

String

name

```
System.out.println(ScreenTypeEnum.TWOD.name());
System.out.println(ScreenTypeEnum.THREED.name());
System.out.println(ScreenTypeEnum.IMAX3D.name());
System.out.println(ScreenTypeEnum.FOURDX.name());
```

→ TWOD
→ THREED
→ IMAX3D
→ FOURDX

values() List<ScreenTypeEnum>

```
// Looping on Constants of a enum
for (ScreenTypeEnum c : ScreenTypeEnum.values()) {
    System.out.print(c.price + " ");
}
```

→ 250 300 400 600

Java Annotations

→ metadata (data about data) of source code (.java file)

- → additional info about program for compiler
- → does not effect execution of compiled program
- → not mandatory to use but recommended for compile-time checks of developer mistakes & readability!

Types of Annotations

1. Marker Annotations → no method
2. Single value Annotations → one attribute
3. multi value Annotations → more than one attributes

Types of Annotations

1. **Predefined annotations**
   1. `@Deprecated`
   2. `@Override`
   3. `@SuppressWarnings`
   4. `@SafeVarargs`
   5. `@FunctionalInterface`
2. **Meta-annotations**
   1. `@Retention`
   2. `@Documented`
   3. `@Target`
   4. `@Inherited`
   5. `@Repeatable`
3. **Custom annotations**

Use of Annotations

- **Compiler instructions** - Annotations can be used for giving instructions to the compiler, detect errors or suppress warnings. The built-in annotations `@Deprecated`, `@Override`, `@SuppressWarnings` are used for these purposes.
- **Compile-time instructions** - Compile-time instructions provided by these annotations help the software build tools to generate code, XML files and many more.
- **Runtime instructions** - Some annotations can be defined to give instructions to the program at runtime. These annotations are accessed using Java Reflection.

```java
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@interface MyAnno {

    int myValue() default 0;

    String name() default "Durgesh";

    String city() default "DELHI";

}
```

```java
class KeypadPhone {
    @Deprecated
    public static void touchKeypad() {
        System.out.println(x: "Keypad Phone");
    }
}
```

```java
KeypadPhone.touchKeypad();
// Warning Issued: Not recommended to use now
```

```
architaggarwal@Archits-MacBook-Air 01. Core Java Basics % javac Annotations.java
Note: Annotations.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
```

```java
Integer ii = new Integer(value: 5);
System.out.println(ii);
```

```
Annotations.java:38: warning: [removal] Integer(int) in Integer has been deprecated and marked for removal
        Integer ii = new Integer(5);
                         ^
```

```java
class Parent {
    String emailIDAddress;

    public String getEmailIDAddress() {
        System.out.println(x: "Overridden Method");
        return emailIDAddress;
    }
}

class Child extends Parent {

    // @Override
    // public String getEmailIdAddress() {
    // System.out.println("Overrided Method");
    // return emailIDAddress;
    // }

    @Override
    public String getEmailIDAddress() {
        System.out.println(x: "Overrided Method");
        return emailIDAddress;
    }
}
```

Parent obj = new child();
obj.getEmailIDAddress();

without @Override
↳ Parent's method

with @ Override
↳ compilation error

with/without @override
↳ Child's method

```java
Integer ii = new Integer(value: 5);
System.out.println(ii);

ArrayList arr = new ArrayList();

// Adjacency List (Graphs)
ArrayList<Integer>[] adj = new ArrayList[5];
for (int idx = 0; idx < 5; idx++) {
    adj[idx] = new ArrayList<>();
}
```

autobox only value of should be used

ArrayList is raw type. Make it bounded <>

Some portals will not compile if warnings!

Unchecked warnings (IMP) → online assessment

```
Note: Annotations.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
1 warning
```

```java
@SuppressWarnings("all")
Run | Debug
public static void main(String[] args) {
    KeypadPhone.touchKeypad();
    // Warning Issued: Not recommended to use now

    Integer ii = new Integer(value: 5);
    System.out.println(ii);

    ArrayList arr = new ArrayList();

    // Adjacency List (Graphs)
    ArrayList<Integer>[] adj = new ArrayList[5];
    for (int idx = 0; idx < 5; idx++) {
        adj[idx] = new ArrayList<>();
    }
```

Custom Annotations

(1) marker annotation Syntax :→ `@interface MyAnnotation { }`

(2) single value annotation

Syntax :→

```
@interface MyAnnotation{
    int value() default 0;
}
```

Application :→

```
@MyAnnotation(value=10)
```

(3) multi-value annotation

```
@interface MyAnnotation{
int value1() default 1;
String value2() default "";
String value3() default "xyz";
}
```

Application

```
@MyAnnotation(value1=5,
              value2="abc");
```

Other ways of Object Creation : Java Reflection API

newInstance() { method in Class class of java.lang package }

Syntax ① className obj = (className)

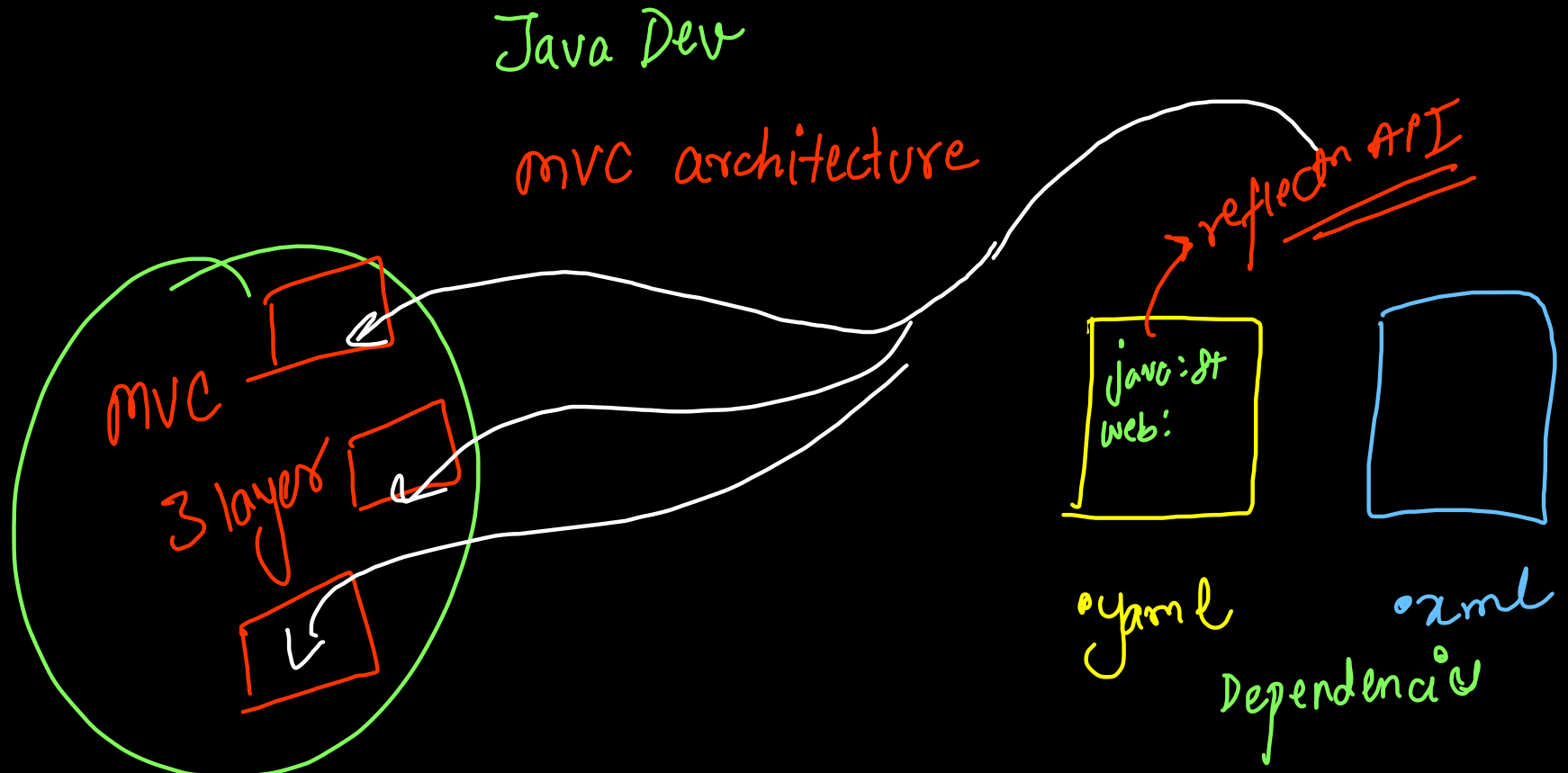
Class.forName("fullyqualifiedclassName").newInstance();

↳ class loading (checked exception)

② className obj = className.class.newInstance();

Explore More : for Spring Frameworks, testing java codes (private data access)

, configuration files (.xml or .yaml file)


Java Dev
mvc architecture
mvc
3 layer
reflection API
java:8f
web:
yaml
xml
Dependency

```java
class Driver {
    Run | Debug
    public static void main(String[] args) {
        // Demo obj = new Demo();

        try {
            Driver obj = (Driver) Class.forName(className: "Driver").newInstance();
            System.out.println(obj);
        } catch (Exception e) {
            System.out.println(x: "Class Not Found");
        }
    }
}
```

checked exception

typecasting

class class

fully qualified

deprecated

```
architaggarwal@Archits-MacBook-Air 02. Core Java Advanced % java Driver
Driver@368102c8
```

Shorter syntax :→

```java
Driver obj2 = Driver.class.newInstance();
System.out.println(obj2);
```

(2) Object Cloning

→ marker interface (no methods)

```
class Movie implements Cloneable {
    String name = "Avengers Endgame";
    int duration = 180;
    double ratings = 4.5;

    public Movie() {…

    public Movie(String name, int duration, double ratings) {…

    @Override
    protected Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
```

```
public static void cloneDemo() throws Exception {
    Movie a1 = new Movie(name: "Avengers Endgame", duration: 180, ratings: 4.9);
    System.out.println(a1);

    Movie a2 = (Movie) (a1.clone());
    System.out.println(a2);
}
```

Advantage → easy, readable/short, fast/efficient

Disadvantage → (1) It must be properly chained in inheritance otherwise it will fail

→ shallow copy by default!

→ object create w/o constructor call/new keyword

```java
class Movie implements Cloneable {
    String name = "Avengers Endgame";
    int duration = 180;
    double ratings = 4.5;

    public Movie() {…

    public Movie(String name, int duration, double ratings) {…

    @Override
    protected Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
```

```java
public static void cloneDemo() throws Exception {
    Movie a1 = new Movie(name: "Avengers Endgame", duration: 180, ratings: 4.9);
    System.out.println(a1);

    Movie a2 = (Movie) (a1.clone());
    System.out.println(a2);

    System.out.println(a1 == a2);
    System.out.println(a1.name == a2.name);
}
```

You, 2 weeks ago • Added Bubble Sort

→ false (different objects)

→ true (Shallow copy)

```
Name : Avengers Endgame , Duration : 180 , Ratings : 4.9
Name : Avengers Endgame , Duration : 180 , Ratings : 4.9
false
true
```

Nested classes in Java

- Member Inner class
  - ↳ Create objects of inner class using object of outer class only.
- Static Nested class
  - ↳ Create objects of inner class w/o outer class object
- Anonymous Inner class
  - ↳ no name for overriding some methods.
- local (method) inner class

Advantages

1. grouping inter-related classes
2. Readable, maintable code → easy to debug.
3. reduced scope (within one class → private)
   - ↳ Increased Encapsulation!

Static, nonstatic variables

outer → inner ?

inner → outer ?

```java
class Outer {
    private String outerData = "Outer";

    public void setOuterData(String outerData) {
        this.outerData = outerData;
    }

    public String getOuterData() {
        return outerData;
    }

    class Inner {
        String innerData = "Inner";

        public String getInnerData() {
            return innerData;
        }

        public void setInnerData(String innerData) {
            this.innerData = innerData;
        }
    }
}
```

```java
class Driver {
    Run | Debug
    public static void main(String[] args) {
        Outer outerobj = new Outer();
        System.out.println(outerobj.getOuterData());

        Outer.Inner innerobj = outerobj.new Inner();
        System.out.println(innerobj.innerData);
    }
}
```

Outer

Inner

```java
class Outer {
    private String outerData = "Outer";

    public String getOuterData() {…

    public static String outerStaticData = "OuterStatic";

    public static void outerFun() {
        System.out.println(Inner.innerStaticData);
        System.out.println(outerStaticData);
    }

    class Inner {
        String innerData = "Inner";

        public String getInnerData() {…

        public static String innerStaticData = "InnerStatic";

        public static void innerFun() {
            System.out.println(innerStaticData);
            System.out.println(outerStaticData);
        }
    }
}
```

```java
class Driver {
    Run | Debug
    public static void main(String[] args) {
        Outer outerobj = new Outer();
        System.out.println(outerobj.getOuterData());

        Outer.Inner innerobj = outerobj.new Inner();
        System.out.println(innerobj.innerData);

        Outer.outerFun();
        Outer.Inner.innerFun();
    }
}
```

Static variables
(if they are public),
they are accessible
everywhere

```java
public void outerFun2() {
    System.out.println(Inner.innerStaticData);
    System.out.println(outerStaticData);
    System.out.println(outerData);
    Syso (innerData);
    Inner obj = new Inner();
    System.out.println(obj.innerData);
}
```

```java
public void innerFun2() {
    System.out.println(innerStaticData);
    System.out.println(outerStaticData);
    System.out.println(innerData);
    System.out.println(outerData);
}
```

```java
class Outer {
    String outerData = "outerData";
    static String outerStaticData = "outerStaticData";

    public void outerFun() {
        System.out.println(outerStaticData + " " + Inner.innerStaticData + " " + outerData);
        // System.out.println(innerData); Not Accessible: You need Inner Class Object
    }

    public static void outerStaticFun() {
        System.out.println(outerStaticData + " " + Inner.innerStaticData);
    }
```

```
outerStaticData innerStaticData innerData
outerStaticData innerStaticData
outerStaticData innerStaticData outerData
outerStaticData innerStaticData
```

```java
static class Inner {
    String innerData = "innerData";
    static String innerStaticData = "innerStaticData";

    public void innerFun() {
        System.out.print(outerStaticData + " " + innerStaticData + " ");
        System.out.println(innerData);
        // System.out.println(outerData);
        // Without Outer Class Object -> Inner Class Object can be made
        // Outer Class Non Static Variables will not be accessible
    }

    public static void innerStaticFun() {
        System.out.println(outerStaticData + " " + innerStaticData);
    }
}
```

```java
class Driver {
    Run | Debug
    public static void main(String[] args) {
        Outer.Inner innerObj = new Outer.Inner();
        innerObj.innerFun();
        Outer.Inner.innerStaticFun();

        Outer outerObj = new Outer();
        outerObj.outerFun();
        Outer.outerStaticFun();

    }
}
```

```java
abstract class Operation {
    abstract int applyOperation(int a, int b);
}

class Sum extends Operation {
    @Override
    int applyOperation(int a, int b) {
        return a + b;
    }
}

class Difference extends Operation {
    @Override
    int applyOperation(int a, int b) {
        return a - b;
    }
}
```

```java
public static void main(String[] args) {
    Operation obj1 = new Sum();
    System.out.println(obj1.applyOperation(a: 5, b: 10));

    Operation obj2 = new Difference();
    System.out.println(obj2.applyOperation(a: 5, b: 10));

    Operation multiply = new Operation() {
        int applyOperation(int a, int b) {
            return a * b;
        }
    };

    System.out.println(multiply.applyOperation(a: 5, b: 10));

    Operation divide = new Operation() {
        int applyOperation(int a, int b) {
            return a / b;
        }
    };

    System.out.println(divide.applyOperation(a: 5, b: 15));
}
```

Has-A relationship in Java

→ type of relationship b/w 2 classes, in which one class have the object of another class as data member.

→ increases reusability and readability/maintainability/debugging. reduces code redundancy.

→ It is also known as "Association". It is a uni-directional relationship, ie one way relationship.

→ When compared to Inheritance, two classes in association are less tightly coupled in comparison to class extending other class.

Inheritance

Parent

↑ extends

Child

drive()

drive with refuel()

Association

Car ——has a——> musicplayer

FM

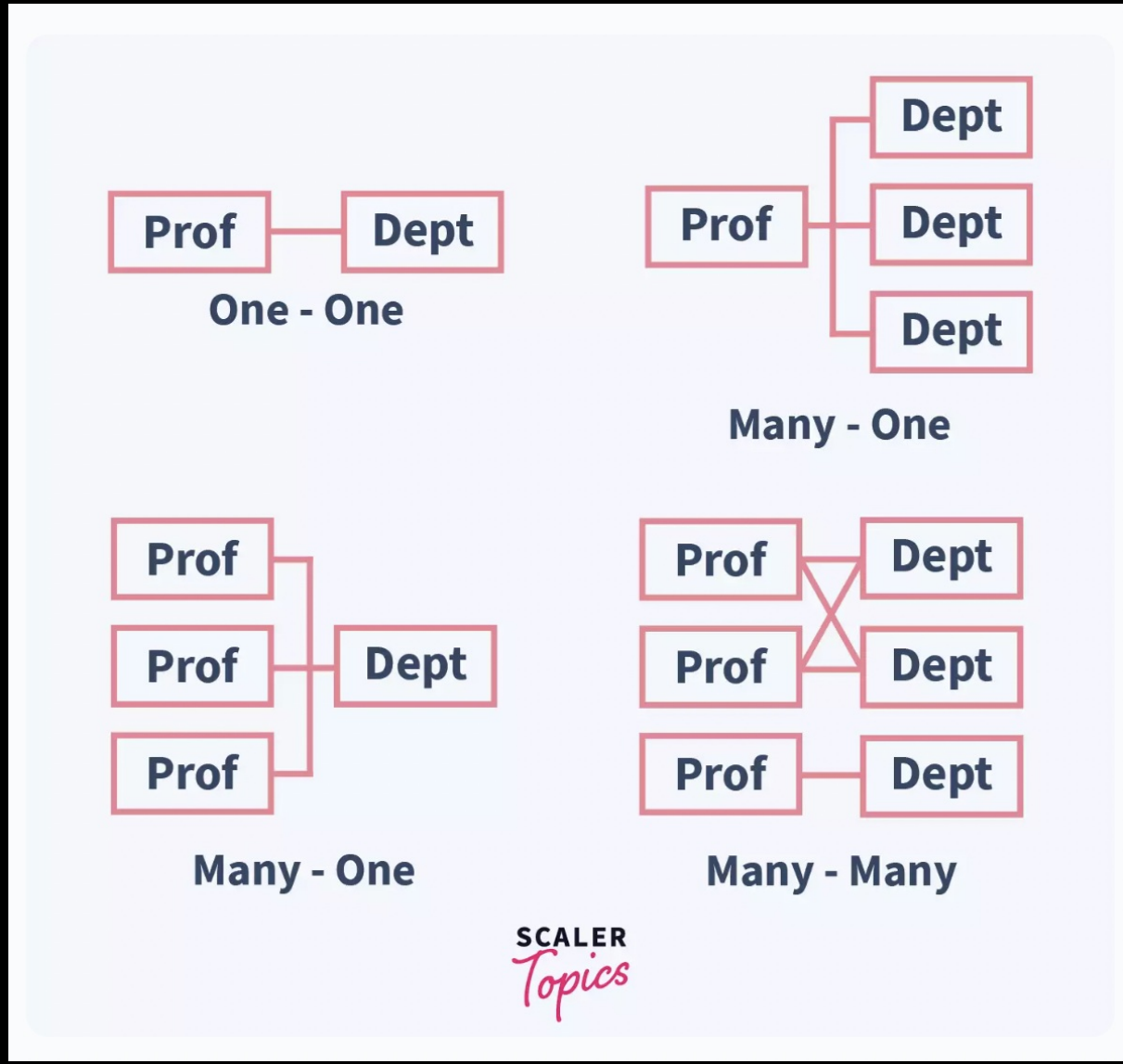
Bluetooth

Coupling

tight coupling
(bloodrelationship)

loose coupling
(not a blood relationship)

Multiplicity It can be 1:1, 1:many, many:1 or many:many association types.



Comparing Inheritance vs Association (or) Is-A relationship with has-A relationship in Java

Is-A relationship
Inheritance

- using "extends" keyword
- tightly coupled classes relationship
  (change in one class will affect the entire class hierarchy/subclasses)
- blood relationship
- Types :- Single level, multi level, and heirarchial
- bottom up relationship

has-A relationship
Association

- using object as data member
- loosely couped classes relationship
  (change in one class will not directly affect the other class)
  - access via object only.
- non blood relationship
- Types: Aggregation (weak) & Composition (strong)
- Top Down relationship

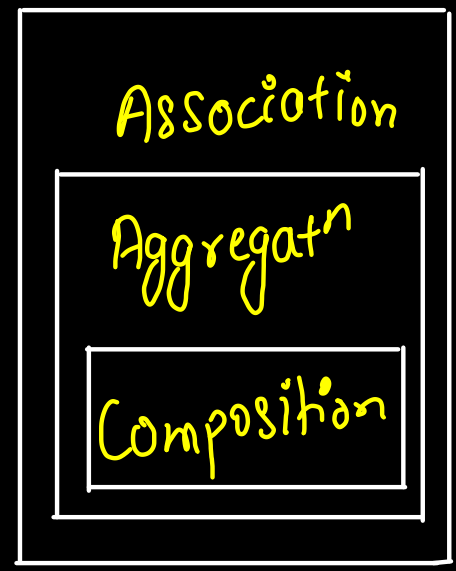
```
class Car {
    Engine obj = new Engine();
    obj.bluetooth()
}
```

Container class

```
class Engine {
}
```

Contained object

Types of Association (has A relationship)



Aggregation (weak)

without container class, if contained object may exist.

eg

Car
|
has A
◇
Music Player

College
|
has A
◇
Professor/
student

Composition (Strong)

without container object, if contained object will also not exist.

eg

Car
|
has A
◆
Engine

College
|
has A
◆
Branches/
Departments

Multithreading Concurrency Synchronization

JVM

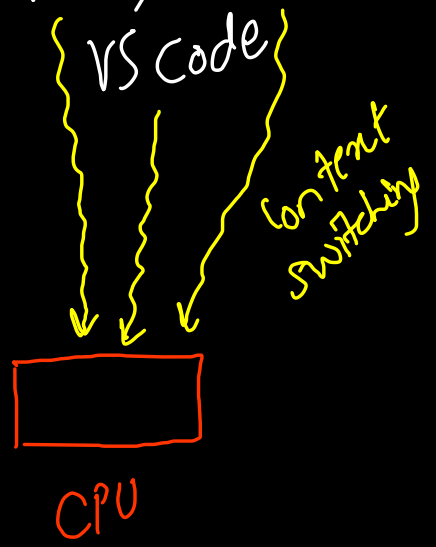
1. main() thread → user thread (application)
2. garbage collector → daemon thread (background)
3. thread scheduler → daemon thread (background)

Single threaded

Multitasking or Multiprogramming

Programs/Processes (independent)

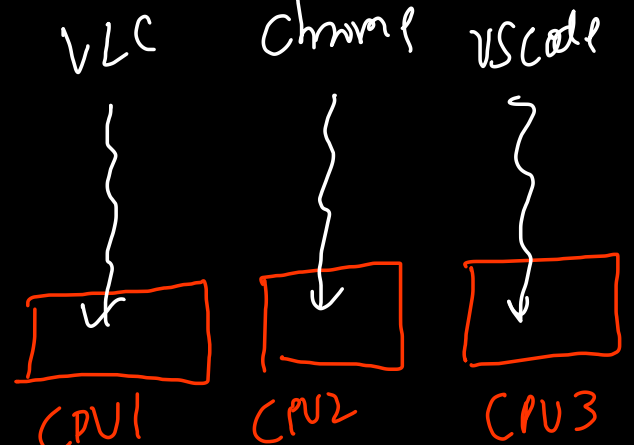
eg VLC, chrome, VS code



vs

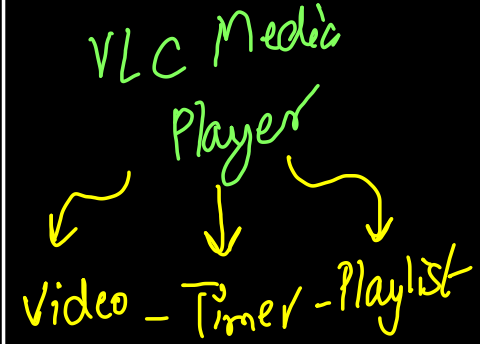
Multiprocessing

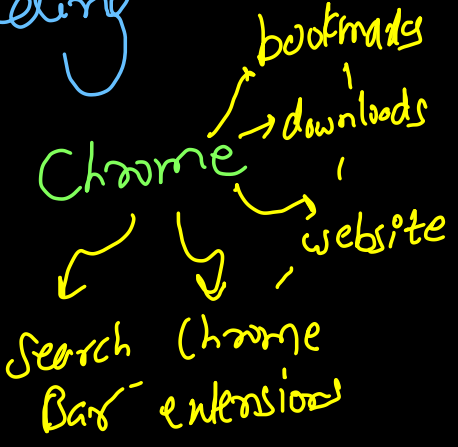
Parallel execution or Concurrent execution

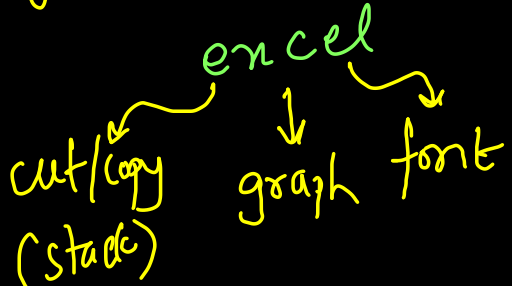


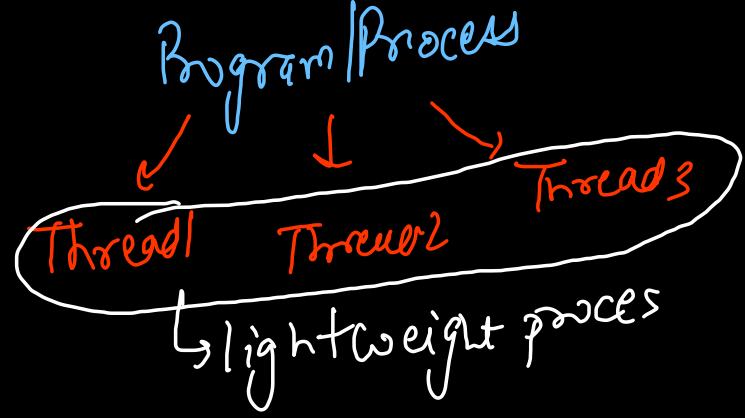
vs

Multithreading









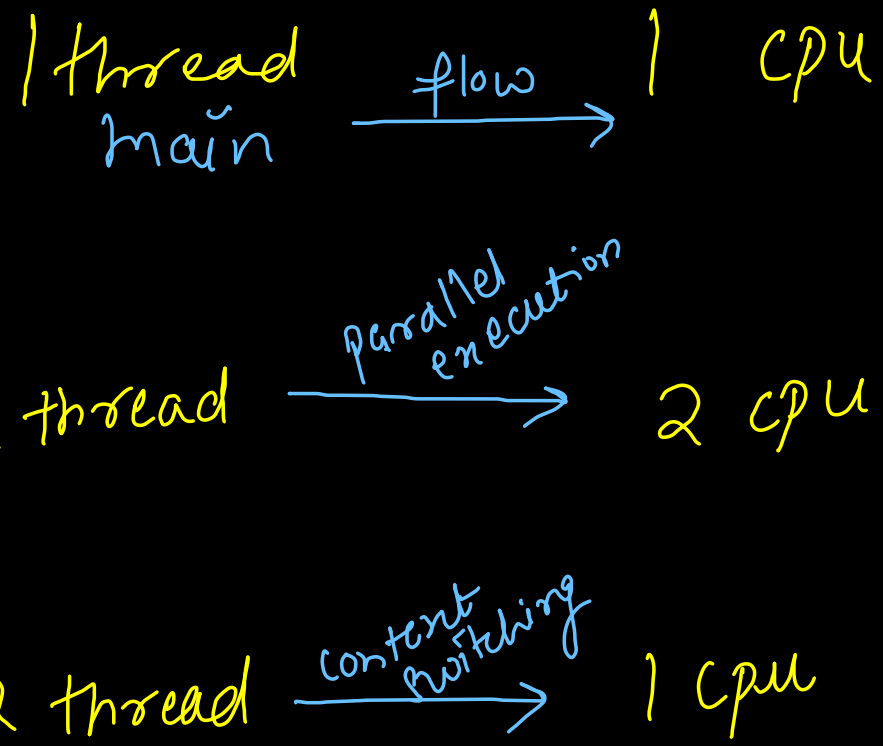
Process vs Thread

	Process	Thread
1	Heavyweight task	Lightweight task
2	Context switching (Slow)	Context switching (fast)
3	interprocess communicatn (slow)	interthread communicatn (fast)
4	memory address space different	memory address space shared
5	Processes are independent of each other	
6		Threads can be dependent on each other

Multithreading in Java

```
Interface Runnable {
    abstract void run();
}
```

```
class Thread implements Runnable {
    @Override
    void run() { }
    ------
    many methods
}
```


1 thread
main
flow
1 cpu
2 thread
parallel execution
2 cpu
2 thread
context switching
1 cpu

```java
class Stream implements Runnable {
    @Override
    public void run() {
        System.out.println("Stream Thread Started");

        for (int idx = 0; idx < 10; idx++) {
            System.out.print(idx + " ");
        }

        System.out.println("\nStream Thread Ended");
    }

    public void fun() {
        System.out.println("It is a normal function");
        // System.out.println(10 / 0);
        // It is within main thread
    }
}
```

↳ extends VLC

Thread Task

```java
class Driver {
    Run | Debug
    public static void main(String[] args) {
        Stream obj = new Stream();
        obj.run();

        Thread thread = new Thread(obj);
        thread.start();
    }
}
```

① Thread(Runnable)

↳ composition

```
It is a normal function
Stream Thread Started
0 1 2 3 4 5 6 7 8 9
Stream Thread Ended
```

Way ①: Implementing Runnable Interface

```java
class Timer extends Thread {
    @Override
    public void run() {
        System.out.println(x: "Timer Thread Started");

        for (int idx = 0; idx < 10; idx++) {
            System.out.print(idx + " ");
        }

        System.out.println(x: "\nTimer Thread Ended");
    }
}
```

② super();

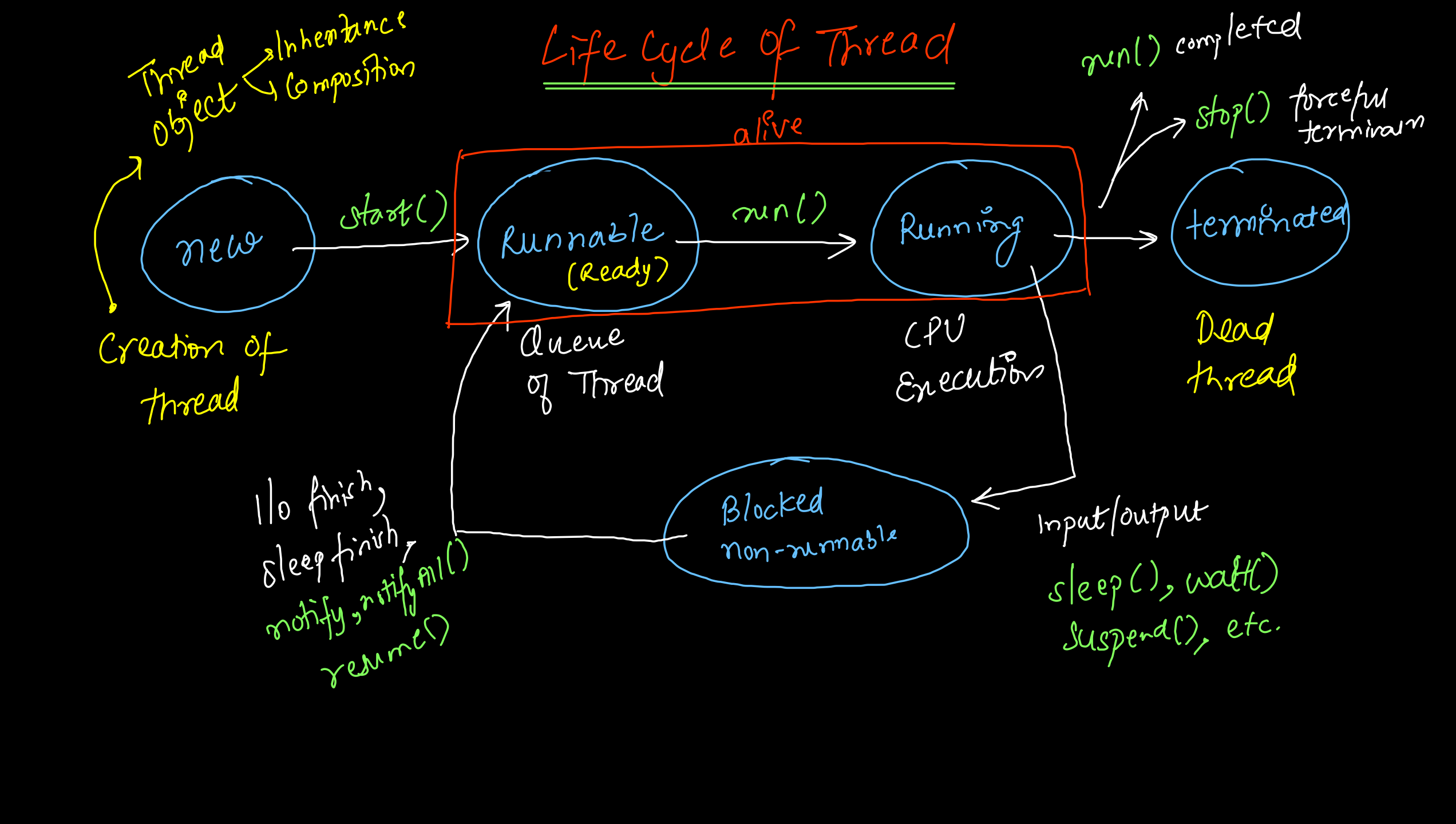
```
Timer Thread Started
0 1 2 3 4 5 6 7 8 9
Timer Thread Ended
```

```java
public static void main(String[] args) {
    // Stream obj = new Stream();
    // // obj.fun(); // main thread

    // Thread thread = new Thread(obj);
    // thread.start();

    Timer obj2 = new Timer();
    obj2.start();
}
```

We do not prefer
Using Thread class
to make thread
↳ we can only extend
one class
because Java doesn't
support
multiple inheritance

Way ② :→ using Thread class

Life Cycle of Thread



Thread Class

```
public class Thread implements Runnable
{
    public void run() { - }
    public synchronized void start() { - }
    public static native Thread currentThread();
    public final native boolean isAlive();
```

thread task

Runnable state

→ Basic Methods

true : Runnable, Running

false : new, Blocked, Terminated

```
    public final String getName() { - }
    public final synchronized void setName(String name) { - }
```

→ Naming Methods

```
    public final boolean isDaemon() { - }
    public final void setDaemon(boolean on) { - }
```

→ Daemon Thread Methods

```
    public final int getPriority() { - }
    public final void setPriority(int newPriority) { - }
```

→ Priority Based Methods

```
public class Thread implements Runnable
{
      ----- (many more methods)

      public static native void sleep(long millis) throws InterruptedException;
      public static native void yield();
      public final void join() throws InterruptedException { - }
      public final void suspend() { - }
      public final void resume() { - }
      public final void stop() { - }
      public void destroy() { - }

      public void interrupt() { - }
      public boolean isInterrupted() { - }
      public static boolean interrupted() { - }
}
```

blocked

terminate

Depricated methods

Prevent Thread Execution Methods

Interrupting a thread Methods

```
public class Object
{
    public final void wait() throws InterruptedException { - }
    public final native void notify();
    public final native void notifyAll();
}
```

Inter-Thread Communication Methods

(Synchronization)

↓

locks

Multithreading

① single task ⟶ single thread

② multiple task ⟶ single thread
- (one after other) synchronous
  - 1 → 2 → 3 ✓
- asynchronous ∝
  - 1 →
  - 2 → } ∝
  - 3 →
  - (parallely)
- ↳ thread.start();
  - ↳ once

③ single task ⟶ multiple thread
- (inter-thread communication synchronization needed)

④ multiple tasks ⟶ multiple thread

```java
Stream obj = new Stream();
// obj.fun(); // main thread

Thread thread = new Thread(obj);
System.out.println(thread.isAlive()); // New -> Not Alive
thread.start();

try {
    Thread.sleep(millis: 1000);
} catch (Exception e) {

}

System.out.println(thread.isAlive());

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

run → different

run : different

multiple threads
multiple tasks

```java
Timer obj2 = new Timer();
obj2.start();

try {
Thread.sleep(millis: 1000);
} catch (Exception e) {

}
```

run : same

multiple threads single task

It might lead to data inconsistent (race condition)

```java
// obj2.start(); // Not Possible

// Possible
Timer obj3 = new Timer();
obj3.start();
```

run : same

```
Stream obj = new Stream();
Thread thread = new Thread(obj);
obj.run(); // Main Thread → main
thread.run(); // Empty Body → main

Timer obj2 = new Timer();
obj2.run(); // Main Thread → main

thread.start(); → Thread-0
obj2.start(); → Thread-1
```

run function calling does not create a thread

Start fun calling Creates a new Thread

Daemon Thread

→ background thread

life ↴

will run till the point the user thread runs.

↳ main thread, etc.

provide services to user thread

Properties of thread will be inherited from parent's thread

{ daemon, priority }

Eg

- Garbage collector } JVM
- Thread scheduler
- Finalizer
- Attach listeners
- Signal dispatcher

```java
class Timer extends Thread {
    @Override
    public void run() {
        // System.out.println("Timer Thread Started");

        // for (int idx = 0; idx < 10; idx++) {
        // System.out.print(idx + " ");
        // }

        System.out.println(Thread.currentThread().isDaemon());
        System.out.println(Thread.currentThread().getName());

        // System.out.println("\nTimer Thread Ended");
    }
}
```

Follow up

Q) Can I set main thread as daemon?

↳ No, it is started already by JVM, hence it cannot be set daemon

```java
Timer obj2 = new Timer();
// obj2.run(); // Main Thread

obj2.setDaemon(on: true);
obj2.start();
```

You can only make a thread as daemon before it starts (while it is in new state)

obj2.setDaemon(true); → IllegalThreadStateException { After started) thread cannot be set daemon }

Thread Priorities

integer

1 2 3 4 5 6 7 8 9 10

1 → MIN_PRIORITY → lesser CPU

5 → NORM_PRIORITY → main thread → By default

10 → MAX-PRIORITY → more CPU time

```java
class HigherPriority extends Thread {
    @Override
    public void run() {
        for (int idx = 0; idx < 20; idx++) {
            System.out.println("Higher Priority " + idx);

            // try {
            // Thread.sleep(1000);
            // } catch (Exception e) {
            // }
        }
    }
}

class LowerPriority extends Thread {
    @Override
    public void run() {
        for (int idx = 0; idx < 20; idx++) {
            System.out.println("Lower Priority " + idx);

            // try {
            // Thread.sleep(1000);
            // } catch (Exception e) {
            // }
        }
    }
}
```

```java
class Driver {
    Run | Debug
    public static void main(String[] args) {
        HigherPriority obj1 = new HigherPriority();
        LowerPriority obj2 = new LowerPriority();

        obj1.start();
        obj2.start();
    }
}
```

```
Higher Priority 0
Higher Priority 1
Higher Priority 2
Higher Priority 3
Higher Priority 4
Higher Priority 5
Higher Priority 6
Higher Priority 7
Higher Priority 8
Lower Priority 0
Lower Priority 1
Lower Priority 2
Lower Priority 3
Lower Priority 4
Lower Priority 5
Higher Priority 9
Higher Priority 10
Higher Priority 11
Higher Priority 12
Higher Priority 13
Higher Priority 14
Lower Priority 6
Lower Priority 7
Higher Priority 15
Lower Priority 8
Lower Priority 9
Lower Priority 10
Higher Priority 16
Lower Priority 11
Higher Priority 17
Lower Priority 12
Higher Priority 18
Lower Priority 13
Higher Priority 19
```

```java
class HigherPriority extends Thread {
    @Override
    public void run() {
        for (int idx = 0; idx < 30; idx++) {
            System.out.println("Higher Priority " + idx);

            try {
                Thread.sleep(millis: 1000);
            } catch (Exception e) {
            }
        }
    }
}

class LowerPriority extends Thread {
    @Override
    public void run() {
        for (int idx = 0; idx < 30; idx++) {
            System.out.println("Lower Priority " + idx);

            try {
                Thread.sleep(millis: 1000);
            } catch (Exception e) {
            }
        }
    }
}
```

```
Higher Priority 0
Lower Priority 0
Higher Priority 1
Lower Priority 1
Higher Priority 2
Lower Priority 2
Lower Priority 3
Higher Priority 3
Lower Priority 4
Higher Priority 4
Lower Priority 5
Higher Priority 5
Lower Priority 6
Higher Priority 6
Lower Priority 7
Higher Priority 7
Lower Priority 8
Higher Priority 8
Lower Priority 9
Higher Priority 9
Lower Priority 10
Higher Priority 10
Lower Priority 11
Higher Priority 11
Lower Priority 12
Higher Priority 12
Higher Priority 13
Lower Priority 13
Lower Priority 14
Higher Priority 14
Higher Priority 15
Lower Priority 15
Higher Priority 16
Lower Priority 16
Higher Priority 17
Lower Priority 17
Higher Priority 18
Lower Priority 18
Lower Priority 19
```

```java
public static void main(String[] args) {
    HigherPriority obj1 = new HigherPriority();
    LowerPriority obj2 = new LowerPriority();
    obj1.setPriority(newPriority: 10);
    obj2.setPriority(newPriority: 1);

    obj1.start();
    obj2.start();
}
```

→ static method → currentThread

(Interrupted)

Thread.sleep(int milliseconds) → throws Checked Exception

0 - 999999 ms

Running state ⟶ Blocked state

↳ It holds the execution of current Running Thread.

↳ JVM → locks
mutex-semaphore } will not be taken out

Thread.yield()

- Stop the current thread to make the other threads execute
- greater or equal priority other threads will execute
- execution will resume on JVM's thread scheduler!
- It is a request to JVM to stop/hold current thread and not forceful preemption

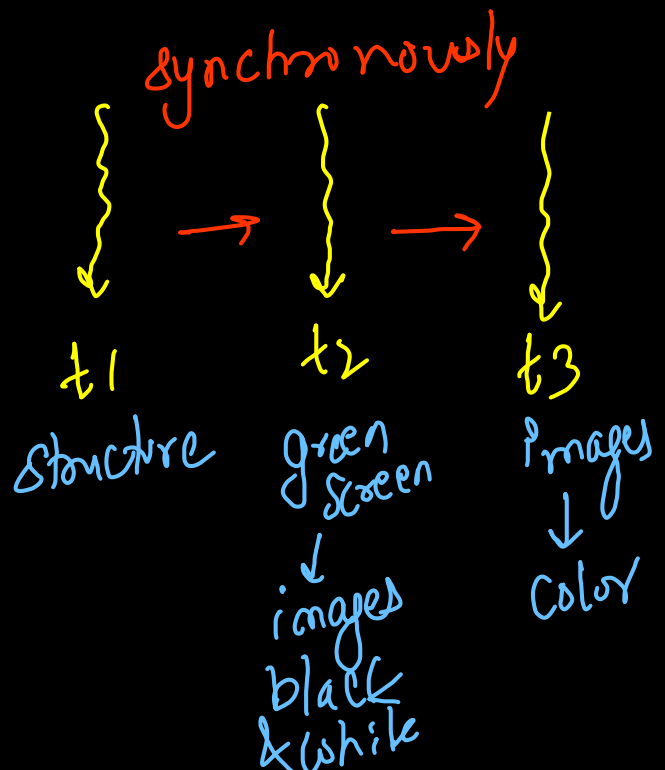
Eg Preemption of Thread!

```java
class HigherPriority extends Thread {
    @Override
    public void run() {
        for (int idx = 0; idx < 10; idx++) {
            System.out.println("Higher Priority " + idx);

            // try {
            // Thread.sleep(1000);
            // } catch (Exception e) {
            // }
        }
    }
}

class LowerPriority extends Thread {
    @Override
    public void run() {
        Thread.yield();

        for (int idx = 0; idx < 10; idx++) {
            System.out.println("Lower Priority " + idx);

            // try {
            // Thread.sleep(1000);
            // } catch (Exception e) {
            // }
        }
    }
}
```

```java
class Driver {
    Run | Debug
    public static void main(String[] args) {
        HigherPriority obj1 = new HigherPriority();
        LowerPriority obj2 = new LowerPriority();
        obj1.setPriority(newPriority: 10);
        obj2.setPriority(newPriority: 1);

        obj1.start();
        // try {
        // obj1.join();
        obj2.start();
        // } catch (Exception e) {

        // }

    }
}
```

```
Higher Priority 0
Higher Priority 1
Higher Priority 2
Higher Priority 3
Higher Priority 4
Higher Priority 5
Higher Priority 6
Higher Priority 7
Higher Priority 8
Higher Priority 9
Lower Priority 0
Lower Priority 1
Lower Priority 2
Lower Priority 3
Lower Priority 4
Lower Priority 5
Lower Priority 6
Lower Priority 7
Lower Priority 8
Lower Priority 9
```

Thread Name, join()



Chain of Responsibility / Decorator Design Pattern

T2 will stop it's execution till t1 ends its execution!

```
t2.run(){
    t1.join();
}

t3.run(){
    t2.join();
}
```

```java
class HigherPriority extends Thread {
    @Override
    public void run() {
        for (int idx = 0; idx < 10; idx++) {
            System.out.println("Higher Priority " + idx);

            // try {
            // Thread.sleep(1000);
            // } catch (Exception e) {
            // }
        }
    }
}

class LowerPriority extends Thread {
    @Override
    public void run() {
        Thread.yield();

        for (int idx = 0; idx < 10; idx++) {
            System.out.println("Lower Priority " + idx);

            // try {
            // Thread.sleep(1000);
            // } catch (Exception e) {
            // }
        }
    }
}
```

```java
class Driver {
    Run | Debug
    public static void main(String[] args) {
        HigherPriority obj1 = new HigherPriority();
        LowerPriority obj2 = new LowerPriority();
        obj1.setPriority(newPriority: 10);
        obj2.setPriority(newPriority: 1);

        obj1.start();
        try {
            obj1.join();
            obj2.start();
        } catch (Exception e) {

        }

    }
}
```

```
Higher Priority 0
Higher Priority 1
Higher Priority 2
Higher Priority 3
Higher Priority 4
Higher Priority 5
Higher Priority 6
Higher Priority 7
Higher Priority 8
Higher Priority 9
Lower Priority 0
Lower Priority 1
Lower Priority 2
Lower Priority 3
Lower Priority 4
Lower Priority 5
Lower Priority 6
Lower Priority 7
Lower Priority 8
Lower Priority 9
```

Sleep

1. definite time blocked
2. Static
3. eg Timer (Bulbs, Scorecard), Counting (clock)

Yield

Higher Priority Thread Preemption (concept switching)

Static

eg Forceful Preemption

Join

waiting for other thread to finish execution (synchronous)

non-static

eg Synchronous Execution

sleep() method Prototype :-

1. public static native void sleep(long ms) throws InterruptedException;
2. public static void sleep(long ms, int ns) throws InterruptedException;

yield() method Prototype :-

1. public static native void yield();

join() method Prototype :-

1. public final void join() throws InterruptedException;
2. public final synchronized void join(long ms) throws InterruptedException;
3. public final synchronized void join(long ms, int ns) throws InterruptedException;

interrupt() method → throws checked Exception

→ If any thread is in blocked state
{ sleep(), wait(), yield(), join() }

And if you want to interrupt the current running thread!

```java
class LowerPriority extends Thread {
    @Override
    public void run() {
        // Thread.yield();

        for (int idx = 0; idx < 10; idx++) {
            System.out.println("Lower Priority " + idx

            try {
                Thread.sleep(millis: 1000);

            } catch (Exception e) {
            }
        }
    }
}

class Driver {
    Run | Debug
    public static void main(String[] args) {
        HigherPriority obj1 = new HigherPriority();
        LowerPriority obj2 = new LowerPriority();
        obj1.setPriority(newPriority: 10);
        obj2.setPriority(newPriority: 1);

        try {
            // obj1.join();
            obj2.start();

            obj2.interrupt();
            obj1.start();
        } catch (Exception e) {

        }

    }
}
```

Synchronization. (Race Condition → Data Inconsistent)

```java
class Theater {
    int availableSeats;

    Theater(int availableSeats) {
        this.availableSeats = availableSeats;
    }

    public void bookSeats(User user, int seats) {
        if (availableSeats >= seats) {
            System.out.println("Username " + user.name + " "
                    + " has booked " + seats + " seats ");

            availableSeats -= seats;

            System.out.println("Remaining Seats : " + availableSeats);
        } else {
            System.out.println(x: "There are no seats available");
        }
    }
}
```

→ shared resource

// critical section

u3 u2 u1 u4

```
Username Sita  has booked 10 seats
Username Gita  has booked 10 seats
Username Ram  has booked 10 seats
Username Shyam  has booked 10 seats
Remaining Seats : 10
Remaining Seats : 0
Remaining Seats : -20
Remaining Seats : -10
```

```java
class User extends Thread {
    String name;
    static Theater pvr = new Theater(availableSeats: 20);

    User(String name) {
        this.name = name;
    }

    @Override
    public void run() {
        pvr.bookSeats(this, seats: 10);
    }
}
```

```java
public static void main(String[] args) {
    User u1 = new User(name: "Ram");
    User u2 = new User(name: "Shyam");
    User u3 = new User(name: "Sita");
    User u4 = new User(name: "Gita");

    u1.start();
    u2.start();
    u3.start();
    u4.start();
}
```

parallely
↳ multiple threads
↳ same resource

Synchronization (multithreading)

Mutual Exclusion
{multiple thread single task}

objects lock
1. Synchronized method
2. Synchronized block
3. Static synchronized block

Thread → class lock

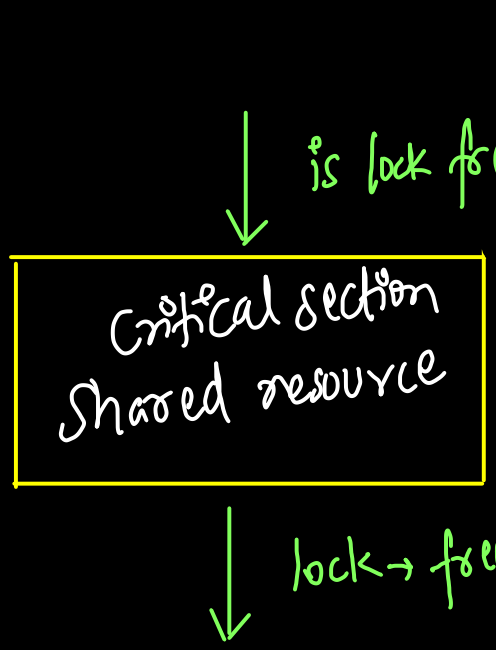
Synchronized keyword

Inter Thread Communication (ITC)
{multiple thread multiple task}

mutex/semaphores/lock

1. wait()
2. notify()
3. notifyAll()

Object class



Progress
↳ independent execution order

Bounded wait
↳ no indefinite time of waiting
↳ starvation!

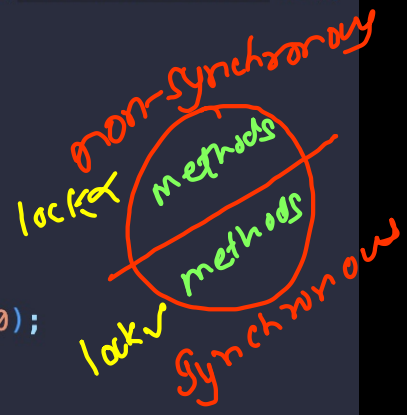
```java
class Theater {
    int availableSeats;

    Theater(int availableSeats) {
        this.availableSeats = availableSeats;
    }

    synchronized public void bookSeats(User user, int seats) {
        if (availableSeats >= seats) {
            System.out.println("Username " + user.name + " "
                    + " has booked " + seats + " seats ");

            availableSeats -= seats;

            System.out.println("Remaining Seats : " + availableSeats);
        } else {
            System.out.println(x: "There are no seats available");
        }
    }
}
```

lock acquire

lock free

```
Username Ram  has booked 10 seats
Remaining Seats : 10
Username Gita  has booked 10 seats
Remaining Seats : 0
There are no seats available
There are no seats available
```

```java
class User extends Thread {
    String name;
    static Theater pvr = new Theater(availableSeats: 20);

    User(String name) {
        this.name = name;
    }

    @Override
    public void run() {
        pvr.bookSeats(this, seats: 10);
    }
}
```



Thread

Object

You, 2 days ago | 1 author (You)

```java
class Driver {
    Run | Debug
    public static void main(String[] args) {
        User u1 = new User(name: "Ram");
        User u2 = new User(name: "Shyam");
        User u3 = new User(name: "Sita");
        User u4 = new User(name: "Gita");

        u1.start();
        u2.start();
        u3.start();
        u4.start();
    }
}
```

① Synchronized method

```java
synchronized public void bookSeats(User user, int seats) {
    System.out.println("User " + user.name + " : Please Proceed to
    Payment Gateway");

    if (availableSeats >= seats) {
        System.out.println("Username " + user.name + " "
                + " has booked " + seats + " seats ");

        availableSeats -= seats;

        System.out.println("Remaining Seats : " + availableSeats);
    } else {
        System.out.println(x: "There are no seats available");
    }

    System.out.println(x: "Do you want to go back to the main screen ?
    ");
}
```

```
User Ram : Please Proceed to Payment Gateway
Username Ram  has booked 10 seats
Remaining Seats : 10
Do you want to go back to the main screen ?
User Gita : Please Proceed to Payment Gateway
Username Gita  has booked 10 seats
Remaining Seats : 0
Do you want to go back to the main screen ?
User Sita : Please Proceed to Payment Gateway
There are no seats available
Do you want to go back to the main screen ?
User Shyam : Please Proceed to Payment Gateway
There are no seats available
Do you want to go back to the main screen ?
```

① This is not effective way of achieving synchronization

↳ unnecessarily increasing the critical section

↳ parallel execution of thread is reduced

Performance is impacted!

```java
public void bookSeats(User user, int seats) {
    System.out.println("User " + user.name + " : Please Proceed to Payment Gateway");

    synchronized(this){
        if (availableSeats >= seats) {
            System.out.println("Username " + user.name + " "
                    + " has booked " + seats + " seats ");

            availableSeats -= seats;

            System.out.println("Remaining Seats : " + availableSeats);
        } else {
            System.out.println(x: "There are no seats available");
        }
    }

    System.out.println(x: "Do you want to go back to the main screen ?");
}
```

② Synchronized Block

```
User Shyam : Please Proceed to Payment Gateway
User Gita : Please Proceed to Payment Gateway
User Sita : Please Proceed to Payment Gateway
User Ram : Please Proceed to Payment Gateway
Username Shyam  has booked 10 seats
Remaining Seats : 10
Do you want to go back to the main screen ?
Username Ram  has booked 10 seats
Remaining Seats : 0
Do you want to go back to the main screen ?
There are no seats available
Do you want to go back to the main screen ?
There are no seats available
Do you want to go back to the main screen ?
```

```java
class Theater {
    static int availableSeats = 20;

    public static void bookSeats(User user, int seats) {
        System.out.println("User " + user.name + " : Please Proceed to Payment Gateway");

        if (availableSeats >= seats) {
            System.out.println("Username " + user.name + " "
                    + " has booked " + seats + " seats ");

            availableSeats -= seats;

            System.out.println("Remaining Seats : " + availableSeats);
        } else {
            System.out.println("There are no seats available");
        }

        System.out.println("Do you want to go back to the main screen ?");
    }
}
```

synchronized

```java
class User extends Thread {
    String name;
    Theater theater;

    User(String name) {
        this.name = name;
    }

    User(String name, Theater theater) {
        this.theater = theater;
    }

    @Override
    public void run() {
        theater.bookSeats(this, seats: 10);
    }
}
```

```java
class Driver {
    Run | Debug
    public static void main(String[] args) {
        Theater delhi = new Theater();
        Theater mumbai = new Theater();

        User u1 = new User(name: "Ram", delhi);
        User u2 = new User(name: "Shyam", delhi);
        User u3 = new User(name: "Sita", mumbai);
        User u4 = new User(name: "Gita", mumbai);

        u1.start();
        u2.start();
        u3.start();
        u4.start();
    }
}
```

Synchronization

↳ { class level }

Synchronized

```
User Gita : Please Proceed to Payment Gateway
User Shyam : Please Proceed to Payment Gateway
User Sita : Please Proceed to Payment Gateway
User Ram : Please Proceed to Payment Gateway
Username Gita  has booked 10 seats
Username Sita  has booked 10 seats
Username Ram  has booked 10 seats
Username Shyam  has booked 10 seats
Remaining Seats : -20
Do you want to go back to the main screen ?
Remaining Seats : -10
Do you want to go back to the main screen ?
Remaining Seats : 0
Do you want to go back to the main screen ?
Remaining Seats : 10
Do you want to go back to the main screen ?
```

```java
class Theater {
    static int availableSeats = 20;

    public synchronized static void bookSeats(User user, int
    seats) {
        System.out.println("User " + user.name + " : Please
        Proceed to Payment Gateway");

        if (availableSeats >= seats) {
            System.out.println("Username " + user.name + " "
                    + " has booked " + seats + " seats ");

            availableSeats -= seats;

            System.out.println("Remaining Seats : " +
            availableSeats);
        } else {
            System.out.println(x: "There are no seats available");
        }

        System.out.println(x: "Do you want to go back to the main
        screen ?");
    }
}
```

```
User Ram : Please Proceed to Payment Gateway
Username Ram  has booked 10 seats
Remaining Seats : 10
Do you want to go back to the main screen ?
User Gita : Please Proceed to Payment Gateway
Username Gita  has booked 10 seats
Remaining Seats : 0
Do you want to go back to the main screen ?
User Sita : Please Proceed to Payment Gateway
There are no seats available
Do you want to go back to the main screen ?
User Shyam : Please Proceed to Payment Gateway
There are no seats available
Do you want to go back to the main screen ?
```

```java
class Theater {
    int availableSeats;

    Theater(int availableSeats) {
        this.availableSeats = availableSeats;
    }

    public void bookSeats(User user, int seats) {
        System.out.println("User " + user.name + " : Please Proceed to Payment
        Gateway");

        synchronized (this) {
            if (availableSeats >= seats) {
                System.out.println("Username " + user.name + " "
                        + " has booked " + seats + " seats ");

                availableSeats -= seats;
            } else {
                System.out.println(x: "There are no seats available");
            }
        }

        System.out.println(x: "Do you want to go back to the main screen ?");
    }
}
```


```
Remaining Seats : 20
Remaining Seats : 20
Remaining Seats : 20
Remaining Seats : 20
User Ram : Please Proceed to Payment Gateway
User Shyam : Please Proceed to Payment Gateway
User Gita : Please Proceed to Payment Gateway
User Sita : Please Proceed to Payment Gateway
Username Ram  has booked 10 seats
Do you want to go back to the main screen ?
Username Shyam  has booked 10 seats
Do you want to go back to the main screen ?
There are no seats available
Do you want to go back to the main screen ?
There are no seats available
```


```java
class User extends Thread {
    String name;
    static Theater pvr = new Theater(availableSeats: 20);

    User(String name) {
        this.name = name;
    }

    @Override
    public void run() {
        pvr.bookSeats(this, seats: 10);
    }
}
```

```java
class Driver {
    Run | Debug
    public static void main(String[] args) {
        User u1 = new User(name: "Ram");
        User u2 = new User(name: "Shyam");
        User u3 = new User(name: "Sita");
        User u4 = new User(name: "Gita");

        u1.start();
        System.out.println("Remaining Seats : " + User.pvr.availableSeats);
        u2.start();
        System.out.println("Remaining Seats : " + User.pvr.availableSeats);
        u3.start();
        System.out.println("Remaining Seats : " + User.pvr.availableSeats);
        u4.start();
        System.out.println("Remaining Seats : " + User.pvr.availableSeats);
```

```java
public void bookSeats(User user, int seats) {
    System.out.println("User " + user.name + " : Please Proceed to Payment
    Gateway");

    synchronized (this) {
        if (availableSeats >= seats) {
            System.out.println("Username " + user.name + " "
                    + " has booked " + seats + " seats ");

            availableSeats -= seats;
        } else {
            System.out.println(x: "There are no seats available");
        }

        this.notify();
    }

    System.out.println(x: "Do you want to go back to the main screen ?");
}
```

```
User Ram : Please Proceed to Payment Gateway
Username Ram  has booked 10 seats
Do you want to go back to the main screen ?
Remaining Seats : 10
User Shyam : Please Proceed to Payment Gateway
Username Shyam  has booked 10 seats
Do you want to go back to the main screen ?
Remaining Seats : 0
User Sita : Please Proceed to Payment Gateway
There are no seats available
Do you want to go back to the main screen ?
Remaining Seats : 0
User Gita : Please Proceed to Payment Gateway
There are no seats available
Do you want to go back to the main screen ?
Remaining Seats : 0
```

```java
public static void main(String[] args) throws Exception {
    User u1 = new User(name: "Ram");
    User u2 = new User(name: "Shyam");
    User u3 = new User(name: "Sita");
    User u4 = new User(name: "Gita");

    u1.start();

    synchronized (u1) {
        u1.wait(); → before lock acquiring
        System.out.println("Remaining Seats : " + User.pvr.availableSeats);
    }

    u2.start();
    synchronized (u2) {
        u2.wait();
        System.out.println("Remaining Seats : " + User.pvr.availableSeats);
    }

    u3.start();
    synchronized (u3) {
        u3.wait();
        System.out.println("Remaining Seats : " + User.pvr.availableSeats);
    }

    u4.start();
    synchronized (u4) {
        u4.wait();
        System.out.println("Remaining Seats : " + User.pvr.availableSeats);
    }

}
```

```java
public static void main(String[] args) throws Exception {
    User u1 = new User(name: "Ram");
    User u2 = new User(name: "Shyam");
    User u3 = new User(name: "Sita");
    User u4 = new User(name: "Gita");

    u1.start();
    u2.start();
    u3.start();
    u4.start();

    synchronized (u1) {
        u1.wait();
        System.out.println("Remaining Seats : " + User.pvr.availableSeats);
    }

    // u2.start();
    synchronized (u2) {
        u2.wait();
        System.out.println("Remaining Seats : " + User.pvr.availableSeats);
    }

    // u3.start();
    synchronized (u3) {
        u3.wait();
        System.out.println("Remaining Seats : " + User.pvr.availableSeats);
    }

    // u4.start();
    synchronized (u4) {
        u4.wait();
        System.out.println("Remaining Seats : " + User.pvr.availableSeats);
    }

}
```

(infinite/unbounded wait) (u2 is in dead state now)

```
User Shyam : Please Proceed to Payment Gateway
User Ram : Please Proceed to Payment Gateway
User Sita : Please Proceed to Payment Gateway
User Gita : Please Proceed to Payment Gateway
Username Shyam  has booked 10 seats
Do you want to go back to the main screen ?
Username Gita  has booked 10 seats
Do you want to go back to the main screen ?
There are no seats available
Do you want to go back to the main screen ?
There are no seats available
Do you want to go back to the main screen ?
Remaining Seats : 0
```

Forcefully exit: Control+C

```java
class Payment extends Thread {
    public synchronized void run() {
        System.out.println(x: "Payment Processing");
        try {
            for (int idx = 0; idx < 5; idx++) {
                Thread.sleep(millis: 300);
                System.out.println(x: ".");
            }
        } catch (Exception e) {
        }

        this.notify();
    }
}
```

```
Payment Processing
.
.
.
.
.
Ticket Generation
```

Unbounded wait → contention

```java
class Food extends Thread {
    Payment t1;

    public void run() {

        synchronized (t1) {
            try {
                t1.wait();
            } catch (Exception e) {
            }

            System.out.println(x: "Food Buy Starts");
        }
    }
}

class Ticket extends Thread {
    Payment t1;

    public synchronized void run() {
        synchronized (t1) {
            try {
                t1.wait();
            } catch (Exception e) {
            }
        }

        System.out.println(x: "Ticket Generation");
    }
}
```

Leetcode Concurrency Problems

1. Print Foo bar Alternately LC 1115
2. Print In Order LC 1114
3. Print Zero even odd LC 1116
4. Building H_2O LC 1117
5. Fizz Buzz multithreaded LC 1195
6. Dining Philosopher LC 1226

```java
class FooBar {
    private int n;

    public FooBar(int n) {
        this.n = n;
    }

    public void foo(Runnable printFoo) throws InterruptedException {

        for (int i = 0; i < n; i++) {

            printFoo.run(); // print foo
        }
    }

    public void bar(Runnable printBar) throws InterruptedException {

        for (int i = 0; i < n; i++) {

            printBar.run(); // print bar
        }
    }
}
```

Non-Synchronous

↓

Parallely Threads

↓ ↓

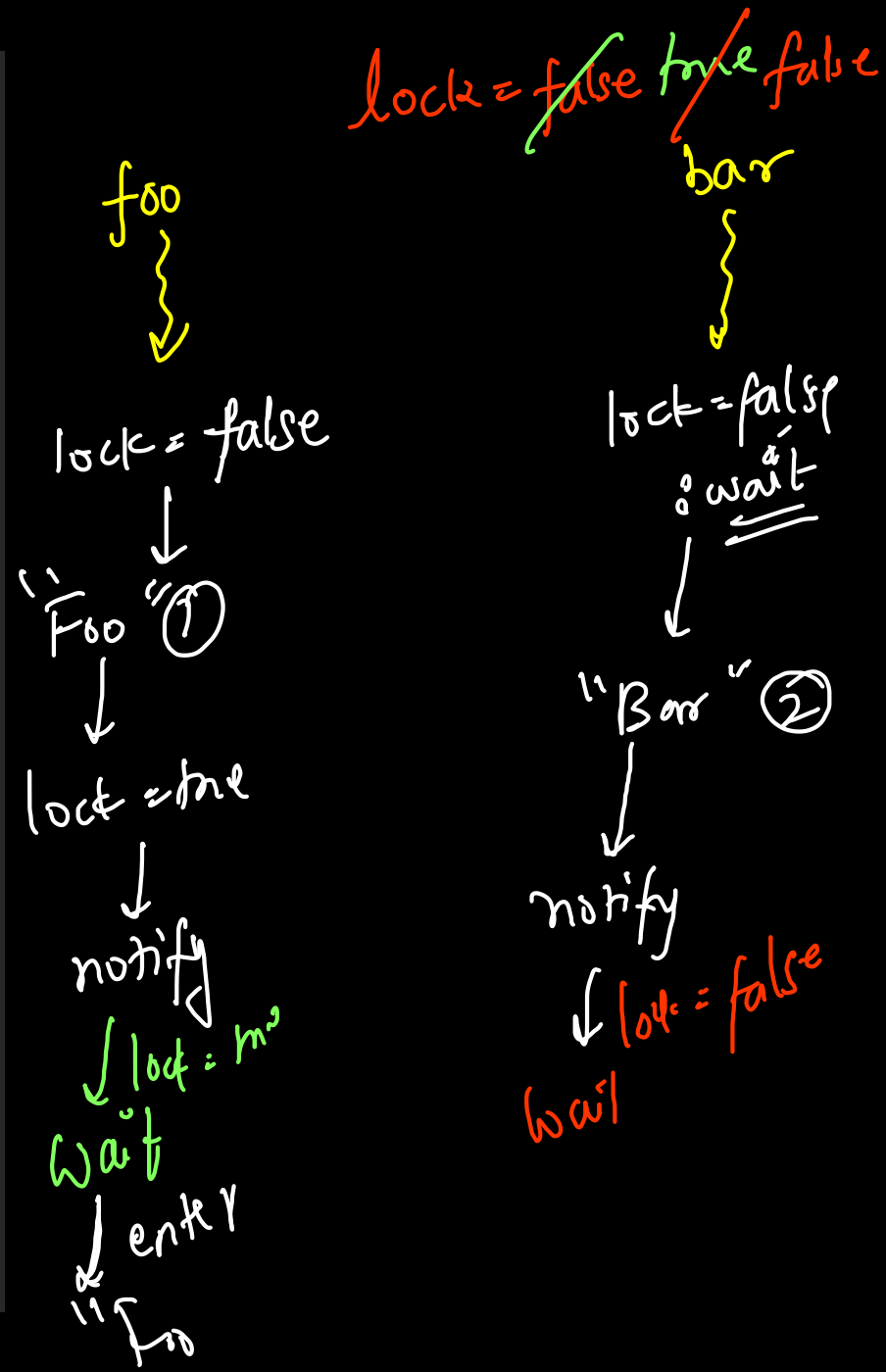
PrintFoo PrintBar

PrintFooBar Alternatively

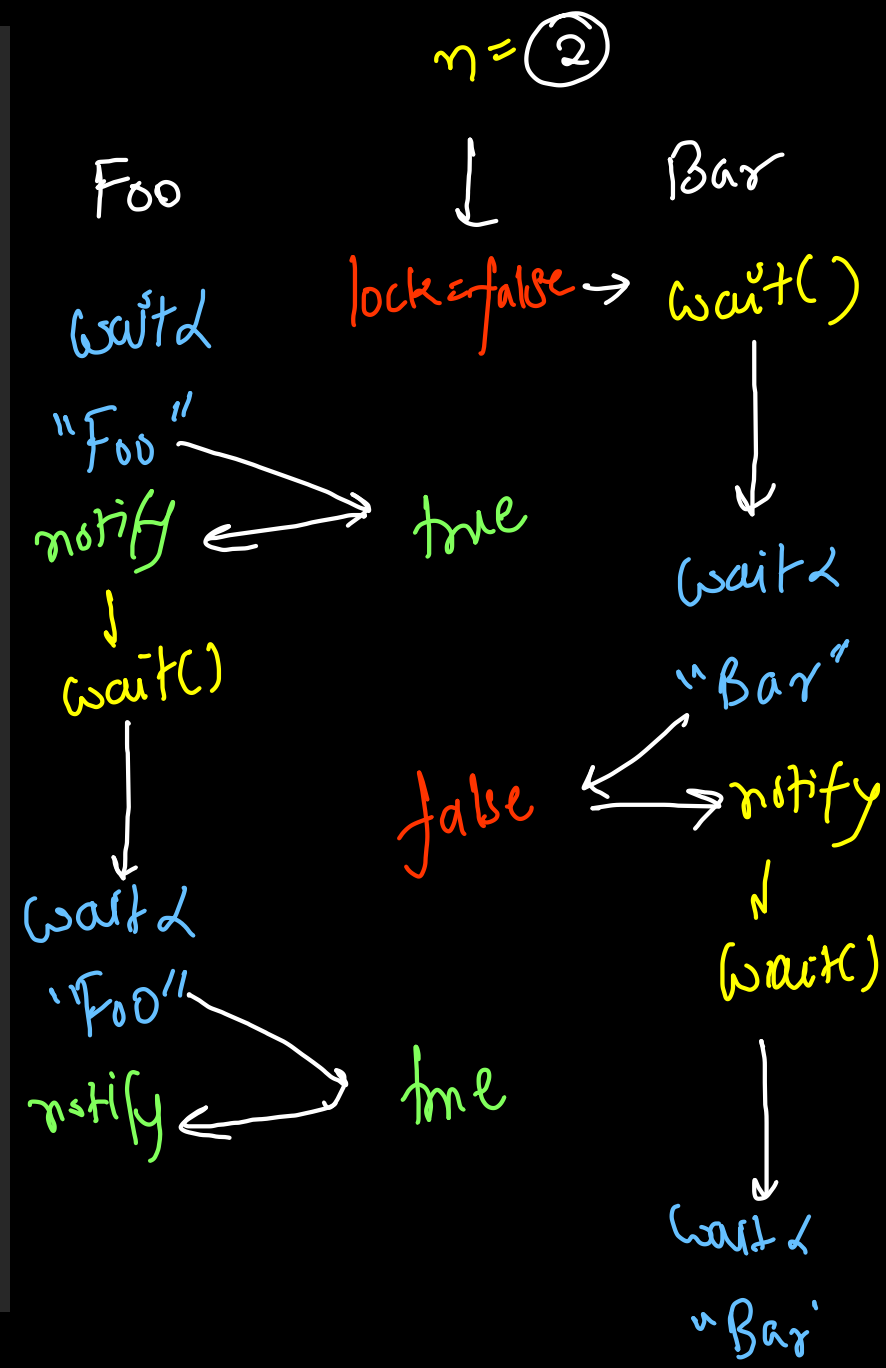
LC 1115

```java
class FooBar {
    private boolean lock = false;
    // lock = false: foo can enter AND bar will wait
    // lock = true: bar can enter AND foo will wait

    private int n;
    public FooBar(int n) {
        this.n = n;
    }

    public synchronized void foo(Runnable printFoo) throws InterruptedException {

        for (int i = 0; i < n; i++) {
            while(lock == true) this.wait();
            printFoo.run(); // print foo
            lock = true;
            this.notifyAll();
        }
    }

    public synchronized void bar(Runnable printBar) throws InterruptedException {

        for (int i = 0; i < n; i++) {
            while(lock == false) this.wait();
            printBar.run(); // print bar
            lock = false;
            this.notifyAll();
        }
    }
}
```

n=5



```java
class FooBar {
    private boolean lock = false;
    // lock = false: foo can enter AND bar will wait
    // lock = true: bar can enter AND foo will wait

    private int n;
    public FooBar(int n) {
        this.n = n;
    }

    public synchronized void foo(Runnable printFoo) throws InterruptedException {

        for (int i = 0; i < n; i++) {
            while(lock == true) this.wait();
            printFoo.run(); // print foo
            lock = true;
            this.notifyAll();
        }
    }

    public synchronized void bar(Runnable printBar) throws InterruptedException {

        for (int i = 0; i < n; i++) {
            while(lock == false) this.wait();
            printBar.run(); // print bar
            lock = false;
            this.notifyAll();
        }
    }
}
```



1114. Print in Order

```
class Foo {
    public void first(Runnable printFirst) throws InterruptedException {
        printFirst.run(); // print first
    }

    public void second(Runnable printSecond) throws InterruptedException {
        printSecond.run(); // print second
    }

    public void third(Runnable printThird) throws InterruptedException {
        printThird.run(); // print third
    }
}
```

lock = 0 → 1

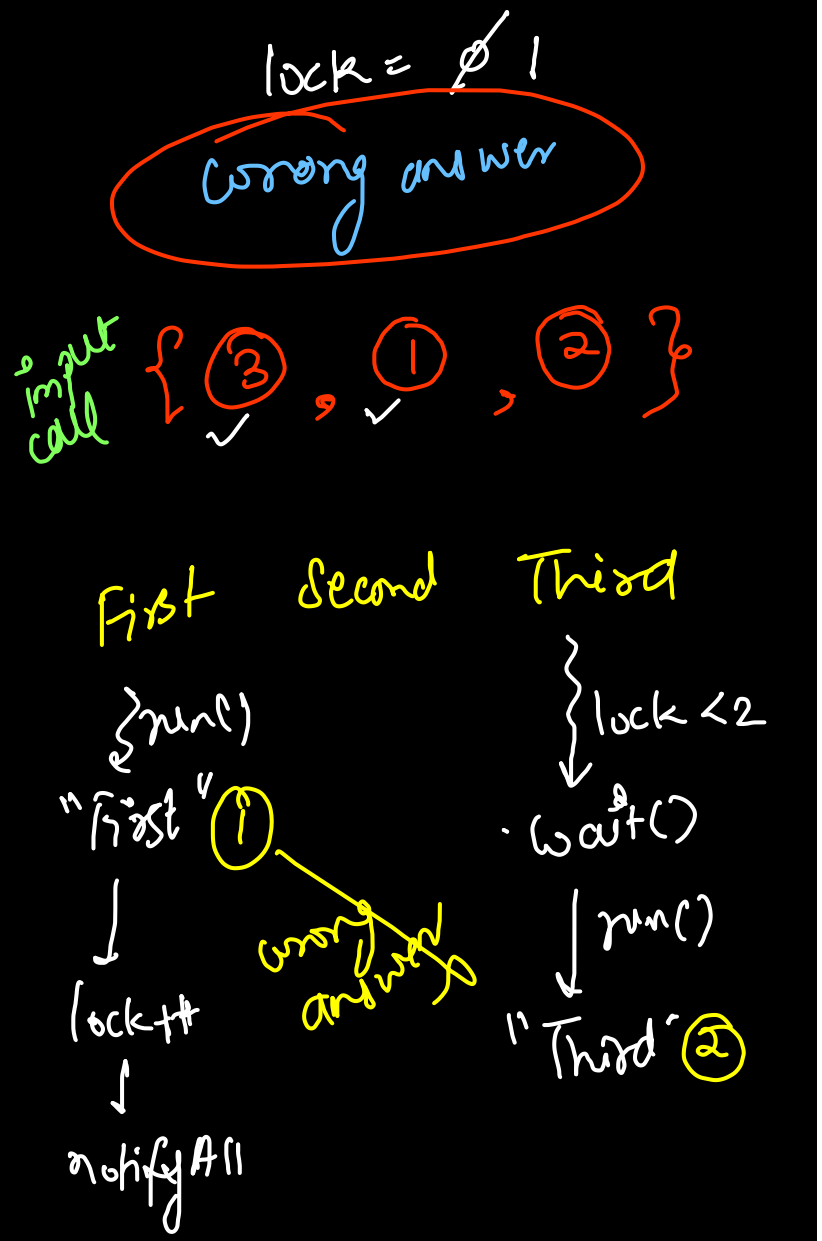
lock == 0 ⇒ wait
1 → 2

lock < 2 ⇒ wait

```
int lock = 0;

public synchronized void first(Runnable printFirst) throws InterruptedException {
    printFirst.run(); // print first
    lock++;
    this.notifyAll();
}

public synchronized void second(Runnable printSecond) throws InterruptedException {
    if(lock < 1) wait();
    printSecond.run(); // print second
    lock++;
    this.notifyAll();
}

public synchronized void third(Runnable printThird) throws InterruptedException {
    if(lock < 2) wait();
    printThird.run(); // print third
}
```

```
class Foo {
    int lock = 0;

    public synchronized void first(Runnable printFirst) throws InterruptedException {
        printFirst.run(); // print first
        lock++;
        this.notify();
    }

    public synchronized void second(Runnable printSecond) throws InterruptedException {
        if(lock == 0) wait();
        printSecond.run(); // print second
        lock++;
        this.notifyAll();
    }

    public synchronized void third(Runnable printThird) throws InterruptedException {
        if(lock == 0) wait();
        if(lock == 1) wait();
        printThird.run(); // print third
    }
}
```

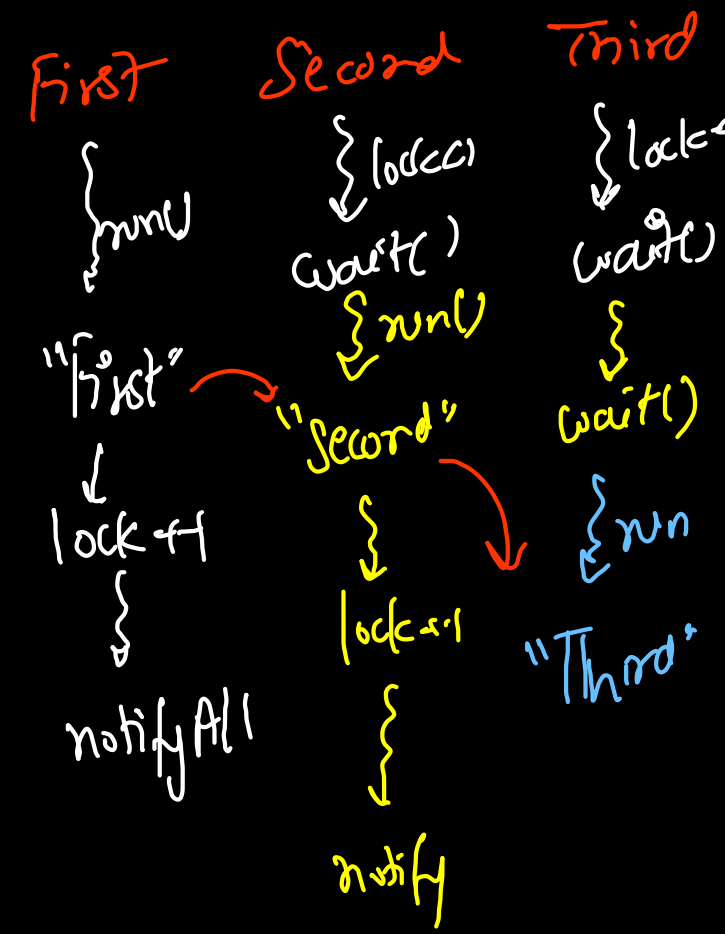
→ deadlock / unbounded wait of second



```java
class Foo {
    int lock = 0;

    public synchronized void first(Runnable printFirst) throws InterruptedException {
        printFirst.run(); // print first
        lock++;
        this.notifyAll();
    }

    public synchronized void second(Runnable printSecond) throws InterruptedException {
        while(lock < 1) wait(); → wait till first is not printed
        printSecond.run(); // print second
        lock++;
        this.notifyAll();
    }

    public synchronized void third(Runnable printThird) throws InterruptedException {
        while(lock < 2) wait(); → wait till first & second are not printed
        printThird.run(); // print third
    }
}
```

Right Answer

{ 3 2 1 }



0 ⇒ first ✓ second & third ⇒ wait
1 ⇒ second ✓ & third ⇒ wait
2 ⇒ third ✓

```
class Foo {
    int lock = 0;

    public synchronized void first(Runnable printFirst) throws InterruptedException {
        printFirst.run(); // print first
        lock++;
        this.notifyAll();
    }

    public synchronized void second(Runnable printSecond) throws InterruptedException {
        if(lock == 0) wait();
        printSecond.run(); // print second
        lock++;
        this.notifyAll();
    }

    public synchronized void third(Runnable printThird) throws InterruptedException {
        if(lock == 0) wait();
        if(lock == 1) wait();
        printThird.run(); // print third
    }
}
```

Approach No 2

n = 6

"0 1 0 2 0 3 0 4 0 5 0 6"

zero → printNumber.accept(0)

odd → printNumber.accept(oddNo)

even → printNumber.accept(evenNo)

```java
class ZeroEvenOdd {
    private int n;
    private int counter = 0;

    public ZeroEvenOdd(int n) {
        this.n = n;
    }

    // printNumber.accept(x) outputs "x", where x is an integer.
    public void zero(IntConsumer printNumber) throws InterruptedException {
        printNumber.accept(0);
    }

    public void even(IntConsumer printNumber) throws InterruptedException {
        if(counter > n) return;

        if(counter % 2 == 0)
            printNumber.accept(counter);
        counter++;
    }

    public void odd(IntConsumer printNumber) throws InterruptedException {
        if(counter > n) return;

        if(counter % 2 == 1)
            printNumber.accept(counter);
        counter++;
    }
}
```

```java
class ZeroEvenOdd {
    private int n;
    private int counter = 1;
    private boolean zeroLock = false;
    // false: zero can enter, even and odd will wait
    // true: even or odd will run, zero will wait

    public ZeroEvenOdd(int n) {
        this.n = n;
    }

    public synchronized void zero(IntConsumer printNumber)
        while(counter <= n){
            if(zeroLock == true) {
                this.wait();
                continue;
            }
            printNumber.accept(0);
            zeroLock = true;
            this.notifyAll();
        }
    }
```

→ counting semaphore (counter)

→ binary semaphore (mutex) (zeroLock)

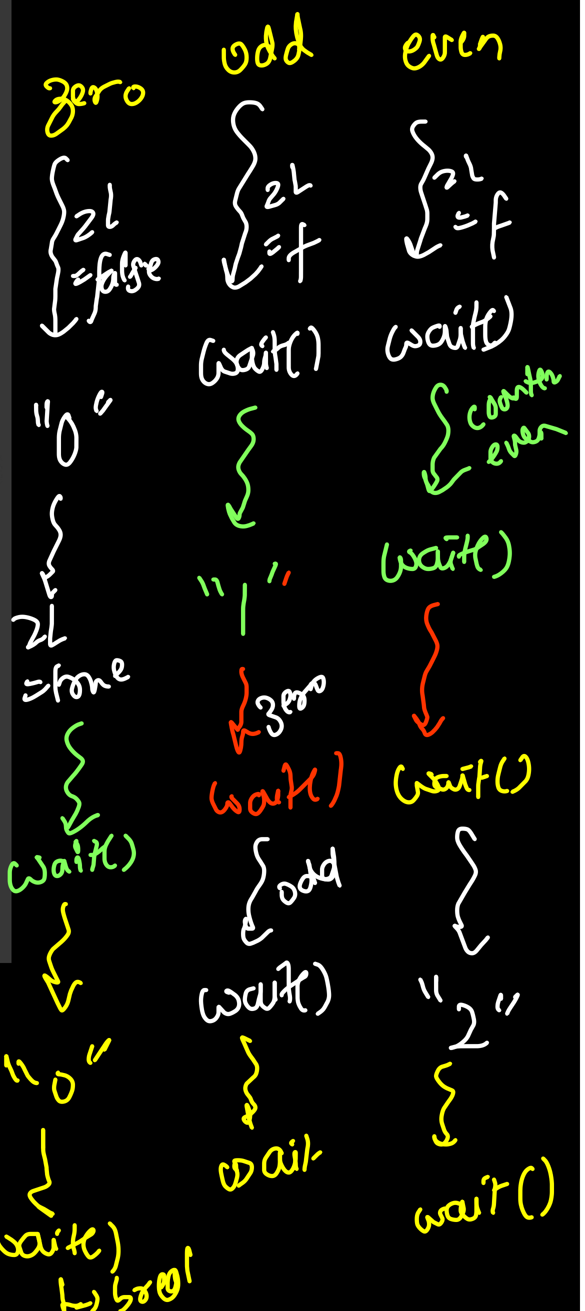
"0 1 0 2 0"

n=3

Counter = 1 2 3

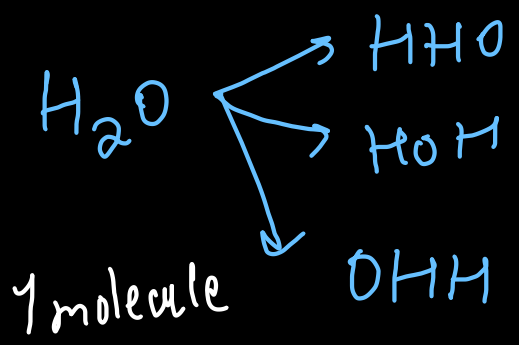
zerolock : false true false true false

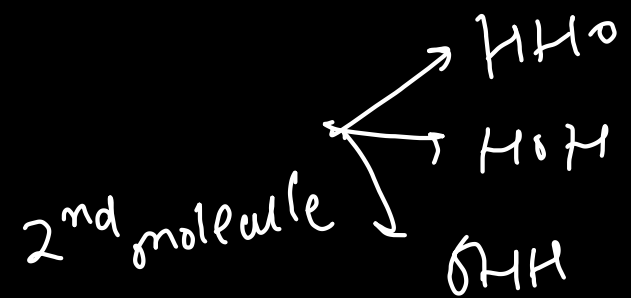
```java
public synchronized void even(IntConsumer printNumber)
    while(counter <= n){
        if(zeroLock == false || counter % 2 == 1) {
            this.wait();
            continue;
        }
        printNumber.accept(counter);
        counter++;
        zeroLock = false;
        this.notifyAll();
    }
}

public synchronized void odd(IntConsumer printNumber)
    while(counter <= n){
        if(zeroLock == false || counter % 2 == 0) {
            this.wait();
            continue;
        }
        printNumber.accept(counter);
        counter++;
        zeroLock = false;
        this.notifyAll();
    }
}
```



1117 Leetcode

Building H_2O





HHO HHO
HHO HOH
HHO OHH
HOH HHO
HOH HOH
HOH OHH
OHH HHO
OHH HOH
OHH OHH
Valid orders

Explanation: "HOHHHO", "OHHHHO", "HHOHOH", "HOHHOH", "OHHHOH", "HHOOHH", "HOHOHH" and "OHHOHH" are also valid answers.

Invalid orders → HHHOHO, HHHHHOO, HOOHHH

```java
private int hydrogen = 0, oxygen = 0;

public synchronized void hydrogen(Runnable releaseHydrogen)
                                  throws InterruptedException {
    while(hydrogen == 2 && oxygen == 0) wait();
    // 3rd hydrogen waits for the previous molecule

    releaseHydrogen.run(); // print "H"
    hydrogen++;
    if(hydrogen == 2 && oxygen == 1){
        hydrogen = oxygen = 0;
    }
    this.notifyAll();
}
```

HH 1H → wait

} reset or complete molecule

```java
public synchronized void oxygen(Runnable releaseOxygen)
                                  throws InterruptedException {
    while(oxygen == 1 && hydrogen < 2) wait();
    // 2nd oxygen waits for the previous molecule

    releaseOxygen.run(); // print "O"
    oxygen++;
    if(hydrogen == 2 && oxygen == 1){
        hydrogen = oxygen = 0;
    }
    this.notifyAll();
}
```

HO '1' wait

OH '0' wait

O '0' wait

} reset on complete molecule

Hyd = 0 1 2 0 Oxy = 0 1 0

Hydrogen

{ "H"

hyd++

"H"

Oxygen

{ 1st oxy

"O"

{ oxy++

wait()

O wait OHHHH

"O HH

Print FizzBuzz Multithreaded: LC 1195

```
Input: n = 15
Output: [1,2,"fizz",4,"buzz","fizz",7,8,"fizz","buzz",11,"fizz",13,14,"fizzbuzz"]
```

3 5 6 9 10 9 15

Case1: If n%3 != 0 && n%5 != 0 ⇒ Number itself

Case2: If n%3 == 0 && n%5 != 0 ⇒ "Fizz"

Case3: If n%3 != 0 && n%5 == 0 ⇒ "Buzz"

Case4: If n%3 == 0 && n%5 == 0 ⇒ "FizzBuzz"

```java
class FizzBuzz {
    private int n;
    private int counter = 1;
    public FizzBuzz(int n) {
        this.n = n;
    }

    public synchronized void fizz(Runnable printFizz) {
        while(counter <= n) {
            if(counter % 3 != 0 || counter % 5 == 0) {
                wait(); continue;
            }
            printFizz.run(); // print "fizz"
            counter++;
            notifyAll();
        }
    }

    public synchronized void buzz(Runnable printBuzz) {
        while(counter <= n){
            if(counter % 3 == 0 || counter % 5 != 0) {
                wait(); continue;
            }
            printBuzz.run(); // print "buzz"
            counter++;
            notifyAll();
        }
    }
```

```java
    public synchronized void fizzbuzz(Runnable printFizzBuzz) {
        while(counter <= n){
            if(counter % 3 != 0 || counter % 5 != 0) {
                wait(); continue;
            }
            printFizzBuzz.run(); // print "fizzbuzz"
            counter++;
            notifyAll();
        }
    }

    public synchronized void number(IntConsumer printNumber) {
        while(counter <= n){
            if(counter % 3 == 0 || counter % 5 == 0) {
                wait(); continue;
            }
            printNumber.accept(counter); // print n
            counter++;
            notifyAll();
        }
    }
}
```

Producer Consumer Problem

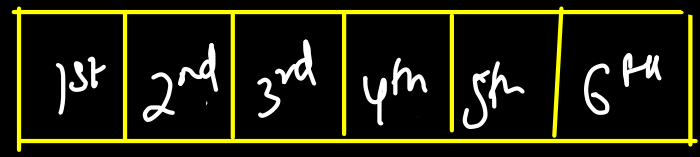
Producer

↳ man ⇒ 6 balls buffer

livestream {ball by ball}

1 over = 6 ball

Consumer



wait()

↳ size == man {ball == 6}

full → overflow

wait()

↳ size == 0

empty → underflow

Sol =

Producer Code- solution

```
void producer( void )
{
  wait ( empty );
  wait(S);
  Produce_item(item P)
  buffer[ in ] = item P;
  in = (in + 1)mod n
  signal(S);
  signal(full);

}
```

Consumer Code- solution

```
void consumer(void)
{
  wait ( empty );
  wait(S);
  itemC = buffer[ out ];
  out = ( out + 1 ) mod n;
  signal(S);
  signal(empty);
}
```

Application of using locks ~> counting semaphore

```
private int bufferSize = 6;
int ball = 0;
Queue<Integer> over = new ArrayDeque<>();

public synchronized void producer(){
    if(over.size() == bufferSize) this.wait();
    over.add(ball++);
    this.notify();
}

public synchronized void consumer(){
    if(over.size() == 0) wait();
    System.out.println(over.remove());
    this.notify();
}
```

Observer Pattern

DevOps

Webhooks

Reader - Writer Problem

Multiple readers, multiple writers

Read - Read Conflict No

Write - Read Conflict Yes

write - write conflict Yes

Payment ⟶ Payment successful ⟶ Recharge

Payment metro

read ⟶ Payment successful

read ⟶ Recharge waiting

data consistent ≠

Payment successful
↓
Payment Jugpay
↓
writer 1

Recharge waiting
↓
DMRC
↓
writer 2

writer ⇝ binary semaphore

reader ⇝ counting semaphore

```
wait (mutex);
rc ++;
if (rc == 1)
wait (wrt);
signal(mutex);
.
. READ THE OBJECT
.
wait(mutex);
rc --;
if (rc == 0)
signal (wrt);
signal(mutex);
```

Reader

```
wait(wrt);
.
. WRITE INTO THE OBJECT
.
signal(wrt);
```

Writer

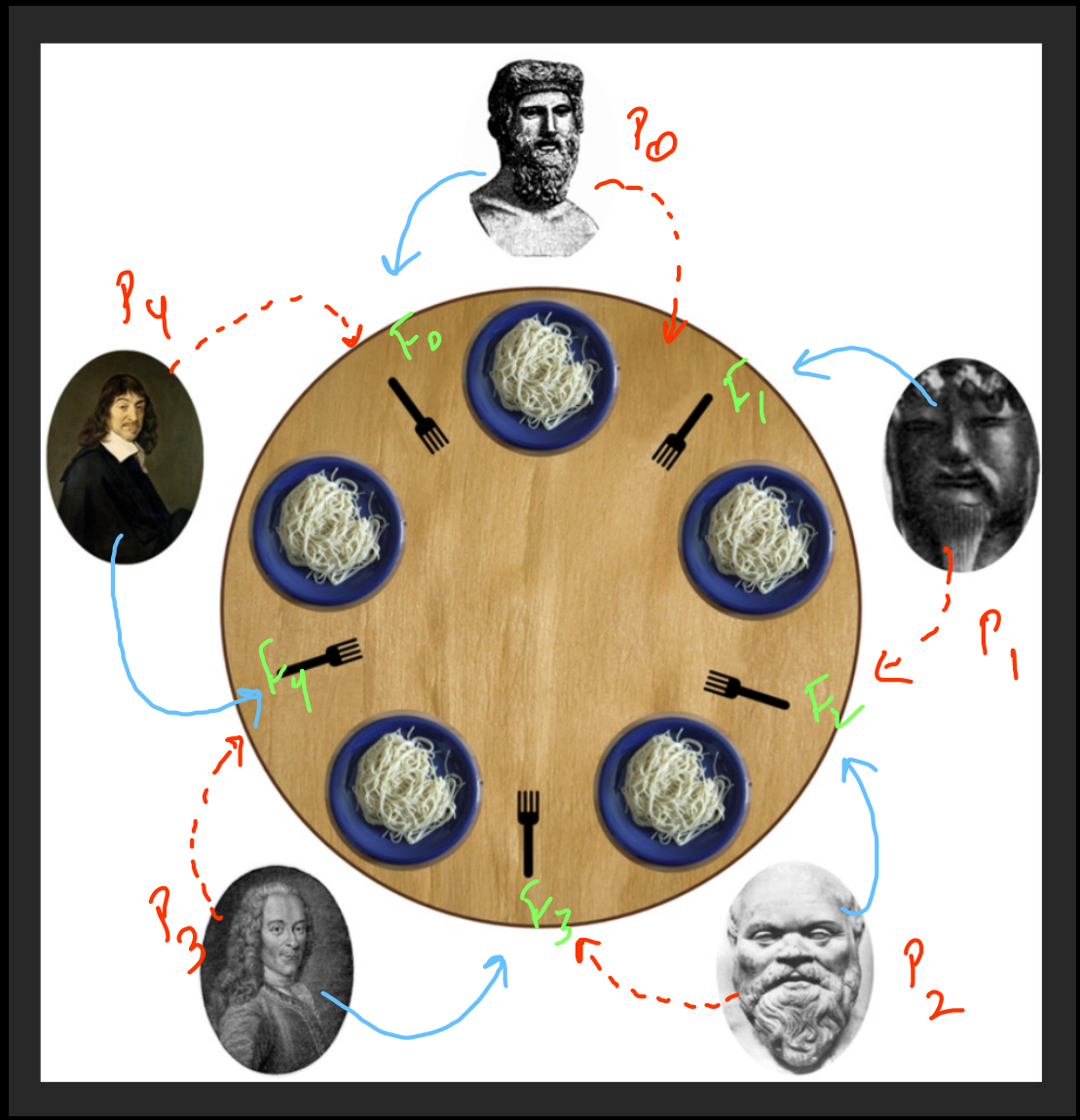
Java Application

```
private int readerCount = 0; // counting semaphore
private boolean writerCount = false; // binary semaphore

public synchronized void reader(){
    while(writerCount == true) this.wait();
    readerCount++;
    System.out.println("Reading");
    readerCount--;
    if(readerCount == 0) this.notifyAll();
}

public synchronized void writer(){
    while(writerCount == true || readerCount > 0) wait();
    System.out.println("Writing");
    writerCount = false;
    this.notifyAll();
}
```

Dining Philosophers Problem

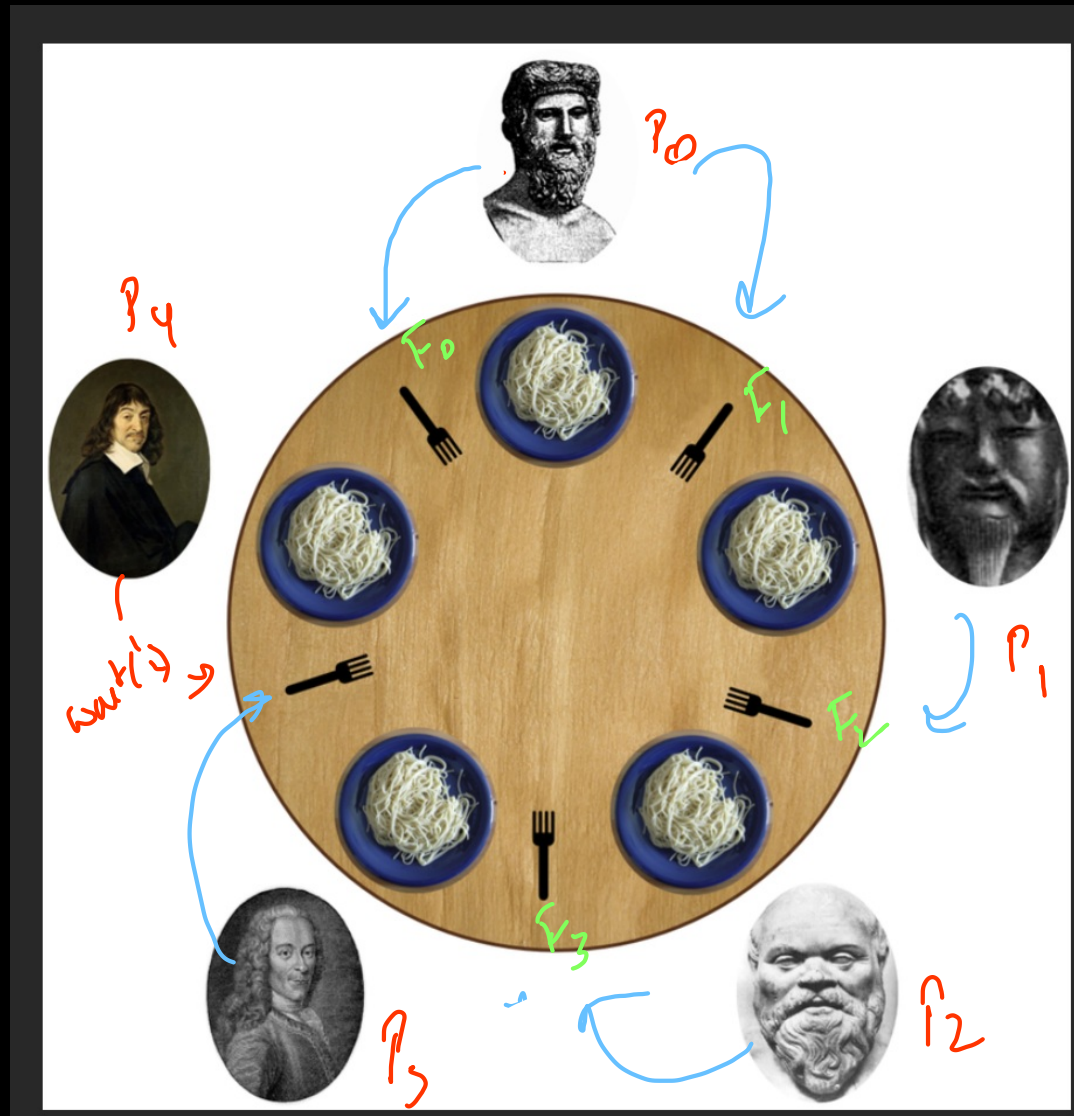
5 philosophers

5 chopsticks




```
public int forks[] = {-1, -1, -1, -1, -1};

public synchronized void acquire(int personId){
    int left = personId, right = personId + 1;

    if(forks[left] != -1 && forks[left] != personId) this.wait();
    if(forks[right] != -1 && forks[right] != personId) this.wait();

    System.out.println("Eat Food");

    forks[left] = forks[right] = -1;
    this.notifyAll();
}
```

↳ Deadlock situation might occur




```java
class DiningPhilosophers {
    public int forks[] = {-1, -1, -1, -1, -1};

    public synchronized void acquire(int personId){
        int left = personId, right = personId + 1;
        if(personId == 4) {left++; right--;} // Solution 1: to avoid deadlock
        // if(personId % 2 == 0) {left++; right--;} Solution 2

        if(forks[left] != -1 && forks[left] != personId) this.wait();
        if(forks[right] != -1 && forks[right] != personId) this.wait();

        System.out.println("Eat Food");

        forks[left] = forks[right] = -1;
        this.notifyAll();
    }
}
```