Sir

```
public class Box{
    private double width;
            "       "  height
            "       "  depth

    Box(int width, int height, int depth){
        this.width = width
        this.height = height
        this.depth = depth;
    }
    double getVolume(){
        return width * height * depth;
    }
    pub s v m(...){
        Box box = new Box(10, 20, 30);
        s.o.p(box.getvolume());
```

Q. Create a new class called calculator with following methods.

1. static method called powerInt(int num1, int num2). This method should return num1 to the power num2
2. A static method called powerDouble (double num1, int num2).
3. Invoke both methods and test the functionalities. (use Math.pow(double, double).

Ans:

```
public class calculator{
    public static int powerInt(int num1, int
                                num2){
        return (int) Math.pow(num1, num2);
    }
    public static double powerDouble(double
                num1, int num2){
```

```
return Math.pow(num1, num2);

}
public static void main(      ) {
    s.o.p(powerInt(12, 3));
    s.o.p(powerDouble(1.5, 2));

}
}
```

31/1

Constructor ~~Changing~~ Chaining

```
class A {
    A() {
        this(10, 20);   --> [should be in the 1st line of constructor]
    }                        A(int a, int b) {
    A(int a) {
        this();              }
    }
    p s v m(      ) {
        A a = new A(10);
    }
```

- Recursive call should not be there.
- super refers to parent class.
- this & super এক constructor এ একসাথে থাকবে না

Polymorphism

- static
  - → compiletime polymorphism
  - → early binding
- dynamic
  - → Run time polymorphism
  - → late binding

Static

( )

```
int add(int a, int b, int c){

}
int add(int a, int b){

}
add(10,20) → during compile its decided which method is called
```

* method has same name, diff no of parameter → method overload
  or diff types of parameter
  method overloading happens in same class/inheritance.
* Constructor checks name same then check no of param. then type of param. but not return type of method.

```
class A {
    int add(int a, int b){                    priority 1

    }
    void add(long a, long b){                 priority 2 (same int and long)

    }
    void add(int a, float b){                 priority 3

    }void add(float a, int b){

    }
    psvm(  ){
        A a = new A();
        a.add(5,10);

}
```

Inheritance

```
class A {
    int a = 10;
    void m1() {
    }
}
```

```
A() {
    s.o.p("A const");
}
```

Properties of A goes to B.

```
class B extends A {
    int b = 20;
    void m2() {
    }
}
```

```
B() { s.o.p("B const"); }
```

(super)

```
class m {
    p s v m() {
        B b1 = new B();
    }
}
```

④ if parent class has parameterized constructor. child has no constructor ⇒ error.

```
class A {
    A() {
        this(10)
    }
    A(int a) {
    }
}
```

4/2/25

Inheritance

① Single Inheritance
② Multi level Inheritance

```
class A {
    A () {
        s.o.p ("cons");
    }
    static {
        s.o.p ("Hello");
    }
    static {
        s.o.p ("Hello");
    }
}
```

Not inherited

static block

Instance initialization block

```
class B extends A {
    static {
        s.o.p ("static B");
    }
}
class C extends B {
    B b = new B ();
}
```

sequence of execution:

↓ static block
Instance initialization block,
Construction.

method overriding:

```
class A {
    void m1() {
    }
    double m2() {
        return 1.2;
    }
    A m3() {
        return new A();
    }
}
```

```
class B extends A {
    void m1() {
    }
    String m4() {
        return new String();
    }
    A m3() {
        return new B();
    }
    or
    B m3() {
    }
}
```

when using over rridden, you can increse visibility, or same but not decrease

* same method in parent & child class
* exact same primitive, data type or reference
* Child has same return class as parent in reference
* Can't reduce the visibility of method.
only increse or same visibility.

```
Class A {
    public void m1(){

    }
}
```

```
class B extends A {
    public int m1(){

    }
}
```

(*) Object class is the parent of all classes of Java.

(*) @override is used to check method is overridden or not.

(*) Runtime poly morphism → which method is called decided during run time after object is created.

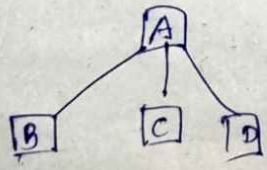
```
class new{
    p s v m( ){
        A b = new A();
        b.m1();
    }
}
```

Term I

1) Features of Java
2) JVM architecture (complete) (*) (*)
3) Polymorphism.
4) Constructor.
5) Inheritance.

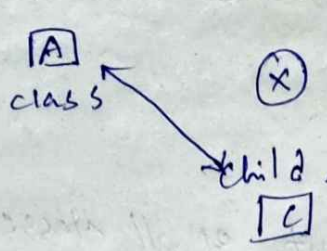
Inheritance

(3) Hierarchial inheritance



& Diamond problem in java..

Multiple Inheritance



(x) (*) not possible using java

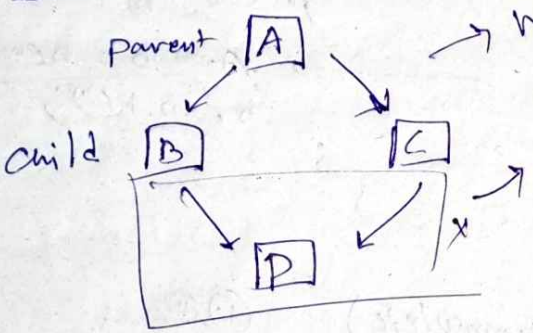
```
class A{
void m1(){
}
}

Class B{
void m2(){
}
}

class C extends A,B{
}
```

(*) possible through interface.

Hybrid Inheritance



7/2

```
class Animal{
    void eat(){
    }
    void sleep(){
    }
}
class Bird extends Animal{
    @override
    void eat(){
    }
    void sleep(){
    }
    void fly(){
    }
}
```

Q. class Person {
string name;
Person (String name) {
this.name1 = name;
}
}
class Employee extends Person {
double Salary;
employee (double salary) {
}
}

Index
item item page, cover for every week
week 1 (example) and week 2
1. statement (eg. wap to sum)
2. program
3. op

11/2
final → can be used in variable, class, method.

final class A {
}

class B extends A {
//compile time error
}

class A {
final int a=10;
a=20; //C.E. ← can't change
int b=a;
a++; ← can't change
}

```
class A{.
final int m1(int a){
    return a*5;
    }
}.
```

overloading possible.

but →

```
class B extends A{.
final int m1(int a){ ← not allowed
    }
}.
```

Abstract class

```
class Fan {
    public int blade();
}
```

* Abstract keyword is opposite of Final keyword.

```
abstract class A {
    void m1() {

    }
    abstract void m2();

}
class B extends A {

}
```

* Not able to create object of abstract class

Compiletime polymorphism → early binding
meth overloading

Runtime polymorphism → late binding
method overriding

Q. can we use 'super' in static method.
no.

```
class A {
    static void m1() {
    }
}
```

For static meth JVM creates an object in context area. Common area that all class can access.

* static method can't be over ridden.
it is called — 'method hiding'

```
class A {
    static void m1() {
    }
}

class B extends A {
    static void m1() {
    }
}

B.m1();
```

static method can be overloaded.

Q. We can't over ride constructor.

Q.
```
class Fruit {
    String name, taste;
    void eat() {
        s.o.p("Name is %s taste %s",
              name, taste);
    }
}

class Apple extends Fruit {
    void eat() {
        s.o.p("Name: Apple, Taste: good");
    }
}

class Orange extends Fruit {
    Apple() {
```

```
name = "Apple";
taste = "sweet";
}
void eat() {
    s.o.p(name + " is " + taste + " in taste");
}
class Orange extends Fruit { }
public class Solution {
    p s v m ( ) {
        Fruit fruit = new Fruit();
        Apple apple = new Apple();
        apple.eat();
    }
}
```

Q.
```
class shape {
    void draw() {
        s.o.p(" Drawing shape");
    }
    void erase() {
        s.o.p(" Erasing shape");
    }
}
class Circle extends shape {
    void draw() {
        s.o.p("Drawing circle");
    }
    void erase() {
        s.o.p(" Erasing circle");
    }
Class Tria angle extends shape {...}
class Square extends shape {...}
public class Solution {
    p s v m (String[] args) {
        Circle obj = new Circle();
        obj.draw();
    }
```

* To achieve 100% security use interface
" " abstraction " abstract

Interface → can have abstract method only (upto Java 1.7)

```
interface A{
(p s f) int i=10; → public, static, final by default for variable.
        void m1();
}
class B implements A {
    public void m1(){
    }
}
```

* you can't reduce visibility of method ⇒ public method we have to write.

* Interface will generate .class file while compiling because its abstract class.

* [class C implements A, B] multiple inheritance is possible by interface.

* both side ~~extends~~ Interface

```
I extends I
```

* class implements I
  class extends class.

```
Interface A {
    void m1();
}
```

```
interface B extends A {
    void m1();
}
```

* Now, inside Interface we can implement methods. (from Java 1.8)

```
Interface A{
    default void m1(){
    }
}
```

This method is accesed by:

20/2

Package

```
package p1;
public class A{
    void add(){...}
}
```

```
import p1.A;
package p2;
class D{
}
```

root
- P1
  - A.java
  - A.class
  - D.java

P1
A.class

javac -d. *.java
all

eg: javac -d . P1\A.java
then dir P1 created
inside it A.class generated
to run it > at root folders
java P1.Ad

(*) singleton class: max 1 object will be created. Used private constructor

(*) Outer class can't be private. Inner class can.

(*) Default - class can be accessed within the package.
protected - Child class in diff erent package can access it. within the package or outside the package within child class.

(*) Static import used to access only static method of import package.

Sub package
package p1.p2;

Package creation

① create directory "mypack"
② Inside it write "MyPackage.java"
③ go to parent dir of "mypack"
④ <command> javac -d . mypack\MyPackage.java
then MyPackage.class generated
⑤ Run> java mypack.MyPackage

Exception Handling

Runtime error

object

↑

Throwable is parent of all class exception

↓ Exception ↓ Error ×

Try
Catch
finally

throw
throws

→ individually they cant go either try or catch, t or f, t or f or c.

c or f

OS shut down that time finaly will not execute.

Overriding

```
class Animal {
    public void sound() {
        s.o.p("Animal sound");
    }
}
class Dog extends Animal {
    public void sound() {
        s.o.p("Dog bark");
    }
}
public class Main {
    p s v m(String[] args) {
        Animal obj1 = new Animal();
        Dog obj2 = new Dog();
        obj1.sound(); → Animal sound
        obj2.sound(); → Dog bark
    }
}
```

Animal obj = new Animal();

- Animal: Class types is referance type, determines which methods can be accesed using 'obj'
- obj: variable stores reference of new object.
- new: allocate memory in heap
- Animal(): this calls the constructor of Animal class to initialize the object

in `Animal obj = new Dog();`

reference type Animal, but actual object is Dog

(upcasting)

Method Overloading

```
class met {
    void m1(int a, int b){
    }
    void m1(int a, int b, int c){
    }
}
public class main {
    p s v m( ){
        met obj = new met();
        s.o.p(obj.m1(1,2));
    }
}
```

Inheritance

```
class Animal {
    String name = "All animals";
}
class Dog extends Animal{
    String breed = "Labra";
}
public class Main{
    p s v m( ){
        Dog mydog = new Dog();
        s.o.p(mydog.name);
                ↳ All animal
        s.o.p(mydog.breed);
                ↳ Labra
    }
}
```

Interface

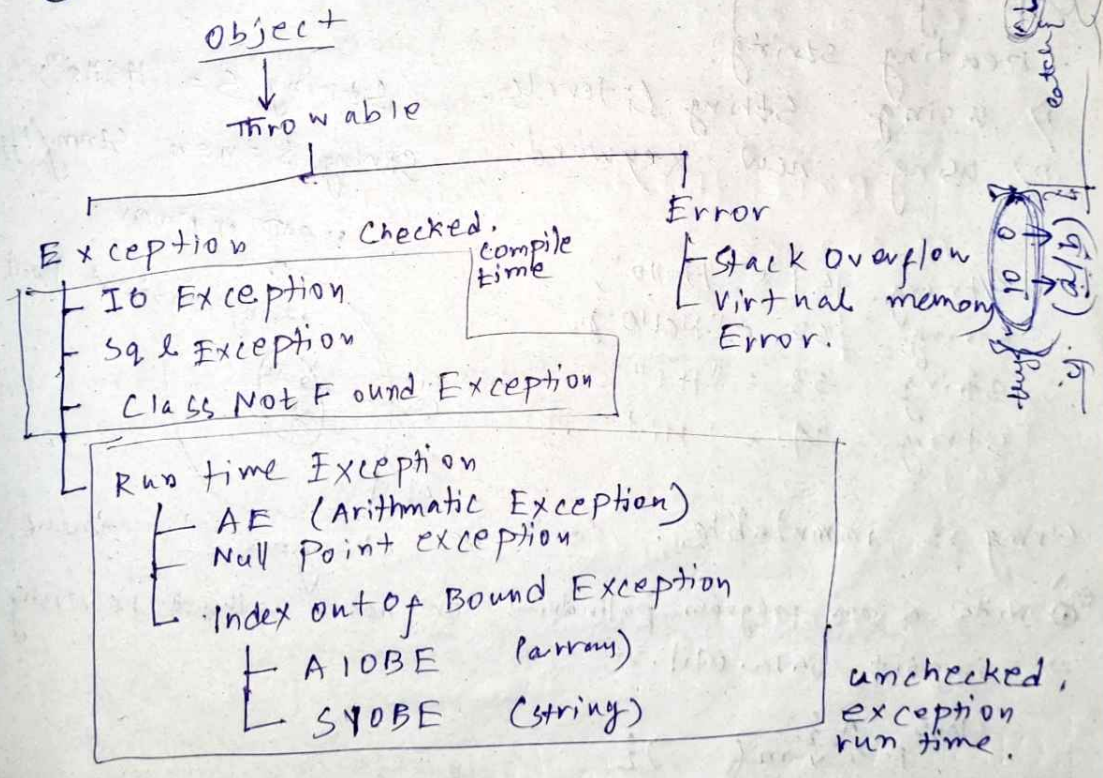
```
interface Animal {
    void makeSound();
}
class Dog implements Animal {
    public void makeSound() {
        S.o.p( );
    }
}
public class main {
    p s v m( ) {
        Dog obj = new dog();
        obj.makeSound();
    }
}
```

op

3/2

① Use of throw ?

② Multiple catch block use ?



③ Throws used in checked exception.

④ How to create custom exception.

Custom Exception

```
class InvalidAgeException extends Exception{
    public InvalidAgeException(){
        super();
    }
}

class A {
    psvm(){
        if(age<18)
            throw new InvalidAgeException();
    }
}
```

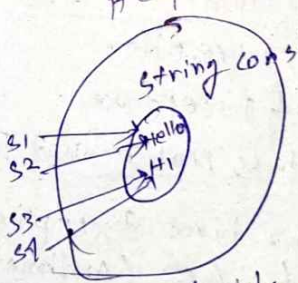
String handling

* String is final class.

Creating string:

1) using String literals. String S = "Hello";
2) using new keyword. String S = new String("Hi");

```
String s1 = "Hello";
String s2 = "Hello";
String s3 = "Hi";
String s4 = "Hi";
```



String is immutable. can't be changed its value.

Q) Write a java program palindrome or not. without reversing.

⇒
```
import java.util.*;
class A{
    psvm(){
        Scanner sc = new Scanner(System.in);
        String s = sc.next();
        int n = s.length()
        int flag=0;
        for(int i=0; i<n/2; i++){
            if(s.charAt(i) != s.charAt(n-i)){
                flag=1;
```

```
        break
    if (flag == 1)
        s.o.print ("not palindrome");
    else s.o.p (palindrome)
}
}
```

Q. 2. concatinate two strings. op. lowercase duplicate character ommitted.

```
String m1 ( String s1, String s2) {
    s1 = s1.ToLowercase()
    s2 = s2.ToLowercase()
    if ( s1.charAt (s1.length()-1 == s2.charAt(0)
        return s1.substring(0, s1.length()-1) + s2;
    else
        return s1 + " " + s2;
}
```

Q. 3

```
String m1 ( String s) {
    n = s.length();
    (n % 2 == 0)? return s.substring(0,(n/2));
return "null";
}
```

(*) indexOf() lastIndexOf() search in string.

7/1

```
String m1 (String s1, String s2) {
    int n1 = s1.length();
    int n2 = s2.length();
    (n1 > n2)? return s2+s1+s2 : return s1+s2+s1;
}
String m2 ( String s1) {
    if ( s1.charAt(0) == 'x' && s1.charAt (s1.length()-1) == 'x')
        return s1.substring (1, s1.length()-1);
else return s1;
```

```
String m3(String s1, String s2){
    String res = "";
    for(i=0;i<s1)
    if(s1.length() <= s2.length()){
        for(int i=0; i < s1.length(); i++)
            res += s1.charAt(i) + s2.charAt(i)
    }
    res += s2.substring(i+1, s2.length());
    }
    else {
        for(int i=0; i<s2.length(); i++)
            res += s1.charAt(i) + s2.charAt(i)
        res += s1.substring(i+1, s1.length());
    }
    return res;
}
```

11/3

Multi threading

Multitasking

- Process based MT
- Thread based Multitasking

Creation of Thread

1. By extending Thread class
2. By implementing Runable interface.

```
class thread1 extends Thread{
    public void run(){ → override run method
        for(int i=1; i<=5; i++){ task of thread.
            s.o.p("child thread");
        }
    }
```

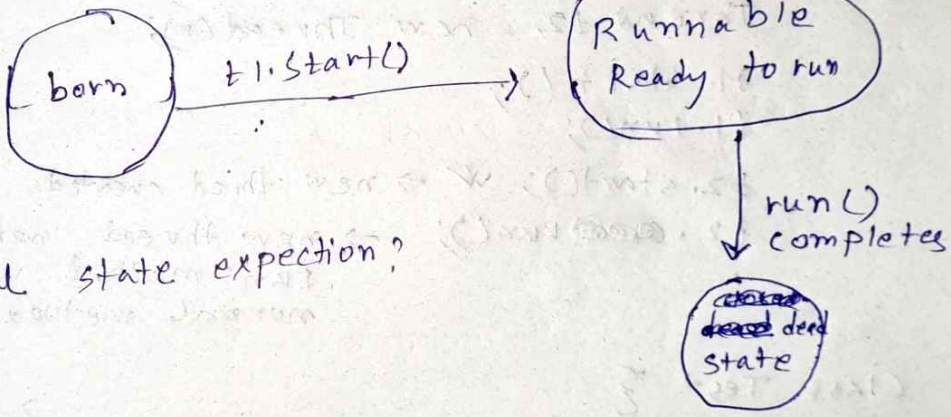
```
Class Main {  ⟶ main tread
    p s v m ( ) {
        thred1 t1 = new thread1();  → thred instances.
        t1.start();  ⟶ at this point executes run(). 2 threads there. main, t1
        for(i=1; i<=5; i++) {
            s.o.p("Main thread");
        }
    }
}
```

⑥ use of start();
1. Thread introduces to thred scheduler
2. Call run() method.

⑦ Can we overload the run method?
→ Yes. over ridden method can be overloaded.

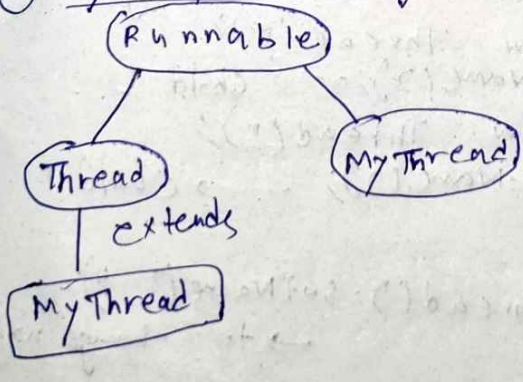
⑧ Can we overload start method?

Thread1 t1 = new Thread1();



④ Illegal state expection?

② By implementing runable interface



```
class MyRunnable implements Runnable {
    public void run() {                          // define thread
        for(i=1; i<=5; i++) {                    // Task of thread
            s.o.p("Child Thread");
        }
    }
```

```
class Test {
    p s v m( ) {
        MyRunnable r = new MyRunnable();
        Thread t1 = new Thread(r);
        t1.start();
    }
}
```

(*)

```
Thread t1 = new Thread();
Thread t2 = new Thread(r);
t1.start();
t1.run();
t2.start();   // → new thred created
t2.run();     // → new thread not created
              //   run' method work as
              //   normal method.
```

```
Class Test {
    p s v m( ) {                                 // → main thred
        s.o.p(Thread.currentThread().getName());
        Thread t1 = new Thread(r);
        s.o.p(t1.getName());   ⟶ child 0
        Thread t2 = new Thread(r);
        s.o.p(t2.getName());   ⟶ child 1
    }
```

```
Thread.currentThread().setName("n");
                // → to change name
```

SetPriority:

```
psvm( ){
    Thread.currentThread().setPriority(8);
    Thread t1 = new Thread(r);
    t1.setPriority(4);
}
```

MAX_PRIORITY (10)
MIN_PRIORITY (1)
NORM_PRIORITY (5)