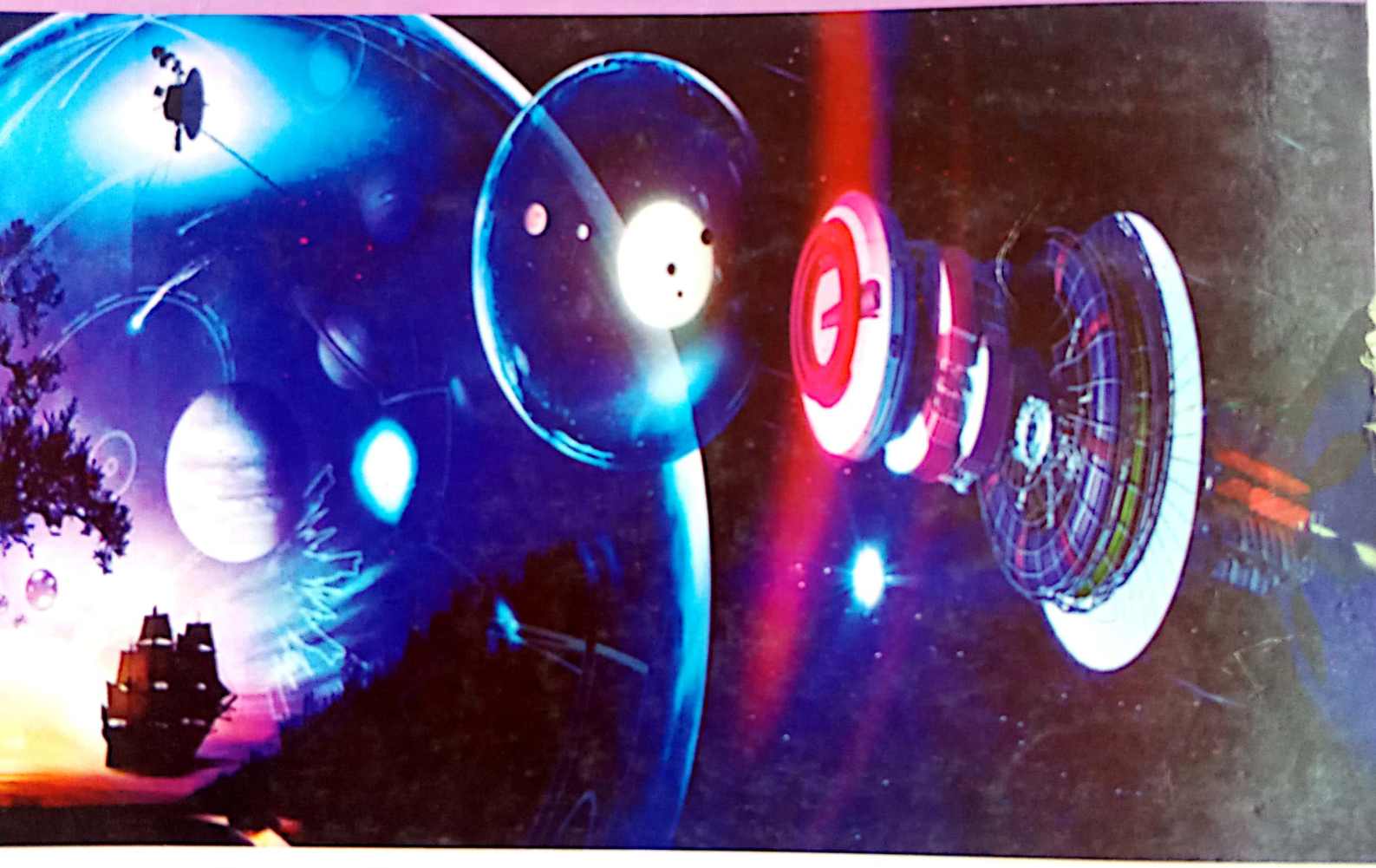
CSE 111

Capital

PROMEE

University

Students Writing Note Book



Education Support

Exercise Book

Quality First

Name :

Institute:

Subject :

Class: Roll. Year.

SAT SUN MON TUE WED THU FRI

Sub:

Page No :-

DATE: 13/05/18 TIME:

*number/alphabet/underscore(_)/dollarsign($)

* A name cannot be a keyword / it should start with alphabet/or underscore

* A variable name cannot start with a number

keyword – fundamental syntax

identifier – name

↳ literal

> 12345 (Java will understand – int literal)

>'c' (Java will und – character literal

There are five types of literals.

i) int literal (12345)

ii) floating literal (25.123)

iii) Character literal ('c') ('0')

single quote is only for single character. it will not work 'dog'

iv) string literal

v) boolean literal (true/false)

↓

(should be lower case)

>"dog"

⇒ error

>'0' + 1 ⇒ 49 (because of ascii code)

>"dog"

⇒ dog

>"0" + 1 ⇒ 01

>True ⇒ error

>true ⇒ true

Capital

There are 8 primitive types.

range
-127 - 128

basic

byte
short
int
long
} int literal

float
double
} floating literal

char
boolean

String => Box

String → An Array of Chars

Capital

int x → type and creates a box

int[] y

key word → new type int [3]

{0, 0, 0}

int[]y = new int [3];
System.out.println (y);

it will show the address

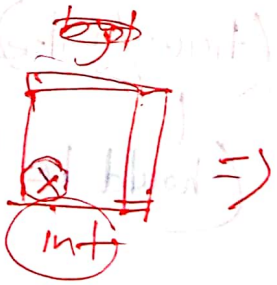
int [] z;

z = y

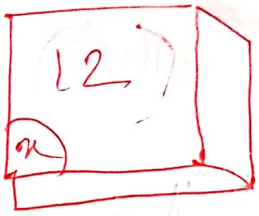
z and y have now same contents

'0' + 1

= 49



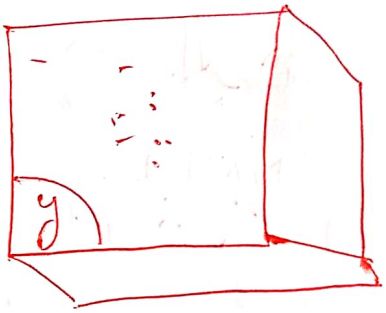
int x;



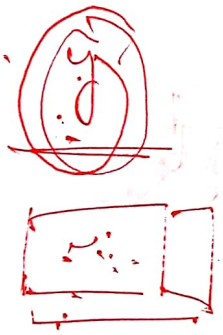
int [] y;

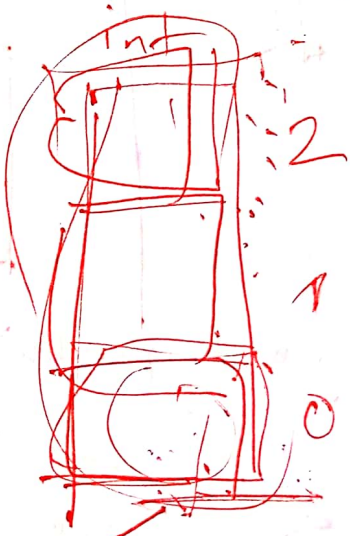
int[]

y = new int[3]

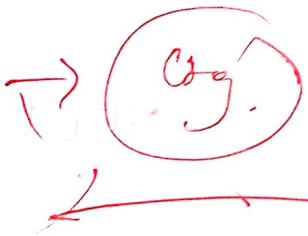


Memory





y is the address of the first index of an array.



Capital

SAT SUN MON TUE WED THU FRI

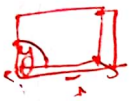
Sub:

DATE: / / TIME:

Page No :-

~~int [] y;~~ ⟹ ~~address~~

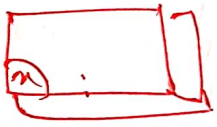
int [] y; ⟹ (address)



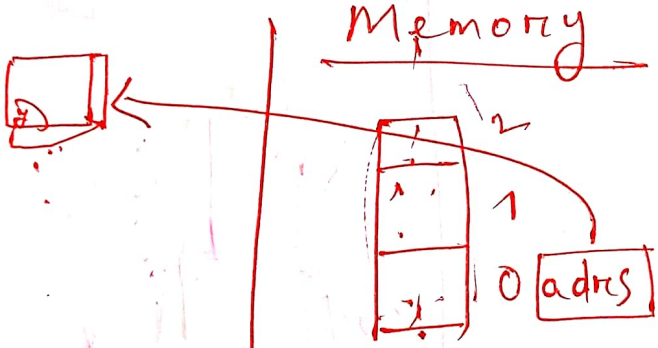
⟹ address

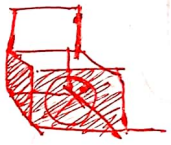
NULL

int x;



y = new int [3];





y [0] = 0

y [1] = 0

y [2] = 0

System.out.println(y[1]); (0-2)

NULL ← java.lang.NullPointException

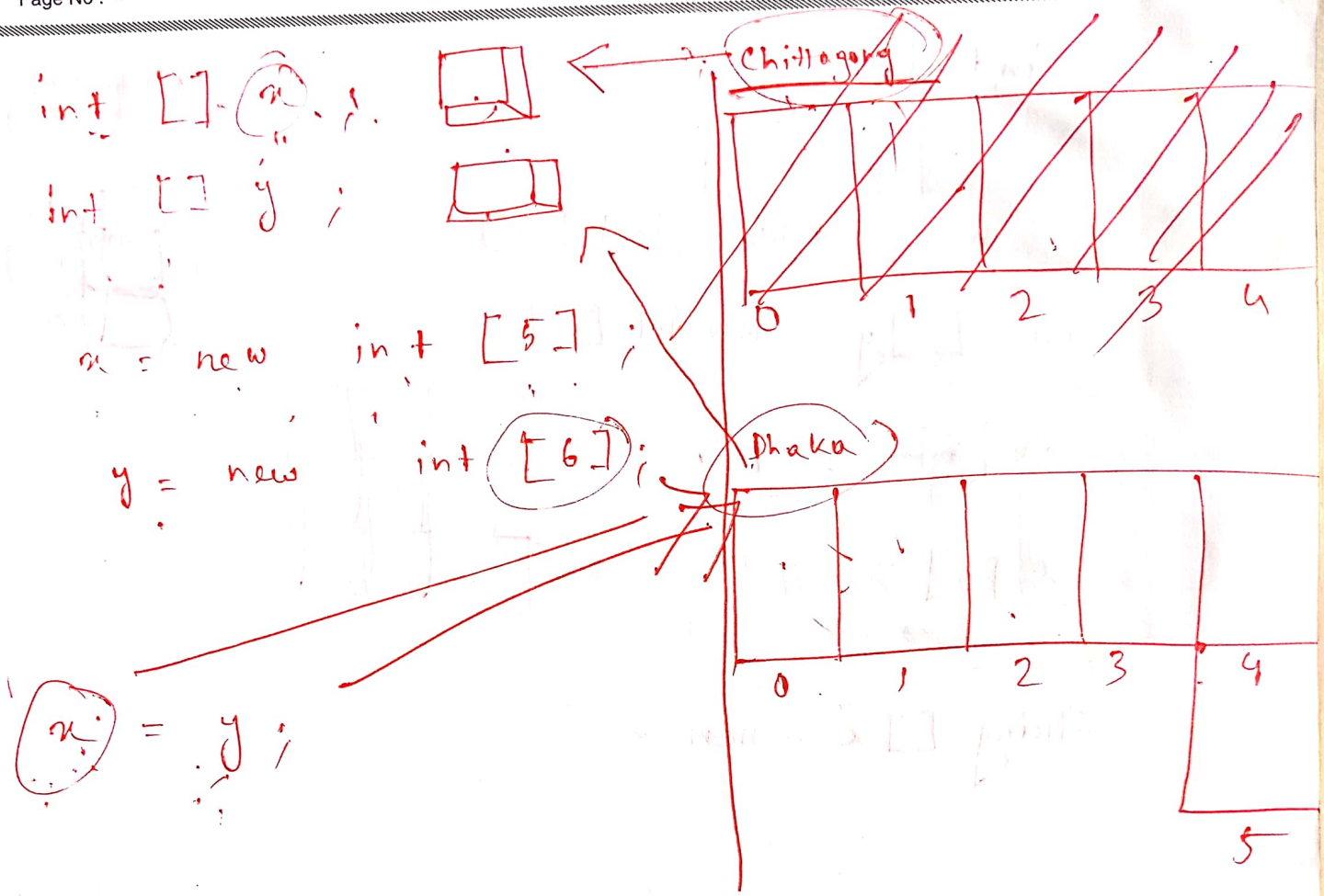
~~y[3]~~ java.lang.indexOutOfBound [3] =

SAT SUN MON TUE WED THU FRI

Sub:

DATE: / / TIME:

Page No :-



promee = ~~Bushabo~~ Bushe

Ba'Samin = Baskunthum

promee = Samin

Capital

SAT SUN MON TUE WED THU FRI

Sub:

Page No :-

DATE: 20/05/18 TIME:

Class and Object:

A class is a design

A object is made from the design.

object ~~is combination of~~ has some stuff and it can do stuff.

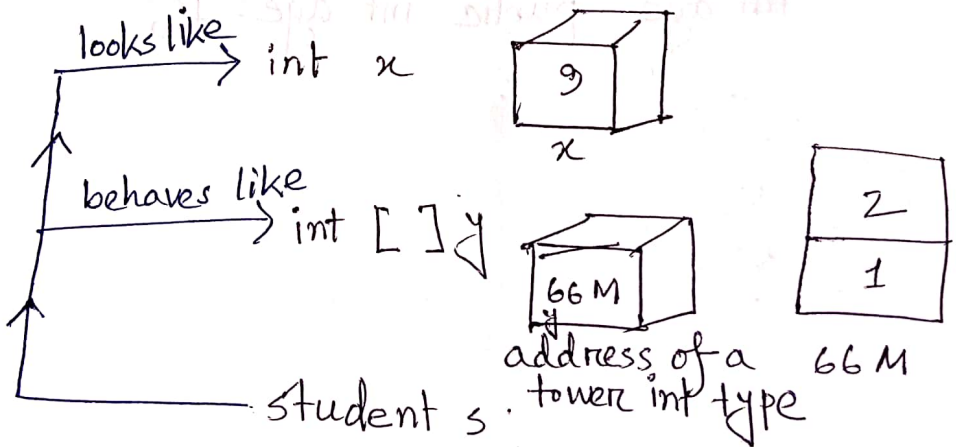
↓ properties are the stuff object has

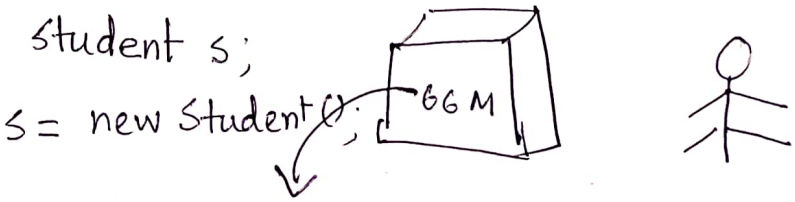
↓ method

Capital

User defined type + 8 types of primitive data ty

```
public class Student{
                        //This will compile but won't run
}

public class Prog {
  public static void main(String[] args){
  }
                        //will run
}

public class Prog {
  public static void main(String[] args){
    int x;
    x = 9
    int y;
    y[0] = 1
    y[1] = 2
  }
}
```

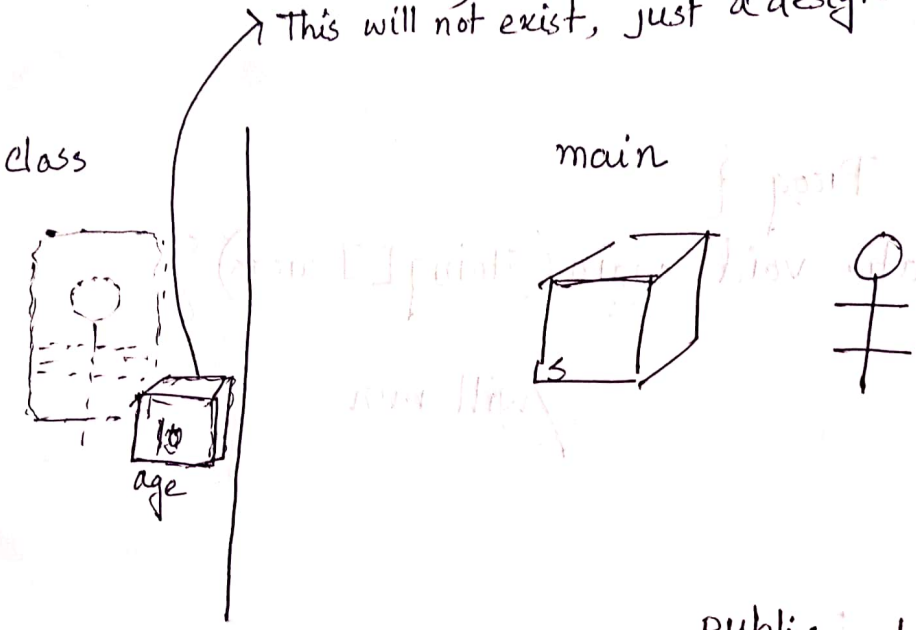




Address of a object
student type

Student s;

s = new Student ();




```
public class Student {
    ~~int age~~ public int age = 10;
```

```
public class Prog {
    public static void main (String[] args) {
        Student s = new Student ();
        System.out.println (s.age);
        s.age = 300;
        Student k;
        k = new Student;
        SOP (k.age)
    }
}
```

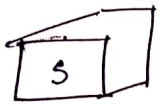
10
10

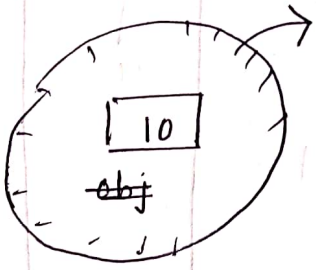
SAT SUN MON TUE WED THU FRI

Sub:

DATE: 29 / 5 / 18 TIME:

Page No :-





What is inside can be seen outside but what is ouside can not be ~~at~~ seen outsides

```
public class Student {
  public String name = "Bob";
```

(0 --> P standUp P-->0)

void standUp ()

h1 / h2		h2 ☆	
age.	height	age	height
~~11~~	~~10·11~~	12	7·11
~~12~~	~~7·11~~		
13	8·11		

output

11
10·11
1211
10·11
~~7·1311~~
7·11
1411
~~10 8·11~~ 10·11
1511
9·11
15
8·11

11+1 = 12

1211
10·11
1311
7·11
14
1411
8·11

Capital

SAT SUN MON TUE WED THU FRI

Sub:

DATE: / / TIME:

Page No :-

4 core concepts

① Encapsulation
② Inheritance
③ Polymorphism
④ Abstraction

Access modifier

① private
② protected
③ public

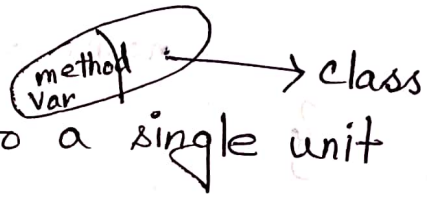
} keywords

(default)

Encapsulation:

a process of

① packaging var and methods into a single unit



② protecting data by declaring them as private

<private data will be hidden from other classes and they can only be accessed through the methods of their current class. This is known as data hiding.>

Capital

setName / getName

private করা variable বাইরের ক্লাস থেকে access করা যায় না, এদের change করতে set/get methods ব্যবহার করা হয়,

```
public class Person{
    private String name;
    private int age;
    public String getName(){
        return name;
    }
    public void setName(String Name){
        this.name = name;
    }
```

SAT SUN MON TUE WED THU FRI

Sub:

DATE: / / TIME:

Page No :-

encapsulation (Binding data with methods)

declare the variables as private

provide public setter and getter method to modify and get the variable's value.

Setter & Getter Methods

```
public class {
    private String name;
    private int age;

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

    public String getName(){
        return name;
    }

    public void setAge(~~Stri~~int age){
        this.age = age;
    }

    public int getAge(){
        return age;
    }
}
```

```
public class EncapTest{
    public static void main(String[]args){
        Person p1 = new Person();
        P1.setName("Anis");
        SOP(P1.getName());
    }
}
```

Sub:

Page No :-

DATE: / / TIME:

Inheritance: (mainly transfers info from one class to another)

যাকে inherit করা হয় parent/super/base class

```
public class Person{
    String name;
    int age;
    void displayInformation(){
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
```

if we want to use all the properties and methods of class Person in another class?

keyword for inheritance

যে inherit করে child/sub/derived class

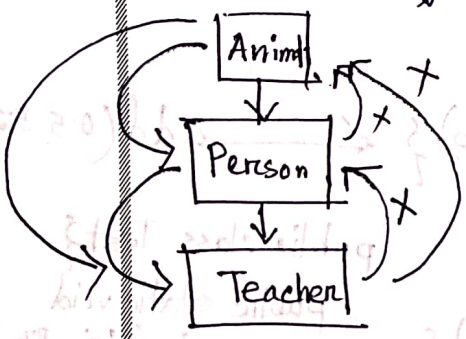
```
public class Teacher extends Person{
    String qualification;
    void displayInformation2(){
        displayinformation();
        System.out.println("Qualification: " + qualification);
    }
}
```

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another.

```
public class test {
    public static void main(String[] args) {
        Teacher t1 = new Teacher();
        t1.name = "Alak Kanti"
        t1.age = 30
        t1.qualification = "CSE";
        t1.displayinformation();
    }
}
```

Capital

Instance of Operator




```
Animal a = new Animal();
Person p = new Person();
Teacher t = new Person();
```

Output

```
SOP (a instanceof Animal) True
SOP (p instanceof Person) True
SOP (p instanceof Teacher) False
```

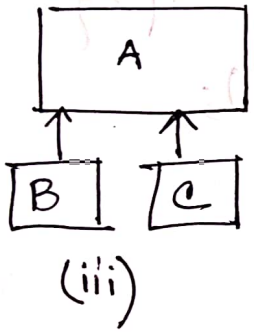
Sub:

Page No :-

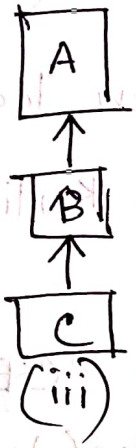
DATE: / / TIME:

Types of Inheritance

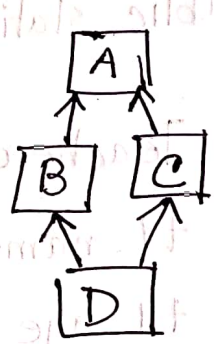
4 types
i) single Inheritance (A subclass will inherit another superclass)
ii) Multilevel Inheritance (Many subclass will herit many superclass)
iii) Hierarchical Inhertice
iv) Multiple Inheritance



(iii)



(iii)



(iv) is not supported by java

Method Overloading

```
public class Overload {
    void add (int a, int b){          <-- add(0,5)
        SOP (a+b);
    }
    void add (double a, double b){    <-- add(0.5, 5.2)
        SOP (a+b);
    }
    void add (int a, int b, int c){
        SOP (a+b);
    }
    void add () {
        SOP ("Nothing to Add");
    }
```

```
public class Test{
    public static void
        main(String[]args)
    {
        Overload ob = new
            Overload();
        ob.add();
        ob.add(5);
```

#Method overloading is a process that allows a class to have two or more methods with same name, as long as their parameter declarations are different.

These are called overloaded Methods

<same method names>
<different parameter>
<same class>

Method Overriding

```
public class Person{
    String name;
    int age;
    void displayinformation(){
        SOP ("Name : " + name);
        SOP ("age: " + age);
    }
}
```

```
public class Teacher extends
                Person{
    String qualification;
    @ override
    void displayinformation(){
        SOP ("Name: " +name);
        SOP ("age: " +age);
        SOP ("qualification" + qu);
    }
}
```

Declaring a method in subclass which is already present in superclass is known as Method Overriding

<Name, signature, parameter same>

Constructor

⇒ It should have the same name of the class.

⇒ It should be called once.

static (solid inside the design)

↓

created when the design is compiled

static methods can only access static variables

SAT ☑ SUN ☐ MON ☐ TUE ☐ WED ☐ THU ☐ FRI ☐

Sub: CSE 111 Lab (8)

DATE: 7 / 7 / 18 TIME:

Page No :-

```
interface Bicycle {

        double brake (double a);
        void speed (double b);
        int padle (int c);

}
```

সব use করতে হবে

Class এ সব implement না করলেও চলে তবে interface এ সব implement করতে হবে।

```
class mybicycle implements Bicycle {

        brake (...) {

        }

          speed (...) {

        }

        padle(....) {

    }

}
```

Capital

Sub: Inheritence (Super/Final)

Page No :-

SAT SUN MON TUE WED THU FRI

DATE: 8 / 7 / 18 TIME:

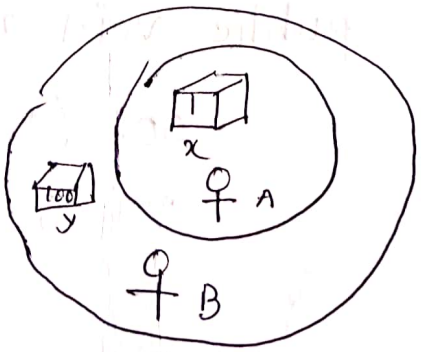
```
public class A{
    int x = 1;
  public void mA(){
     SOP("in method A" + x);
  }
public class B extends A {
    int y = 100;
  public void mB(){
     SOP(x)
     SOP("in method B" + y);
  }
```

```
bB.mB();
```

output

```
1
in method B 100
```

when a object of B is created A will be created first



Sub:

Page No :-

SAT SUN MON TUE WED THU FR

DATE: / / TIME:

over-ride

```
class A {
    x = 1;
    m();
}
class B {
    x = 2
}
B b = new B();
b.x;
```

for refering parent class

super

recursion:
is like loop

2

b

```
public class B extends A {
    public int x = 100;
    public void m() {
        sop("in method B" + x);
        m();
    }
```

recursion

*** when java goes to parent class it will call the default cons

```
public class B extends A {

    public B (){
        super (2);
        SOP ("OLD");
    }

    public B (int x){
        super (x);
        SOP ("NEW");
    }
}
```

this → for same class

final → works like constant → all caps

final method can't be over-ridden

final class can't be extended

SAT SUN MON TUE WED THU FRI

Sub: Inheritence (toString()/MotherClass Object) DATE: 10 / 7 / 18 TIME:

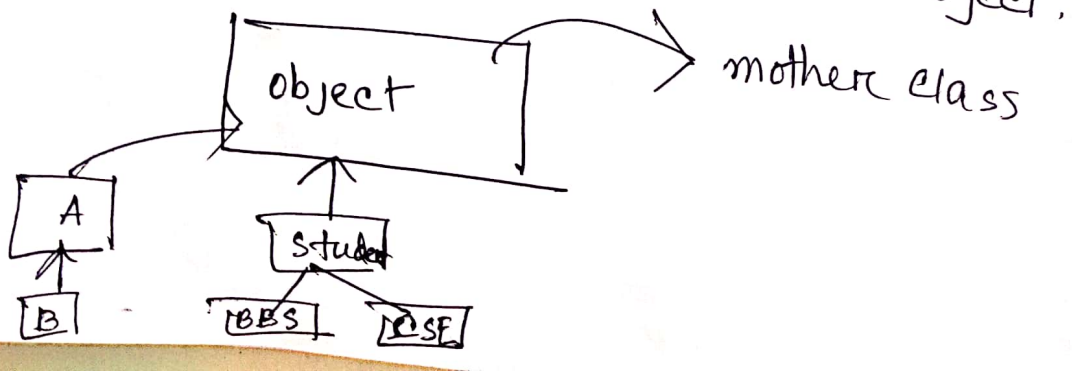
Page No :-

```
public class student{

}

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

    student s = new student();
    s.toString();
    SOP (s);

  }

}
```

output

student@1c7d291

if we don't extend from any class it automacilly extends from a mother class. The class is called object.



Sub: Inheritenc

Page No :- → object class

DATE: / / TIME:

SAT SUN MON TUE WED THU FRI

toString() (default method of object)

getClass().getName() + @ + Integer.toHexString(hashCode(

a string representation of object

If tostring() is written by the designer it will show that value. Then to call toString() of object we have to write super.toString().

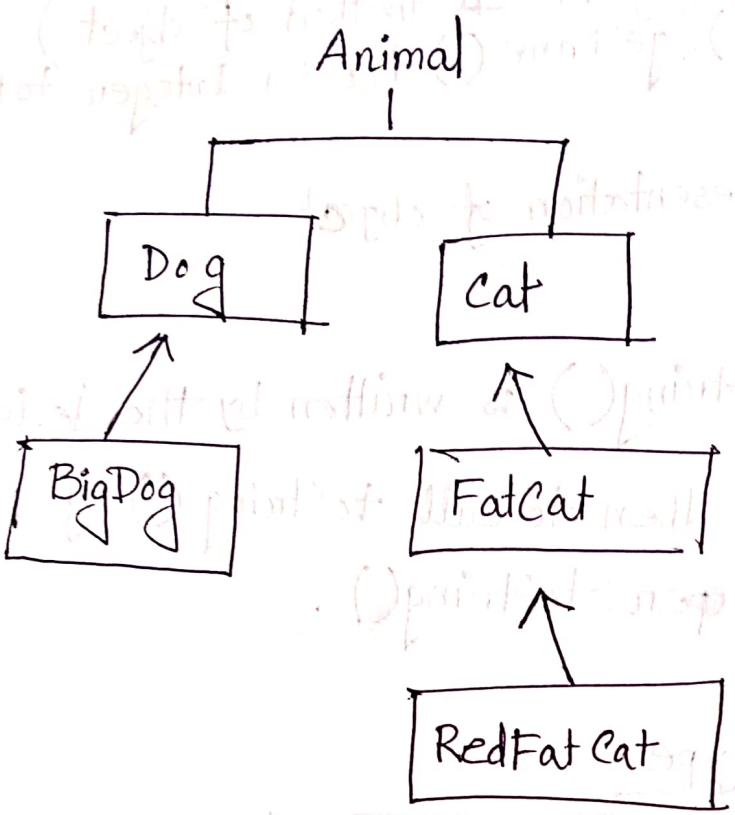
reference type

```
Tree k = new Tree();
```

→ reffered object.

```
Animal t = new Dog();
```

A reference of super class can refer to an object of sub class




```
Animal a = new Animal();
```

```
Animal  a = new Dog();
"       a = new BigDog();
"       a = new Cat()
```

← super class holding the reference of sub class

```
Animal a = null;
d1 = new Dog();
SOP(d1);
a = d1;
SOP(a);
```

`d2 = a` → It will not compile

```
if (a instanceof Dog {
    d2 = (Dog) a;
}
```
→ compiles

`d2 = (dog) a;` → It will compile

output

Dog@1....

Dog@.....

a sub class object is bigger than a super class object

```
int i = 2;
byte b = 1;
i = b;
```

`b = i` → Static error (byte is shorter than it)

```
b = (byte) i
i = 1000;
b = (byte) i;
```

It's called explicit casting

```
public class Animal {
    public String sound = "Animal Sound";
    public void makeSound () {
        SOP (sound);
    }
}
```

```
public class Dogs {
    sound = Bark
    {
    SOP (sound)
}
```

Animal can hold (animal and dog)

* B create a object of super class and pass the refer of sub class

```
public class Prog {
    public static void showSound (Animal k) {
        SOP (k.sound)
    }

    ps v m (String[] args) {
    Animal a = new Animal ();
    SOP (a.sound);
    showSound (a);
    }
```

4. output
Animal sound

```
***
    Dog d = new Dog();
    showSound(d);
}

public static void showSound(Animal k){
    SOP(k.sound);
}

"Animal Sound"

public static void showSound(Animal k){
    if(k instanceof Dog){
        Dog t = (Dog k)
        t.makeSound();
    }
}

psvm {
    Dog d = new Dog();
    showSound(d);
}
```

"Bark"

```
psvm (String [] args)
Dog d = new Dog();
showSound (d);
Cat c = new Cat();
showsound (c);
}
```

```
ps void showSound (Animal k)
  k.makeSound()
}
```

Bark
Meow
~~Mooo~~

```
{SOP (k.sound);
 k.makesound();
}
```

Bark
Animal sound
Meow
Animal sound

Sub: Abstract

Page No :-

DATE: 29 / 6 / 18 TIME:

*abstract method cannot have a body. (body means {})

{

}

any sub class cannot be compiled if it does not overri
de the method

an abstract class can have [abstract
concreate

an ~~animal~~ abstract class cannot have any object. Nev
cannot ~~hav~~ be written for abstract.

public abstract class Animal{

}

Animal a = new Animal(); X not possible

Sub:

Page No :

```
public Abstract class Animal {
    public abstract void makesound();
}

public abstract class Dog e Animal {
    public abstract void wagtail();
}

public class RedDog e Dog {
    public void make sound();
    public void wagtail();
}
```

concrete class cannot hold abstract;

```
public interface soundCon {
    public void makesound();
}
```

→ no body do same work