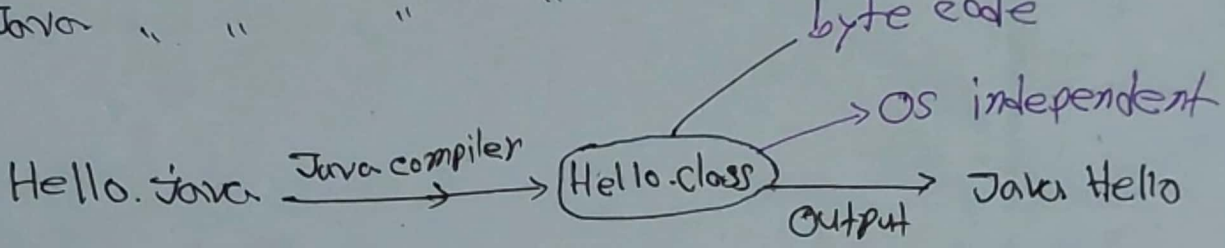
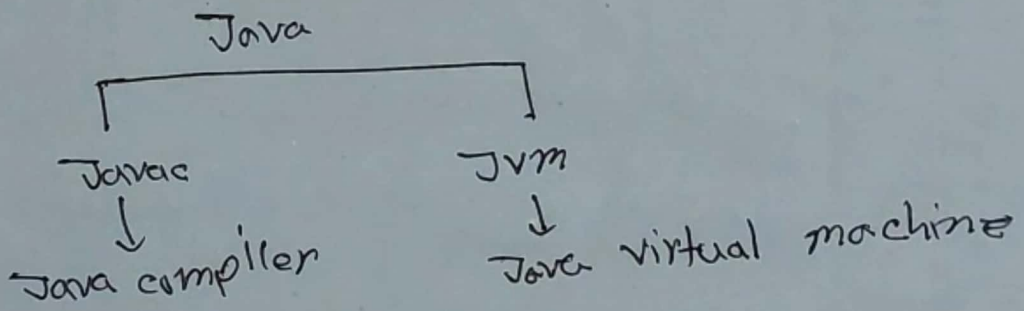
1.2

Why learn Java

C এর প্রোগ্রাম উই রান করে না।

Java " " " " "





Java → Both (compiler + interpreter) language

Object oriented programming mainly যেটা way তার Java হচ্ছে একটা language তার যে way যে জন্য আমরা জাভা (Java) language use করবো।

একটা file এ একটা public class থাকবে।
আর public class এর নাম exactly file এর নামের মতো হবে।

HelloWord.java

```
public class HelloWorld {
    public static void main (String args []) {
        int id=101;
        System.out.println (id);
    }
}
```

```
public class Hello {
```
✗ error

HelloWord.java

```
class Helloword {
```
✗ error

```
}
public class Hello {
```
error

```
}
```

Pubic class এর নাম Exactly file এর নাম হবে।

L3

convention: error → code বলতে বুঝায়

Class naming: HelloRakib, HelloWorld, World

Variable " : Area, areaAndSquare, radius

method " : squareOfEven, evenAndOdd

Rules: error → code বলতে বুঝাবে না.

for naming, A-Z, a-z, 0-9, $ $ (_) under score

doesn't start digit (0-9)

Memory

~~int 4 byte (32 bit)~~

~~char~~

short : 2 byte (16 bits)

~~Integral : 1 byte (8 bits)~~

int : 4 byte (32 bits)

long : 8 byte (64 bits)

Float : 4 byte (32 bits)

We use unicode (International: 0-255)

type casting

~~ছোট~~ to বড়

```
short s1 = 10-00;
int   i2 = s1;        // Auto casting

int   i2 = 1000;
short s2 = (short) i2; // Explicit casting
```

initialize:

```
float a = 10.2f
double a = 10.2
long = 10000000000L
```

Array

```
public class Array {
    public static void main (String args [])
    {
        int arr [] = new int [10];   declare
                                     Array size

    or
        int arr [];
        arr = new int [10] → by defult সব 0 হয়ে থাকা

        arr [9] = 5;   // initialize

        for( int i=0;  i < arr.length;  i++) {
            system.out.println ( arr [i]);
        }

        int arr [] = { 2, 4, 56, 7, 9 };   // initialize

        or
            int arr [];
            arr = new int [] { 2, 4, 56, 7, 9 }
    }
}
```

```
int [] arr2, a2, a4; // All are array

int arr2[], a2, a4, a6[]; // arr2[], a6[] are array
```

for each loop

```
for (int x: arr)
{
    System.out.println(x)
}
```

output: 2, 4, 58, 7, 9

Syntax: for (type variable : collection)

- type declare area type
- collection → array name

for each loop শুধু Array তে use হয়।

14

class point{

int x, y;
- member
- instance variable

satic int a; ⟶ global

point.a = 10 ✓ ⟶ reason for static

point.x = 25 ✗ ⟶ " " not static

point P1 = new point();

point P2 = new point();

P1.x = 1

P1.y = 2

P2.x = 5

P2.y = 5

}

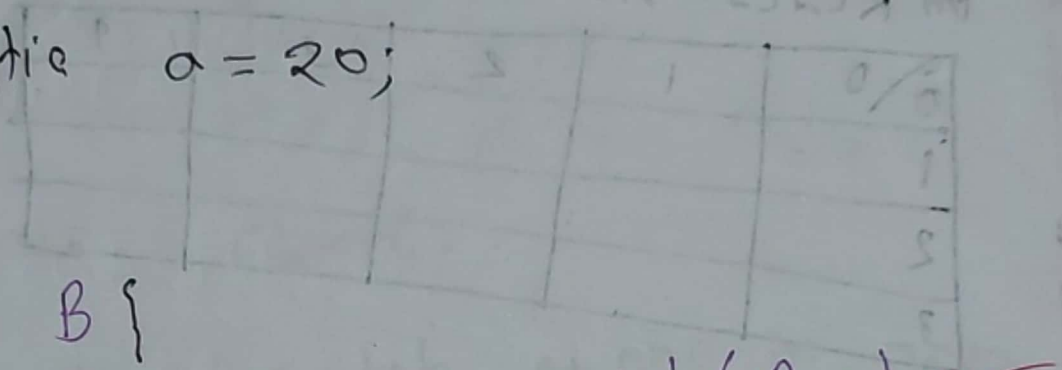
	x	y	
instance	1	2	P1
	5	5	P2
			...
			...

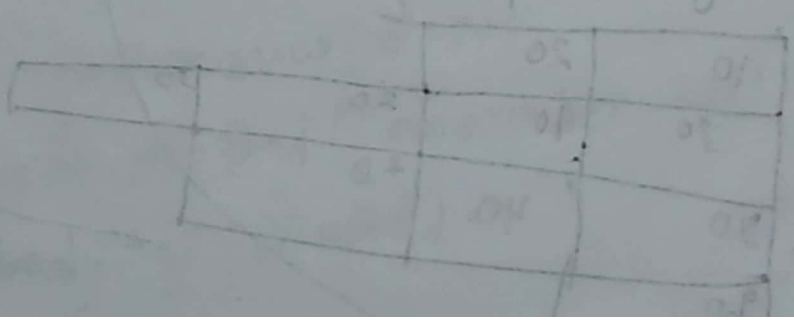
point.java --compile / javac--> point.class --JVM / java point 1 2 3--> point.main()

↳ main টা static তাই class এর নাম দিয়ে call করা যায় গেছে

public static void main(String args[])

```
class A {
    static a = 20;
}
class B {
    ppoint (a);  ✗  ||  print (A.a)  ✓
    int b = 10;
    print (a)  ✓
}
```





L9 Multy-Dimentional Array

int R[][] = new int [4][5];

int R[][] = new int [4][5];

	0	1	2	3	4
0					
1					
2					
3					

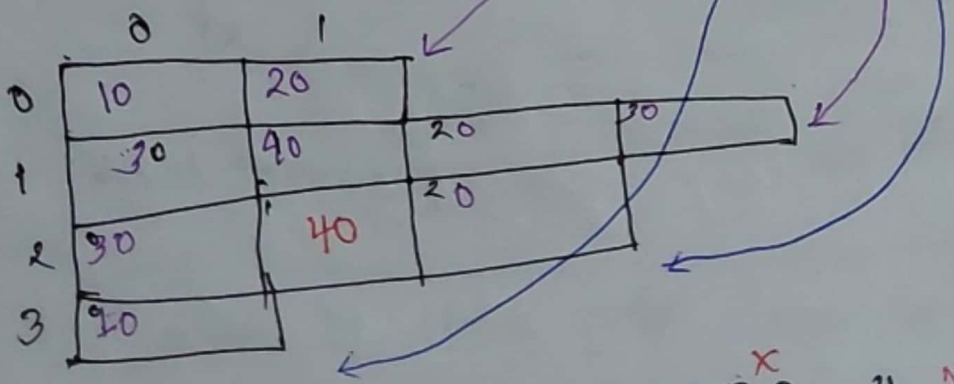
int R[][] = new int [4][]

int R[0] = new int [3];

int R[1] = new int [4];

int R[2] = new int [3];

int R[3] = new int [1];



R[2][1] = 40, R[3][1] = 4 ✗

int R[][] = {{10, 20}, {30, 40, 20, 30}, {30, 40, 2}, {10}}

↓ initialized

Class & Object

- class is the basis for OOP in Java
- it defines a new data type

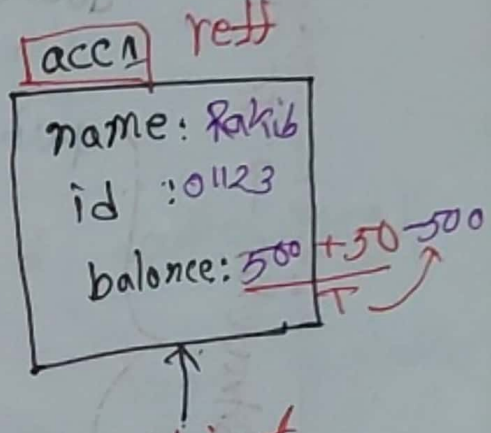
Class [Action
        behaviour

```
class className{

    datatype instance-variable;
    ---------- ---------- ----
    ---------- ----------
    datatype instancevariableN;
    returntype method(parameter-list){
        //body of method
    }
}
```

```
public class BankAccount {
    public string name;
    public String id;
    public double balance;

    public void deposit(double amount) {
        balance += amount;
    }

    public void withdraw(double amount) {
        if (amount < balance)
            balance -= amount;
    }
}
```

instance variable

```
public class Test{
public static void main(String args[])
{
    BankAccount acc1 = new BankAccount();

    acc1.name = "Rakib";
    acc1.id   = "01123";
    acc1.balane = 500;

    acc1.deposit(50);
    acc1.withdraw(500);
```




```
~~BankAccount~~
    System.out.println(acc1.name);
    system.out.println(acc1.id);
    system.out.println(acc.balane);
}
}
```

output: Rakib
01123
50 ⟶ 500 +50 −500

☑ Instance constructor

```
class Student{
    string name;
    int    id ;
    double cgpa;          } instance variable

    public Student( string n, int i, double c){
        name = n;
        id = i;
        cgpa = c;
    }
    void display(){
        System.out.println(name);
        System.out.println(id);
        System.out.println(cgpa);
    }
}

class main{
    public static void main(String[] args){
        Student s1 = new student("Rakib", 101, 3.75);
        s1.display();
        Student s2 = new Student("Raz", 102, 3.50);
        s2.display();
    }
}
```

constructor → for the class to initialize instance

output: Rakib
101
3.75

output: Raz
102
3.50

এইটি লিখা যাবে Java ✓

যদি instance variable এর নাম আর constructor এর নাম একই হয় তাহলে constructor এ declare করার আগে this.variable লিখতে হবে।

this. instance variable

instance constructor

```
public student (string name, int id, double cgpa){
    this.name = name;
    this.id = id;
    this.cgpa = cgpa;
}
```

local variable

local variable > instance variable

বাকি কোড আগের মতো

local এর priority বেশি তাই int instance এর priority বাড়ানোর জন্য this.instance variable লিখা হয়েছে।

```
student s1 = new student(... ... ..)
          ||
      new student(... ... ...)
```

Default value:

Boolean : false

All primitive : 0

string : Null

Scope of variable → and Lifetime

```
public void calculate( double balance )
{
    if ( balance > 1000 )
    {
        float interest = 0.05f;
    }
    else
    {
        interest = 0.02f   X
    }
}
```

Block 1, Block 2, Block 3

এখানে balance variable টা Block 1 এর সব জায়গায় use করা যাবে। interest টা শুধু Block 2 তে use করা যাবে কিন্তু Block 3 তে use করা যাবে না। কারণ - যেই Block এ Declare করা হয় - সেই Block এ আর তার ভেতরের Block এ use করা যাবে।

এখানে যদি interest টা Block 1 এ Declare করা থাকতো তাহলে Block 2 এ; Block 3 তে use করার সময় সমস্যা হতো।

```
public void calculation ( double balance )
{
    float interest;
    if ( balance > 1000)
    {
        interest = 0.05f;
    }
    else
    {
        interest = 0.02f;
    }
}
```

Block 1

Block 2

Block 2

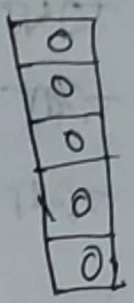
যেমন `for( int i=0; i<10; i++ )` { `print("%d", i);` } `print("%d", i);` ✗

Array

① int [] arr = new int [5];

reference — array creat

arr



② int a[] object create

a[0] = 5 ✗ Array creat

① # 1 কাজ করবে কারণ ঘরে creat করবে তারে arr নাম দিয়ে বা object দিয়ে refer করা আছে

কিন্তু # 2 object টা creat করছে কি also Array, কিন্তু Refer করে নাই তাই error

Object ছাড়া access করা যাবে না

St s [] = new st[5]

system.out.println(s[0].name) ✗

আগে Array এর জন্য individual object create করা লাগবে। দেখো page 6 দেখো

```
class st {
  string name;
  string id ;
  double cgpa;
}

main {

   st [] s = new st[3]; //Array objec create
   print(st[0].name) ✗
   s[0] = new st( );
   s[1] = new st( );
   s[2] = new st( );

   system.out.println(s[0].name)
   system.out.println(s[0].cgpa)

}
```

object create for access All element.

access করার জন্য individual create করতে হবে

access করা যাবে না, তাই এখানে এরর দিবে

Problem

Write a program in Java, that will call a function which will take a string as parameter / input and print the number of world.

```
public class Wordcount {
    public static void count( String str)
    {
        ~~A t] st~~
        String [] A = str.split(" ");
        System.out.println(A.length);
    }
    PSVM ( String [] args)
    {
        ~~wo~~ count("Hello Bangladesh");
    }
}
```

// output : 2

for each loop in two d Array

```
for (int [] x : toDArray)
{
    for (int y = x)
    {
        system.out.print (y+" ");
    }
    system.out.println ()
}
```

যদি ওর গুলার
সম-ওমান হয়
A [2] [3]
↓
এক
সহ ~~বাকা~~
তবে বানাও —
যেতা —

String to integer type casting;

```
string A[] = {"1, 2 4"};
int i = Integer.parseInt(A);
```

Know to Learn

OOP principle

① Abstraction
② Inheritance
③ polymorphism
④ Encapsulation

① Abstraction;

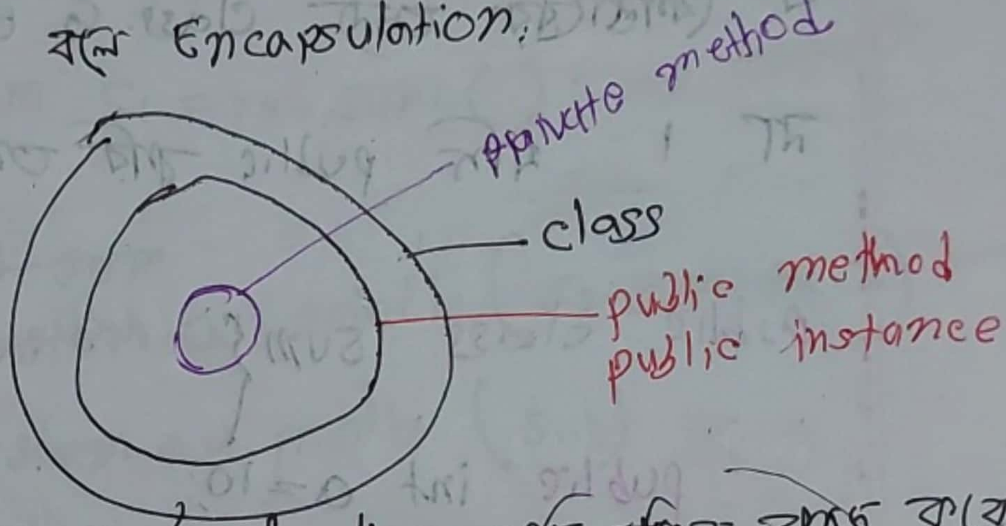
-কোনো particular topic থেকে তার properties গুলো identify করা।

যেমন গাড়ি :→ চাকা, দরজা, সিট ইত্যাদি

যেমন দুটি দুটি pont এর সাথে চিন্তা না করে topic wise করা। যেমন চাকার pont আলাদা, দরজার pont আলাদা

Encapsulation

দুটো কোড একটা মাধ্যমে মধ্যে থাকবে আর ঐ মাধ্যমকেই বলে Encapsulation.



polymorphism: একই function এর ভিন্ন কাজ করে মানে বহুরূপিতা যেমন মেথড overloading

inheritance: উত্তরাধিকার, যেমন CFO এর মধ্যে BBA and CSE student এর versity name same

class And object Encapsulation

যদি কোনো মেথড বা variable কে ও class declare করার সময় private করে রাখি তাহলে ও ফোল্ডারের অন্য class ও তা use করা যাবে না। যদি public করি তাহলে যাবে।

```
public class sum{

    public int a=10;
    private int b=20;

    public int s()
    {
        int result = a+b;
        return result;
    }

}
```

class

method

```
public class Test{
    public static void main (String [] args)
    {
        Sum S1 = new Sum ();  ⟹ object

        S1.Sum
        System.out.println (S1.Sum S1.a)
        System.out.println (S1.b)
        System.out.println (S1.S)

    }
}
```

set And get

যদি class এর variable বা মেথড private থাকে তাহলে বাইরে থেকে তা Access করা যায় না। Access করার জন্য get and set use করা লাগে

set দিয়ে value set করা যায়
get দিয়ে value পাওয়া যায়

```
public class sum
{
    public int a=10;
    private int b=20;

    public int s()
    {
      int result = a+b;
      return result;
    }
    public int getb()
    {
       return return b;
    }
```

instance

method

for private's element access from another place

```
    public void setb(int h)
    {
        b = h;
    }
}
public class Test{
    public static void main (String [] args)
    {
        sum re = new sum();
        system.out.println(re.a);   // 10
        system.out.println(re.getb);   //20
        system
         s re.setb(10);
         system.out.println( re.getb);  //10
    }
}
```

Initialization of Fields

There are three ways to initialize the field

1. Direct Assignment
2. Initialization block
3. constructors

Direct Assignment

```
public class call{
    public int a=0;
    private/public int b = 230;
    public double d = 20.0;
```

instance variable

Direct Assignment

```
    public int a;
    public int b;
    {
        a = 10;
        b = 20;
    }
```

Another way it's called block system

this দ্বারা object কে বুঝায়

```
public call (int h, int p)
{
    a = h;
    b = p;
}
public call (int a, int b)
{
    this.a = a;
    this.b = b;
}
```

constructor

local > instance

local variable

এখানে this.a use করা হচ্ছে local variable এর থেকে instance variable কে বুঝানোর জন্য।

```
public class Test {
    public static void main (String [] args) {
        Call re = new Call (10, 20);
                          constructor

        System.out.println (re.a); // 10
        System.out.println (re.b); // 20
        System.out.println (re.d); // 20.0
    }
}
```

this → a reference to self

```
public class Rec{
    private int h;
    private int w;
    public Rec(int h, int w)
    {
        this.h = h;
        this.w = w;
    }
}
```

block

এখানে this দ্বারা নিজেকেই বুঝায়।
আর এই this টা যে ~~block~~ class এই কাজে বসবে।

Access modifiers

public → সবাই access করতে

private → ~~সবাই~~ কেউই access করতে না block ছাড়া

protected → কেবল subclass and same package এর যে code access করবে।

package

যদি একই package এর হয় তাহলে easily access করবে। আলাদা না হয় তাহলে সেই package এর উপর access দিতে হলে প্রথমে import করা লাগবে তারপরে access করা।

```
package one;
public class Rakib {

    public int a = 10;
    private int b = 20;
    protected int c = 30;

    public void sum()
    {
        System.out.println(a+b+c);
    }

    private void mul()
    {
        System.out.println(a*b*c);
    }
}
```

```
package two;
pub

public class Test{
    public static void main (String [] args) {
    Rakib r= new Rakib ()  ✗ আরেক import করতে হবে।
}
```

```
package two;

import package.Rakib;  → শুধু Rakib class এ sum method যাবে।

import packageOne.*;  → সব class এ যাবে যারা সমস্যা

public class Test {
    public static void main (String [] args)
    {
        Rakib r = new Rakib ()  ✓
        r.sum()  ✓
        r.mul()  ✗
    }
}
```

static qualifier

Static access only
Static method or variable

static হলে e এর global এর মতো,
যে কেউ change করতে পারবে। আর change
করার পর latest টা contain করবে।

```
public class card {
    static string name = "Jack";
    string name2 = "king";
}

Pubic class Test {
    public static void main (string [] args)
    {
        System.out.println (card.name2) ✗
        System.out.println (card.name);
        card c1 = new card();
        System.out.println (c1.name2)
        card.name = "quin";
        spreend.name2 = "quin"; ✗
    }
}
```

static হলে তার class এর নামটিই যথেষ্ট call করার জন্য object এর

print=Jack

print = quin

print = king not quin

constant field

end encapsulation

constant value রাখার জন্য final ~~var~~ in final int size = 20; হিসেবে Declare করতে হবে। declare এর সময়ই initialize করা লাগে কারণ পরে এ final variable কে change করা যাবে না।

```
public class finaltest {

    final int size = 20;
    size++ ✗
    size = 40 ✗
    s.size = r.nextInt(); ✗

    final int s;
    public finaltest()
    {
        s = 10; ✓
        s = r.nextInt(); ✓
    }
}
```

input না করা সাপেক্ষে constructor এর ভিতরে এর মান ? initialize করা যাবে।

inheritence

যেকোনো কোনো class ব্যবহার করে সে নতুন কোনো ক্লাস তৈরি করাকেই বলে inheritence, আমি আগের class গুলোর কাজে কারও সাথে উত্তরাধিকার সূত্রে,

আর যাদেরকে ব্যবহার করে Inheritence তৈরি করো তাদেরকে বলে Ancestor classes বা base classes.

যেই Inheritence তৈরি করা হয় তাকে বলে derived classes

old class name Superclass | base class | perent class
new " " Sub class | child class | derived class

Super class কে আরো লিখতে হয়-

```
public class width=5; Rectangle {
    public int width = 5;
    public int height = 4;
    public int getArea()
    {
        return width * height;
    }
}
```

Super class

```
public class coloredRectangle extends Rectangle {
    private String color = "Red";
    public int getPeriphery() {
        return 2* width * height; }
    public String getcolor() { return color; }
    public void setcolor (String color) {
        this.color = color;
    }
}

public class Test {
    public static void main (String [] args)
    {
        coloredRectangle cRec = new colorRectangle();

        system.out.println ( cRec.getArea() ); //20
        system.out.println ( cRec.periphery() ); //40
        system.out.println (cRec.getcolor()); // Red
        system.out.println ( cRec.height ); // 4
        system.out.println (cRec.height); // 5
    }
}
```

Sub/child class

class এর নাম দিয়ে access করে colorRectangle

Access

→ super class এর সব কিছুই sub class নিয়ে যাবে কিন্তু access করতে সমস্যা হবে যদি public না থাকে।

super / parent class

```
Public class Parentone {
    private int p1 = 12;
    public int p2 = 10;

    public int getproduct() { return p1 * p2; }

    private int getLess() { return p1 - p2; }

    public int getpprivate() { return p1; }
    public void setpprivate(int p1)
    {
        this.p1 = p1;
    }

    public int getppublic() { return p2; }
}
```

```
public class Child extend ParentOne {
    private int diff;
    public int getDiff ()
    {
        diff = Math.abs ( P2 - P1 );   X
        diff = Math.abs (getLess ());   X
        diff = Math.abs (P2 - getPprivate ());
        return diff;
    }
}

public class Test {
    public static void main (String [] args) {
        Child ch = new child ();
        System.out.println ( ch.geDiff ());
    }
}
```

sub / child class

private not access

main

Reference

OOP তে primitive ছাড়া যা আছে তা Reference

```
public class Rectangle {
    public int width = 5;
    public int heigth = 4;
    public int getArea()
    {
        return width * heigth;
    }
}
```

Super class

```
public class coloredRectangle extends Rectangle{
    private string color = "Red";
    public int getperiphery(){
        return 2* width * heigth; }
    public String getColor() { return color; }
}
```

Subclass

এখানে pret Rectangle এর সব কিছু access করতে পারবে কিন্তু colorRectangle এর কোনো কিছু পারবে না। sret সবগুলোই পারবে, আর না হয় pret = sret

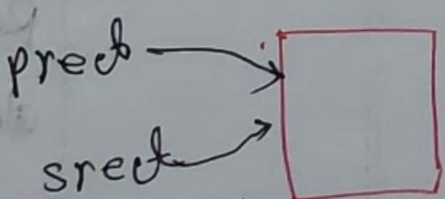
(sret = Access sub and sup] pret ≠ access sub class

class;

```
public class ReferenceTest {
    public static void main(String [] args) {

        Rectangle pret = new Rectangle();
        pret.heigth = 12;
        pret.width = 13;
        System.out.println("Area of parent: "
                                + pret.getArea);
        System.out.println(pret.getperiphery);   X
        - - - - - - - - - - - - - - - - -
        ColorRectangle sret = new colorRactangle();
        sret.heigth = 10;
        sret.width = 15;
        sret.setcolor("Yellow");
        System.out.println(sret.getArea);        //150
        System.out.println(sret.getperiphery);   300

        pret = sret

        System.out.println(pret.getperiphery);   300
        System.out.println(pret.getArea());      150
} }
```

pret: heigth, width, getArea()

sret: color ~~Red~~ Yellow, heigth ~~12~~ 10, width ~~13~~ 15



Casting / type casting

~~Subsub class এর object কে type cast করা যায়~~ ~~যায়~~ parent class ~~দিয়ে~~ । কিন্তু

Sub Super class এর object কে type cast করে Super ~~Sub~~ class বানানো যায়। কিন্তু sub class এর object কে type cast করে super class বানানো যায় না।

```
public class Test {
    public static void main (String [] args) {
        ColorRectangle srect = new colorRectangle();
        Rectangle prect = sRect;
        srect = (colorRectangle) prect
```

srect এর object type

object type colorRectangle

```
        Rectangle p = new Rectangle ();
        ColorRectangle child = (colorRectangle) p
    }
}
```

যাবে না

Super keyword

→ super keyword, this keyword এর মতোই কাজ করে কিন্তু super দ্বারা child এর immediate parent / super class এর member বা variable কে call করা যায়।

```
public class SuperTwo {
    public String name = "UIU";
}

public class ChildTwo extends SuperTwo {
    String name;
    public ChildTwo(String a, String b) {
        super.name = a;
        name = b;
    }
    public void printMe()
    {
        System.out.println(super.name);
        System.out.println(this.name);
    }
}
```

this.name self

super.name parents

ChildTwo

```
public class SuperTest {
    public static void main (String [] args) {
        ~~Childtwo p = new child (Rakib, Tamim);~~
        ~~System.out.println(~~
        Supertwo R = new supertwo()
        System.out.println (R.name); // - VIU

        Childtwo P2 =    child ("Rakib", "Tamim");
        System.out.println (R.name); // Rakib
        System.out.println (P2.name); // Tamim

        P2.printme(); // Rakib
                      // Tamim
    }
}
```

Name clash / polymorphism

- method overloading
- method overriding

→ ~~এর~~ method overloading মানে হচ্ছে ১ class ও ভিন্ন ভিন্ন নাম এর class অনেক গুলো তৈরি করা যাবে ~~কিন্তু~~ parameter আলাদা আলাদা হতে হবে।

```
class overloading {

    int x = 10;

int ~~void~~ getDouble()
    {
    ~~print~~ return ~~this.x~~ x*2;
    }

int getDouble(int a)
{
    return this.x*a;
}
int getDouble(int a, int b)
{
    return this.x * a*b;
}
```

Method overriding

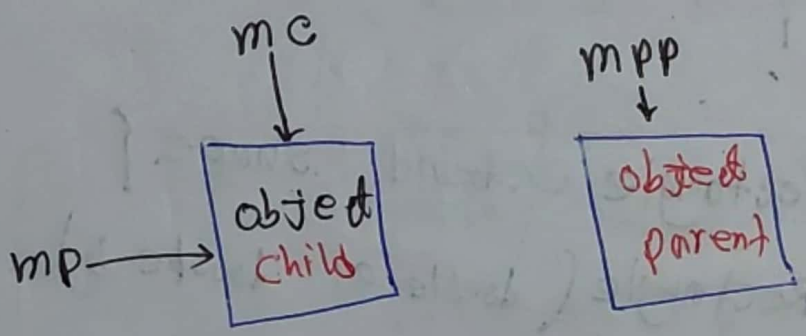
→ Static method কে override করা যায় না।

→ method override করলে আমি parent ক্লাস এর যেটা যা যেখানে আছে ঠিক তাস যেখানে child class ও declare করা।

→ override করি method ফিরে হয়।

```
for class parent {
    int x=10;
    int getDouble()
    {
        return 2*x; //20
    }
}

class child extends parent {
    int x=20;
    int getDouble()
    {
        return 2*x; //40
    }
}
```

```
public class Test {
    public static void main ( String [] args)
    {
        child mc = new child ()
        Sout ( mc.getDouble()); // 40

        parent mp = mc;
        Sout ( mp.getDouble()); // 40

        parent mpp = new parent ()
        sout ( mpp.getDouble()); // 20
}   }
```



এখানে 2 বার new করায় 2টা object তৈরি হয়েছে

object type (=) যে সবার উপর

object যেই টার যে প্রপার্টি call হবে

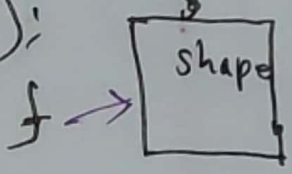
Ex:

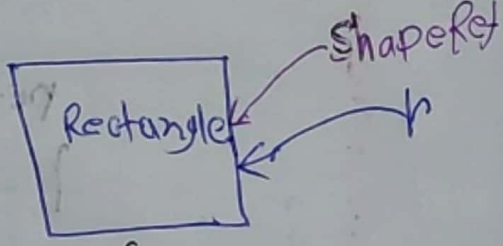
```
class Shape {
    double dim1;
    double dim2;

    Shape(double a, double b)
    {
        dim1 = a;
        dim1 = b;
    }
    double get_Area() {
        cout("no valid value to" + "the
            getArea of shape");
        return -1
    }
}
```

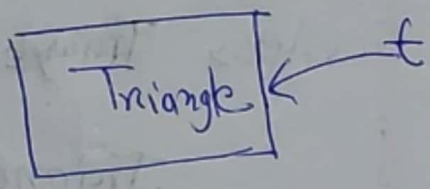
```
class Rectangle extend shape {
    Rectangle( double a, double b)
    {   super(a,b);
    }
```

```
class Rectangle extends Shape {
    Rectangle ( double a, double b)
    {
        super ( a, b);
    }

    double get_Area() {
    Sout ("Inside Area " + for Rectanle 2D");

    return dim1 * dim2;
    }
}

class Triangle extends Shape {
    Triangle ( double a, double b)
    {
        super (a,b);
    }

    double get_Area () {
        return dim1 * dim2 / 2
    }
}
```

```
public class Tester {
    publis static void main (String args []) {
        Shape f = new Shape (10, 10);
        Rectangle r = new Rectangle (9.5);
        Triangle t = new Triangle (10, 8);
        sout ( shaperef.getArea);
        sout ( shaperef. getArea)
```







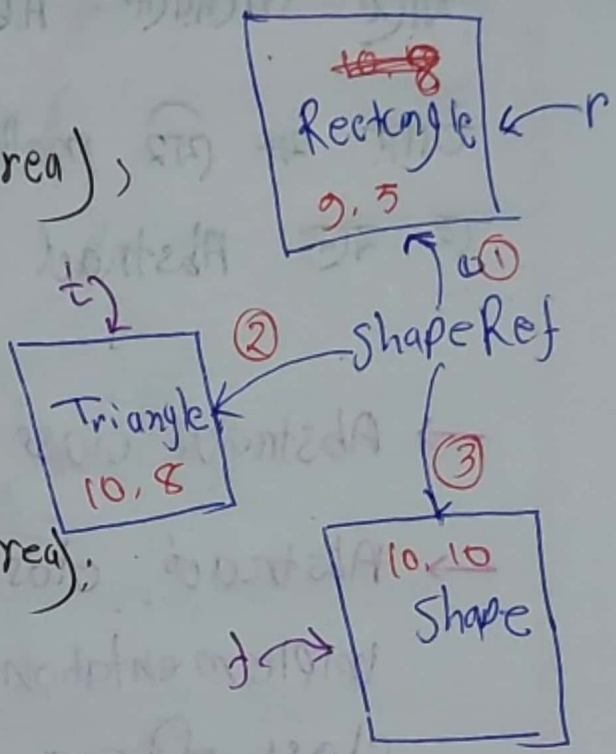
Shape shapeRef;

shapeRef = r

45.9// sout(shapeRef.get_Area);

shapeRef = t

40.0// Sout(shapeRef.get_Area);



~~1=11~~ shapeRef = f;

-1 // sout(shapeRef.get_Area);

and no valid ----

এখানে যেই টাইপের object কে refere করবে
সেই object বা class এর get_Area call হবে

super দ্বারা parent class এর dim1, dim2
কে access করছে

Abstract

যেই class এর অন্তত একটা unimplemented method থাকে তাকেই Abstract class বলে।

আর যেই method টা unimplemented সেই method কে বলে Abstract method.

→ Abstract class এর object তৈরি করা যায় না

→ Abstract class এর child class কে implementation ready করতে হয় না হয় child class নিজেও Abstract class হয়ে যাবে।

→ Abstract class এর method এর ();; braces এর সাথে সেমিকোলন দিয়ে দেওয়া হয়।

→ Abstract constructor বা Abstract static method তৈরি করা যায় না।

→ যে class এ Abstract এর code Run করাবে main

→ Abstract class থাকলে সেখানে যদি কোন আরো static method থাকে তাহলে call করা যাবে।

```
abstract class shape {

    double dim1;
    double dim2;

    shape( double a, double b)
    {
        dim1 = a;
        dim2 = b;
    }

    abstract double getArea();

}
```

Abstract class

Abstract method

```
abstract class Animal {

abstract void Sound();

}
```

```
abstract cat extends Animal {
    int x=10;
    void sound()
    {
        sout("cat class");
    }
    abstract void Show();

}
```

```
abstract Dog extends cat {

    void show()
    {
        sout("Dog class");
    }
    abstract void print();
```

```
public static void showX()
{
    sout(x);  sout(super.x);
}
}
```

```
public class Test {
    public static void main(String args[]) {

        Animal A = new Animal();
        Cat    c = new cat();
        Dog    d = new Dog();

        Dog.showX();
```

X
Abstract call করা যাবে না

static হবে Right

```
abstract class car {

abs: vad Show ();
}
```

```
abstract class Bike extends car {
        int x=10
        void Show () {
            sout ("car"); }

    abstract void print ();
}
```

```
class Bus extends Bike {

        void print () {
        sout ( this.x);
        }

    public static void Showx() {
            sop sout ( super.x) }
}
```

```
public class Test {
    public static void main (String args[]) {

        car c = new car();
        Bike b = new Bike();
```

} ✗ Abstract call করা যাবে না।

```
        Bus bus = new Bus();

        bus.print()
```

✗ this keyword not allow for access to parent class

```
        bus.show();
    or  Bus.show();
```

} Allow

```
        car c1 = new Bus();
        Bike b1 = new Bus();
```

} ✓

```
        Bike b2 = new car();
        Bus bus1 = new Bike();
```

} ✗

```
        Bus
```

Blocks

→ class এর object create করার সময়ে class এ execute করবে।

→ ~~আগে~~ আগে local Block গুলো execute করবে।

→ ~~Jina~~ Static Block object তৈরি করার সময় একবারই execute করবে পরে আর করবে না।

→ আগে static execute করবে পরে local (static > local) Block

→

```
public class Blocks {
    {                          // local Blocks
        sout(" Print 1);
    }

    void Show() {
        sout(" Show 1);
    }

    Static {                   // Static Blocks
        sout(" ~~Print 2~~ Static 1");
    }
```

```
static {
    sout (" static 1"); }

void show1 () {
        sout (" show 2"); }

{   sout (" print 3");
}
}

public class Test {
    psvm () {
        Book Blocks b1 = new Blocks ();
        // output
          static 1, static 2, print 1, print 2, print 3

        Blocks b2 = new Blocks ();
        // output
            print 1, print 2, print 3

        new Bloks ();
        // print 1, print 2, print 3
```

final keyword

→ Class final মানে ঐ class কে extends করা যাবে না।

→ method final করলে ঐ method কে override করা যাবে না child class এ

→ variable final করলে ঐ variable একবার initialize করার পর আর নতুন করে value set করা যাবে না।

```
final class A {
    void show()
    {
        sout("I am final class");
    }
}
```

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

× সম্ভব না

```
class A {
    final int a = 10;
    final void show() {
        sout (" final method"); }
}

class B extends A {
    super.a++; X
    void show() {
        sout (" override"); } X
}
```

চলবে না

Super And this keyword

→ super.x; parent class এর variable কে access করে

→ super.show(); parent class এর মেথোড কে access করে

→ this.a; দ্বারা নিজের class এর variable কে access করে,

→ this.show(); দ্বারা নিজের class এর method কে কল করা হয়

```
Class A {

    int a=10;

    void show() {
      Sout (" parent"); }

    Void show (int a) {
        Sout (" parent: "+a); }
}
```

```
class B extends A {
    int a = 20;
    void show () {
        sout (" child: " + this.a); }

    void show2 (int a) {
        sout (a); }

    void show2 () {
        this. show ();
        this. show2 (this.a);
        super. show (this.a);
        super. show (super.a);
        sout (super.a);
        sout (this.a);
    }
}
```

According → ob. show2 → Output

```
child: 20
20
parent: 20
parent: 10
10
20
```

type casting / object type casting

→ 1 Upcasting
2 Downcasting

→ Java Downcasting support করে না।

→ object type casting এমনি একটার class এর object
এ অন্য একটার class এর object রাখা।

→ upcasting by default হয়ে যায়

parent A = new child();

→ Downcasting child B = (child) new parent()

→ Runtime error
→ compile time error না

Inheritance relationship

① is-kind-Of, is_type-of, is-a

Ex: Manager is a type of Employe

Rectangle is a type of Shape2D
child parent

②

→ is-analogous-to

Ex: Rectangle and Triangle. They are same type.
analogous মানে তাদের মধ্যে super class আছে
যেমন: Rectangle ও Triangle এর মধ্যে একটা super
class আছে।

→ is-part-of

কোনো inheritance থাকবে না।

United International University (UIU)

Dept. of Computer Science & Engineering (CSE)

Midterm Exam, Trimester: Fall 2023

Course Code: CSE-1115, Course Title: Object Oriented Programming

Total Marks: 30, Duration: 1 Hour 45 Minutes

Any examinee found adopting unfair means will be expelled from the trimester / program as per UIU disciplinary rules.

Q1: (a) Logarithms are mathematical functions of the form $\log_b x$ which consists of two parts: base (int b) and argument (double x). The logarithmic value is calculated using the formula: [3+3]

$$\log_b x = \frac{\log_e x}{\log_e b}$$

where $\log_e x$ can be evaluated by Java code as Math.log(x).

You have to write a class **Logarithm** and include the **member** variables b and x, different types of **constructors** and a function **myfunc()** to evaluate $\log_b x$ such that the **Main** generates the output provided in the following table:

```
public class Main {
 public static void main(String[] args)
  {
    Logarithm log1 = new Logarithm(2, 9);
    Logarithm log2 = new Logarithm(log1);
    Logarithm log3 = new Logarithm();
    System.out.println(log1.b +"  "+log1.x+"  "+log1.myfunc());
    System.out.println(log2.b +"  "+log2.x+"  "+log2. myfunc());
    System.out.println(log3.b +"  "+log3.x+"  "+log3. myfunc());
  }
}
```

Output

```
2  9.0  3.0

2  9.0  3.0

0  0.0  0.0
```

(b) What is the output of following java codes:

```
public class Animal {
  public String color;
  public String name;
  public Animal() {
    System.out.println("Default animal");
    color = "Unknown";
  }
  public void showNameColor() {
    System.out.println("Color is: "+ color+" Name is: "+
name); }
  {
     System.out.println("Animal instance initialization ");
  }
}
```

```
public class Pokemon extends Animal
{
  public String name = "Pikachu";
  public String color = "Red";
}
public class AnimalTest {
  public static void main(String[] args) {
    Animal defaultAnimal = new
            Animal();
    Animal pk = new Pokemon();
    defaultAnimal.showNameColor();
    pk.showNameColor();
  }
}
```

Q2: (a) Consider the following codes [3+3]

```
public class BankAccount {
   public String name;
   private String account_id;
   private double balance;
   BankAccount(String name,double balance,
                  String gender){
      this.name=name;
      this.balance=balance;
      this.account_id=gender+"-"+name;
   }
   //Add necessary codes here
}
```

```
class Test{
   public static void main(String[] args) {
      BankAccount b=new
                     BankAccount("Mr.Rahman",1000, "M");
      System.out.println("Account id:"+b.account_id);
      System.out.println("balance before:"+b.balance);
      b.balance = b.balance - 2000;
   }
}
```

Rewrite the codes after correcting the errors by adding necessary getter/setter methods to access private variables. ***Do not change access modifiers*** of member variables. While updating the balance, a condition that ***balance cannot be less than 0*** should be considered.

(b) Consider the following codes:

```
public class PizzaShop {
   private int pizza_price=320;
   private int drink_price=40;
   private int fries_price=100;
   PizzaShop(){
      // Write necessary codes here
   }
 //Write necessary codes here
}
```

```
class Order{
   public static void main(String[] args) {
      PizzaShop p=new PizzaShop();
      p.order(2,4);
      p.order(1,2,1);
      p.order(3);
   }
}
```

Write the necessary **missing codes** so that the following output is produced when the program runs:

```
Welcome to pizza shop
You ordered 2 pizzas, 4 drinks
Total bill: 800 taka
You ordered 1 pizzas, 2 drinks, 1 fries
Total bill: 500 taka
You ordered 3 pizzas
Total bill: 960 taka
```

Q3: Consider the class named *Vehicle*. [1+1+2 +1+1]

```
class Vehicle {
   private String make;
   private String model;
   public Vehicle(String make, String model) {
      this.make = make;
      this.model = model;
   }
   public void start() {
      System.out.println("[Vehicle] The vehicle is starting.");
   }
```

```
    public void stop() {
        System.out.println("[Vehicle] The vehicle is stopping.");
    }
    public void drive() {
        System.out.println("[Vehicle] The vehicle is moving.");
    }
}
```

Create a class named *Car* that inherits the Vehicle class.

The Car class must have the following attributes/methods:

- An additional attribute named *numberOfDoors* (data type: int, access modifier: private).
- A *constructor* that receives the values of make (String), model (String), numberOfDoors (int) as arguments and initializes the attributes.
- A *method* that overrides the method *drive()* of the parent class. This method invokes *drive()* of the parent class first and then prints "[Car] The car is moving.".
- Another method named *honk()* (with access modifier: public and return type: void) that prints "[Car] Honk! Honk!"
- Now, consider the class named Main and write the output.

```
public class Main {
    public static void main(String[] args) {
        Vehicle car1 = new Car("make001", "model001", 4);
        car1.drive();
        car1.honk();
    }
}
```

Q4: Think about your own startup **Uthao** which is a ride sharing program using Java. Now due to our country's rules and regulations, vehicles on your platform must abide by the ***speedLimit*** at 80 km/h. [2+1.5+1.5+1]

(a) *Create a class* name **Ride** that contains an attribute ***speedLimit*** which

- Cannot get changed by its child classes
- Cannot get changed once assigned
- Will contain an Integer value
- Must be **static**

Now create the following child classes of **Ride:**

- Bike
- Car
- Microbus

Each child of Ride (that is, Bike, Car and Microbus) will receive a penalty for exceeding the ***speedLimit***. All the child classes have the following ***initial_speed*** and ***acceleration***:

- Bike: initial_speed 20 km/h , acceleration 2 km/h
- Car: initial_speed 40 km/h, acceleration 10 km/h
- Microbus: initial_speed 15 km/h, acceleration 5 km/h

Now do the following tasks:

(b) Create a method named ***getHighestAccelerationTime()*** (return type: double) which will find out the time needed for a ride to reach the ***speedLimit*** by using the formula : $v = u + at$

$t = \frac{v-u}{a}$

(c) Create a method named ***calculateFine(int hour)*** (return type: double) which will calculate the fine for each vehicle by following the implementation below:

- **Bike:** Base fine will be 50 TK and for each hour exceeding **speedLimit**, it will add an extra 100 TK
- **Car:** Base fine will be 100 TK and for each hour exceeding **speedLimit**, it will add an extra 150 TK
- **Microbus:** Total fine will be 3000 TK for just exceeding **speedLimit**.

(d) Find out the output of the following Code:

```
public class Uthao {
    public static void main(String[] args) {
        Ride car = new Car();
        System.out.println(car.calculateFine( hour: 10));
    }
}
```

Q5: (a) Can you define an abstract method in a non-abstract class? Provide a reason for your answer. [1+1+
(b) An abstract class cannot contain variables. Do you agree with the statement? Provide a reason for your answer. 2+1 +1]

Suppose that you are tasked with modeling a library system for various types of reading materials. You will be working with books, magazines. Each type of reading material has different properties. Thus, you are required to accomplish the following tasks:

(c) Write an **abstract class** named **ReadingMaterial** and
i. Add the following **private fields**: title (String), author (String), and year (int).
ii. Add a **constructor** that initializes these fields: (title: String, author: String, year: int).
iii. Define an **abstract method** named **displayDetails()** which should be implemented in subclass (es) to display the details of the reading material.

(d) Create a **subclass** named **Book** that inherits from **ReadingMaterial** and
i. Add an **additional private field** named genre (String) to hold the genre of the book.
ii. Add a **constructor** that takes all parameters (title, author, year, genre) and sets them properly.
iii. Now **override** the "**displayDetails()**" method to display the details of the book, including its title, author, year, and genre.

(e) Create a subclass named **"Magazine"** that inherits from **ReadingMaterial** and
i. Add an **additional private field** named issueNumber (int) to hold the issue number of the magazine.
ii. Add a **constructor** that takes all parameters (title, author, year, issueNumber) and sets them properly.
iii. Now **override** the **"displayDetails()"** method to display the details of the magazine, including its title, author, year, and issue number.