Object Oriented Programming

CSE1205 → JAVA

Lecture-1 | Date - 6/01/2019

* Course Credit: 3
* Course Teacher: Dr Md. Shahriar Mahbub
* Ref. Book: The complete reference: Java by H. Schildt
* Additional book: Head first: Java, Kathy Sierra & Bert Bates.

* Platform independent: Java, C#

* 1. Source Code (test.java)

↓ compile (Windows/Linux/Unix)

2. Byte code (test.class) → half-compiled

↓ JVM / JRE → (Java Runtime Environment)
↳ (Java Virtual Machine)

run

* JDk → Java development key.

Lecture-2 | Date- 8/1/19

* Three principles of object-oriented programming (oop)

1. Encapsulation : binds code (process + data) and data (variable)
2. Inheritence : (अन्य class से variable लिया जा सकता है)
3. Polymorphism : One interface, multiple methods

Lecture-3 | Date- 9/1/19

* Class & Object:

Class:

Object:

* className: Class Class Class ... : [Ex.- MyClass.]
* Objectname: aaaa Class Class ... : [Ex.- myObject]
* class_className{

```
type variable1;
:
type variable N;

returntype method1(...){

}
returntype methodN(...){

}
}
```

data

code

Encapsulation

* class Box{

```
double height;
double width;
double depth;
```

}

* Object:

```
Box matchBox = new Box();
```

- Box → classname
- matchBox → object name
- new → (keyword to allocate memory)

* class BoxDemo{ → ~~আলাদাভাবে~~ Define করা হলে/হবে।

* public class BoxDemo{

Filename → BoxDemo.java

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

    Box myBox = new Box();
    myBox.height = 5;
    myBox.width = 3;
    myBox.depth = 4;
    double volume = mybox.height * mybox.width * mybox.
                                                depth;
    System.out.println("volume:" + volume);
}
```

}

Lecture-4 (13-1-19)

* (i)

```
class Box{
    double height, width, depth;
    double calculateVolume(){

    }
}
```

(ii)

```
class BoxDemo{
    public static main( ){
        Box myBox = new Box();
                    → constructor call
        myBox.height=(3);
        myBox.width=(2);
        myBox.depth=(1);

        System ("height of myBox"
                + myBox.getHeight);

        System (" Volume "+myBox.calculate
                Volume() );
    }
}
```

```
class Box {

    double height, width, depth;
    double calculateVolume() {

    }

    void setHeight(double h) {
        if (h > 0) {
            height = h;
        }
    }

    void setWidth(double w) {
        if (w > 0) {
            width = w;
        }
    }

    void setDepth(double d) {
        if (d > 0) {
            depth = d;
        }
    }

    double getHeight() {
        return height;
    }
```

(setmethod)

```
Box(double h, double w, double d) {
    height = h;
    width = w;
    depth = d;
}
```

```
    double getWidth(){
        return width;
    }
    double getDepth(){
        return depth;
    }
}
```

* constructor:

special method। class এর নাম এর সাথে exactly same হবে। return করে না। object create করার সময় automatically call হয়।

Lecture-5 (15-1-19)

*

```
public class Box {

    private double height, width, depth;

    public Box() {                          // Parameterized constructor
        height = h;
        width = w;
        depth = d;
    }

    double calculateVolume() {
        return height * width * depth;
    }

    double getHeight() {
        return height;
    }

    public Box(double l) {                  // method overloading
        height = l;
        width = l;
        depth = l;
    }
```

```
public Box () {
        hight = width = depth = 0;
    }
void setWidth (double w) {
        width = w;
    }

class BoxDemo {

    public static ..... main (... ) {

        Box myBox = new Box (5, 3, 1);
        Box matchBox = new Box (10, 7, 9);

        System.out.println( myBox.calculateVolume ());
        System.out.println (matchBox.calculateVolume ());
        System.out.println ("height of my Box" + myBox.height);

        Box cube = new Box (5);
        myBox.setWidth (5);

    }

}
```

* overloading :

number of parameter or data type

* class test {

public double Area (double a, double b){

return 3.1416 * a * b;

}

public double Area (double r){

return 3.1416 * r * r;

}

}

Lecture - 6 (16-1-19)

* Object as parameter:

```
public class Box{

    double height, depth, width;

    public Box(double h, double w, double d){

        setHeight(h);
        setWidth(w);
        setDepth(d);

    }

    public boolean testEqualDimension(Box ob){

        if( height == ob.height && width == ob.width
            && depth == ob.depth)

            return true;

        else

            return false;

    }

}

}
```

```
class BoxDemo{

    public static void main(){

        Box myBox1 = new Box(3,2,1);

        Box myBox2 = new Box(5,3,1);

        Box myBox3 = new Box(3,2,1);

        System.out.println(" is myBox1 equal to myBox2"
                + myBox1.testEqualDimension(myBox2));

    }

}
```

Lecture - 7 (20-1-19)

Quiz - 1 (27-1-19)
Time - class time
Syllabus - This chapter.
(upto lecture - 7)

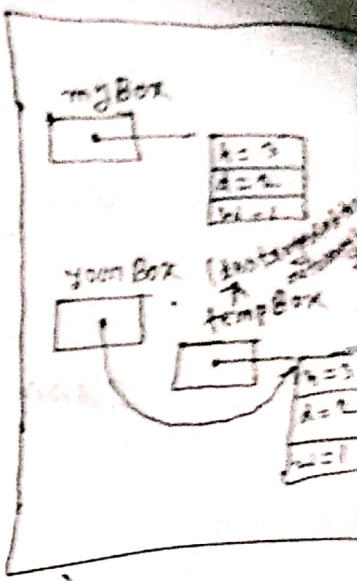
```
class Box{
    double height, depth, width;        // instance variable
    Box(double height, double depth, double width){   // local variable.
        this.height = height;
        this.depth = depth;
        this.width = width;
    }
    Box makeAEqualBox(){
        Box tempBox = new Box();
        //tempBox.height = height;
        //tempBox.depth = depth;
        //tempBox.width = width;
        return tempBox;
    }
}
```

this. refers to current object

return type: Box

(height, depth, width)

```
class BoxDemo{
    public static... main(...){
        Box myBox= new Box(3,2,1);
        Box yourBox;
        yourBox= myBox.makeAEqualBox();
    }
}
```



(১ম পরীক্ষা 1st Quiz)

* does it make sense to call a method, even if no object has been constructed yet?

Lecture- 8 (22-1-19) | static method কে class এর নাম দিয়ে কল করতে হয়

```
class Box{
    double hight, width, depth;
    Box(){
    }
    double calculateVolume(){
        return hight * width * depth;
    }
    static double convertInchToCm(double inchValue){
        return 2.54 * inchValue;
    }
}
```

(যে প্রোগ্রামে Box এর কোন object তৈরি না করে কোন কাজ করতে হবে) ← static

```
class BoxDemo{
    public static void main(String[] args){
        System.out.println(Box.(class name) convertInchToCm(3.2));
    }
}
```

* Static have several restrictions:

1] They can only call other static method.

2] They must access static data/variable.

3] They can not refer to this in any way.

** Why main method is public & static?

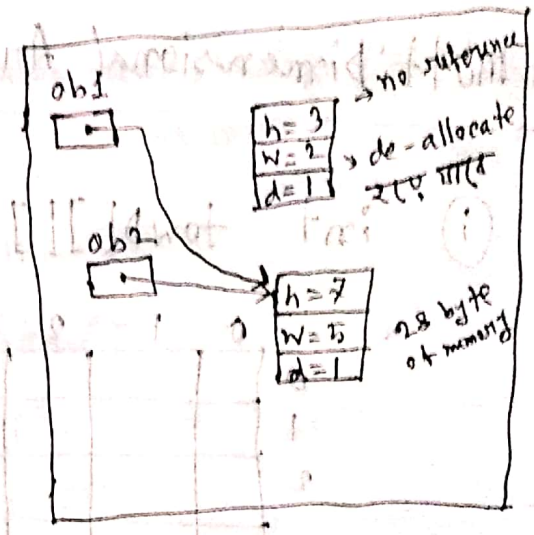
⇒ private হলে Java virtual machine method কে access করতে পারত না, ফলে program run হত না।
যাতে JVM কোন object তৈরি না করেই main method কে call করতে

আজ, আর static.

Lecture - 9 (23-1-19)

* Garbage Collection:

```
Box ob1 = new Box(3,2,1);
Box ob2 = new Box(7,5,1);
ob1 = ob2;
```



* Array:

(i) `int[] intArray = new int[10];`

(ii) `int intArray[] = new int[10];`

(iii)
```
int intArray[];
intArray = new int[10];
```

Initialization:

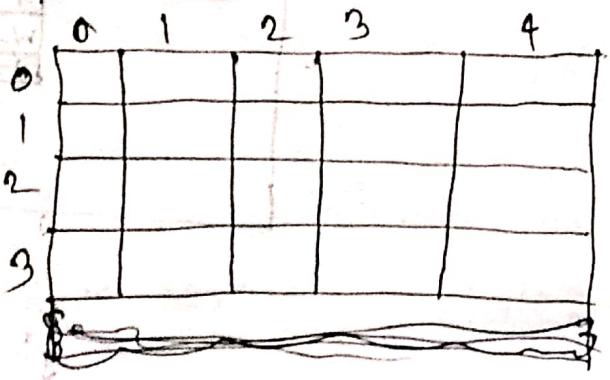
```
int intArray[] = {1,2,3,4,5};
```

Printing:

```
for(i=0; i<intArray.length; i++)
{
    System.out.println(intArray[i]);
}
```

* MultiDimensional Array:

(i) `int twoD[][] = new int[4][5];`



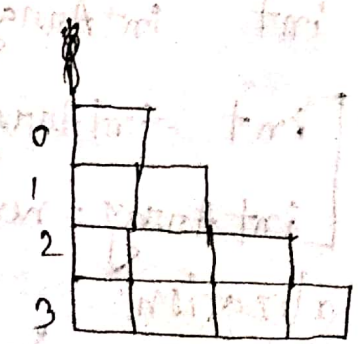
* `int twoD = new int[4][];`

`twoD[0] = new int[1];`

`twoD[1] = new int[2];`

`twoD[2] = new int[3];`

`twoD[3] = new int[4];` ⇒



* Class Student{

```
int roll;
String name, dept, telNo;
Student(){
    roll = 0;
    name = dept = telNo = null;
}
Student(int roll, String name, String dept, String
        telNo){
    this.roll = roll;
    this.name = name;
    this.dept = dept;
    this.telNo = telNo;
}
```

```
Student [] studentArray = new Student[50];

studentArray[0] = new Student(1, "x", "CSE", "017...");
```

Lecture-10 (30-1-19)

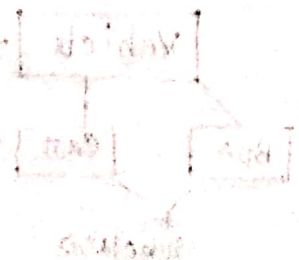
* Nested class/ Inner class:

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

→ inner class

```
* class Student{
    class ID{
        int year, sessionCode, deptCode, serial;
        ID(int y, int sc, int dc, int s)
        {
            year = y;
            sessionCode = sc;
            deptCode = dc;
            serial = s;
            if(s>=1 && s<=50)
                section = "A";
```

```
        else
            section = "B";
    }
}
String name, section;
ID id;
Student(String name, int y, int se, int de, int s)
{
    this.name = name;
    this.id = new ID(y, se, de, s);
}
}

Class Demo{
    public static void main(String[] args){
        Student std1 = new Student("Rakib", 18, 01, 04, 032);
}
```

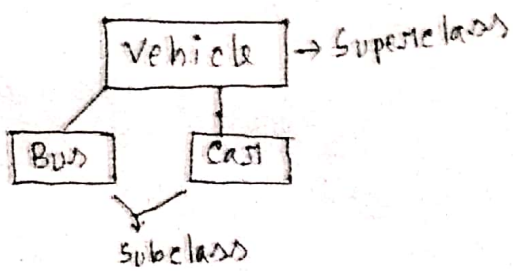


Lecture - 11 (3-2-18)

* Inheritance:

```
Class Vehicle {

        void move( . . . ){
            . . . . .
        }

}
```

* Constructor and set method & get method class এ essential part.



```
class Car extends Vehicle{        // keyword: extends
```

we can use void move() wre, without writing again

```
}
```

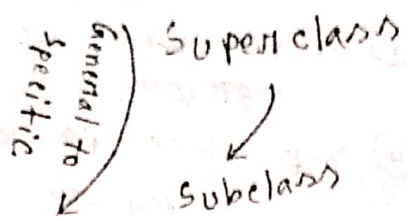
```
* class A {
      int i, j;
      void showij(){
          System.out.println("i "+i +"j "+j);
      }
  }

class B extends A{
      int k;
      void showAll() {
          showij();
          System.println("k "+ k);
      }

}
```

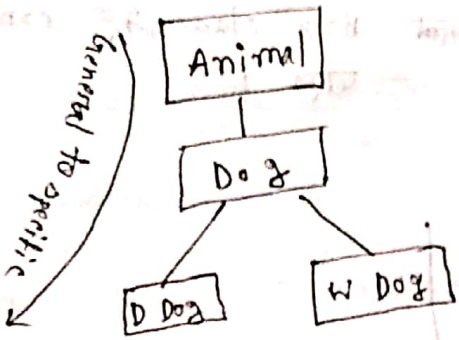
Scanned by CamScanner

```
class Demo{
    public static void main(String args[]){
        A obA = new A();
        obA.i = 3;
        obA.j = 4;
        obA.k = 7;  → ভুল → (A class এ k নামে কোন member নেই, তাই Access করতে পারছে না।)
        obA.showij();

        B obB = new B();
        obB.i = 13;
        obB.j = 15;
        obB.k = 17;
        obB.showAll();
    }
}
```

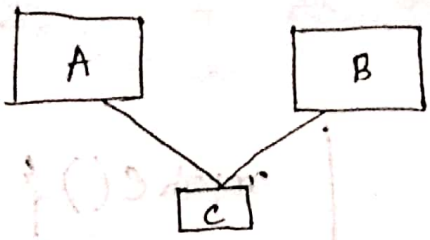
* Private variable কে subclass থেকে Access করা যায় না। Set method দিয়ে Access করা যায়।

Lecture-12(5-2-19)





*



← not allowed (multiple inheritance)

* 1টা subclass এর 2টা বা তার বেশি superclass থাকতে পারে না।

* 1টা superclass এর অসংখ্য subclass থাকতে পারে।

* 'super' keyword :

super ... immediate super class ...

... super ( ) ;

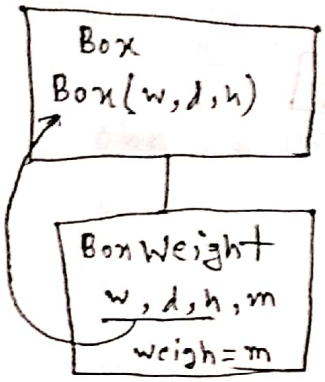
① super class ... constructor ... call ...

② super class ... member ... call ...

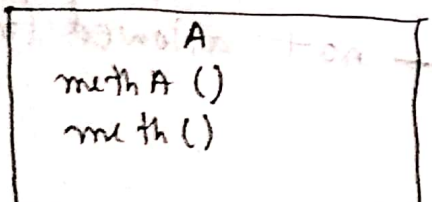
super (w, h, d) ;

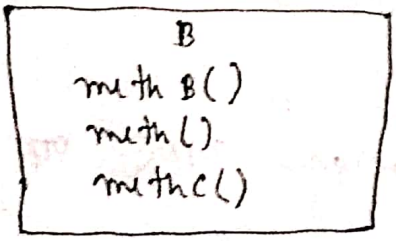
↳ ... Box class ... constructor ...

*



*






```
B ob = new B()
ob.methB();
ob.methA();
```

```
methC(){
    methA();
    methB();
    meth();  → B class ...
    super.meth(); → A ...
}
```

Vidyut Vidyut

↳ Google classroom

Lecture-13 (6-9-19)

* Sub class এর construction এ যখন super class এর construction call করা হয়, তখন super class এর constructor call করতে হবে, এবং subclass এর variable কে assign করতে হবে।

* type mismatch:

- super class এর object এর জন্য subclass এর object কে assign করা possible.

  It is possible to assign an object of subclass to an object of superclass.

```
class A {
        int i;
}
class B extends A {
        int j;
}
class Demo {
        public . . - ( . . ) {
                A obA = new B();
```

obA.i = 3;

obA.j = 4; → এভাবে assign করা যায়।

B obB = new B();

obB.i = 1;

obB.j = 10;

}

* memory representation কেমন হবে?

Lecture-19 (12-2-19)

* Method Overridding:

ভিন্ন ক্লাসে same নামের method থাকে sub-class, super-class কে Inherit করলে।

```
Class A {
    void meth (int a) {
    }
}

Class B extends A {
    void meth (int a) {
    }
}
```

Lecture-15 (13-02-19)

* Quiz-2 (19-2-19)
  Syllabus: inner class + static + inheritance (upto this week)

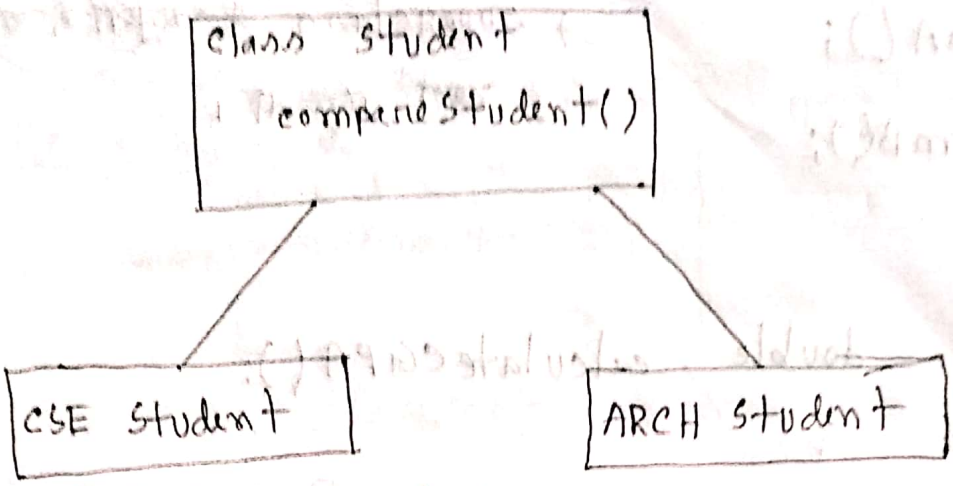
* Dynamic Method Dispatch (calling):

  * By which a call to an overridden function is resolve at run time, rather than compile time.
  * Using this, java implements run time polymorphism.
  * A superclass reference variable can refer to a subclass object.
  * When an overridden method is called through a superclass reference
    - Java determines which version of that method to execute based upon the type of the object being referred to at the time the call occurs.

→ Static method dispatch ⇒ compile time.

→ Dynamic " " ⇒ run time.

Lecture-16 (17-02-19)




```
* Class A {
        int i;
        void mA(){
        }
  }
  class B extends A{
        int j;
        void mB(){
        }
  }
  class Main{ main(...){
        A r;
        B b = new B();
        r = b;
        r.i = 3;
```

✗ ম.j = 2;

✓ ম.mA();

✗ ম.mB();

→ superclass এ অনুপস্থিত। তাই subclass থেকে call করা যাবে না।

* abstract double calculateCGPA();

↓

এই method এর body লিখা হবে না।

* যে ক্লাসে abstract method থাকবে সে class কেও abstract হিসেবে declare করতে হবে।

abstract class Student{

}

* abstract class এর কোন object create করা যাবে না, কিন্তু subclass দিয়ে করা যাবে।

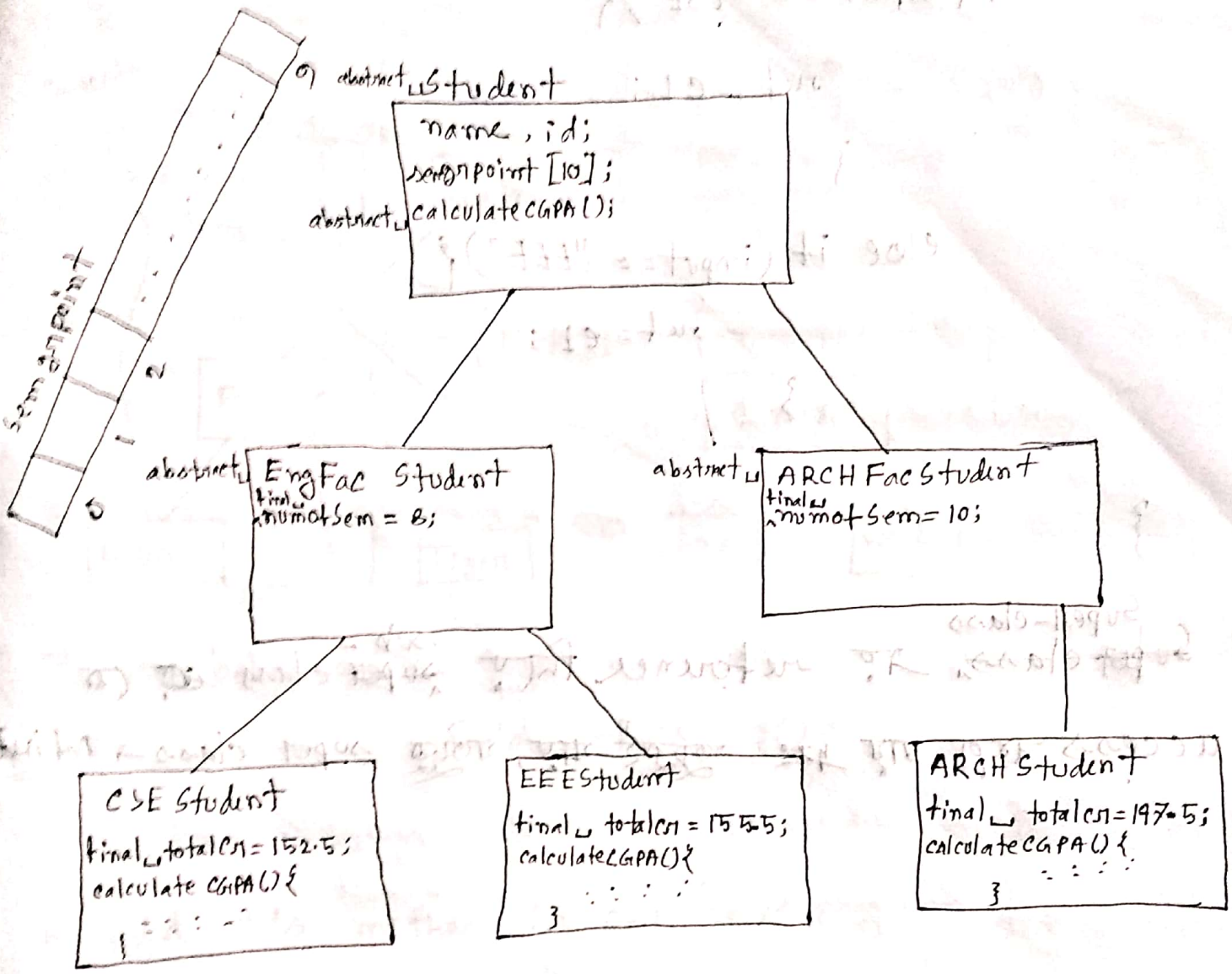
Suppose, abstract class এর object দিয়ে করা যাবে।

তখন এই class এর object দিয়ে abstract method কে call করতে হবে তখন কোন খালি হবে না (কারণ method এর কোন body নেই)। তাই abstract class এর কোন object create করা যাবে না।

Student s1 = new Student("Karim", 150.2);

s1.calculateCGPA();

Lecture - 17 (19-2-19)

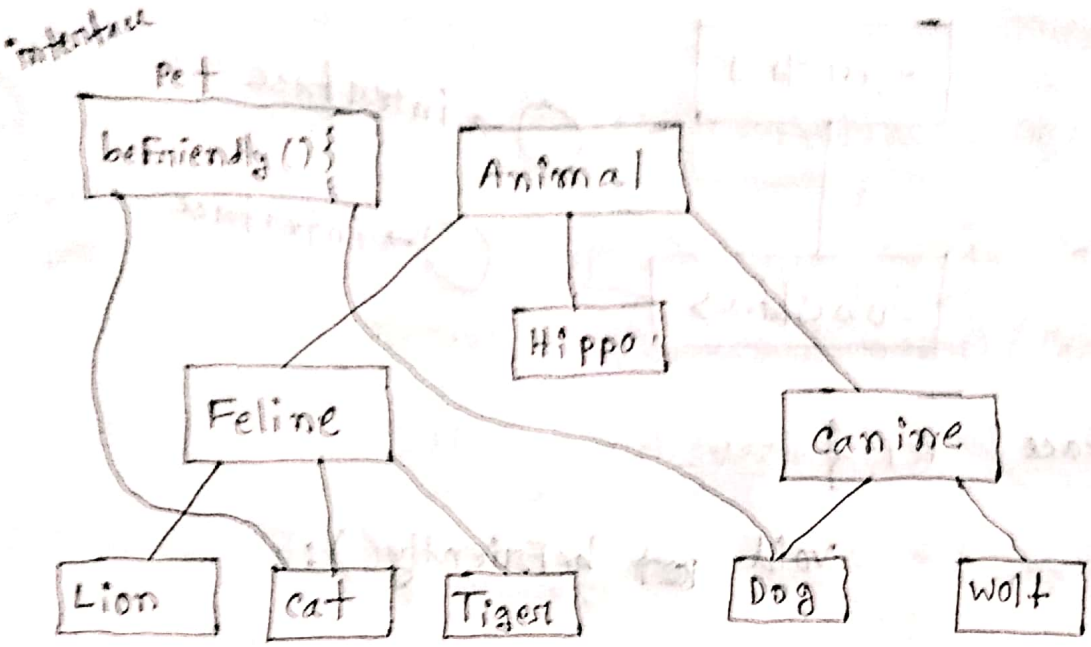



```
main(){
    CSEStudent c1 = new CSEStudent(...);
    EEEStudent e2 = new EEEStudent(...);

    Student ref;

    /*
        - take an input
        - depending on the input call either calculateCGPA() of
            CSE or EEE*/

    // input= take an input.
```

```
if (input == "CSE"){
        ret = c1;
    }
else if (input == "EEE"){
        ret = e1;
    }
}
```

* Super-class ~~Super class~~ এর reference দিয়ে sub-class ~~super class~~ কে access করা যায় এবং ততটুকুই যাবে, যতটুকু super class-এ defined

Lecture-18 (3-3-19)



→ a behaviour of a pet:

```
void beFriendly(){
    .....
}
```

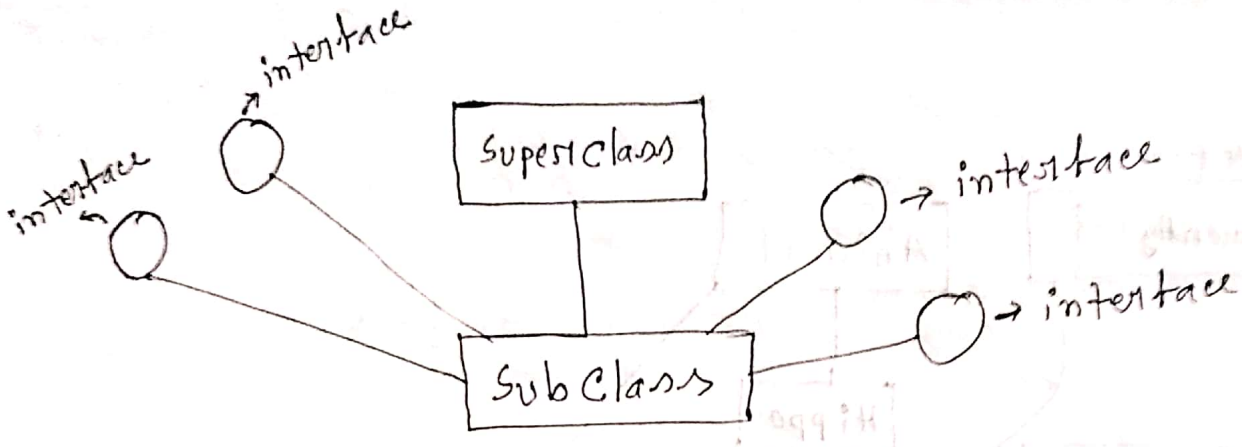
to add this method in our above program:

1. add the method to Animal class
2. " " " to cat & Dog classes
3. " " to Feline & Canine class
4. " " to Animal class but make it abstract.

→ Lion, Tiger, wolf, are not a pet Animal. So it's not acceptable to add the above methods in the super class.

→ not a good design.

→ Runtime polymorphism/Dynamic method dispatch का पता नहीं,




```
interface Pet {
(keyword)
        void beFriendly();

}

class Cat extends Feline implements Pet {
                              (keyword)

}
```

an interface is by definition an abstract. interface a কোন method এর body লেখা যায় না, শুধু declare করা যায়,

Lecture - 19 (5-3-19)

* General form of interface:

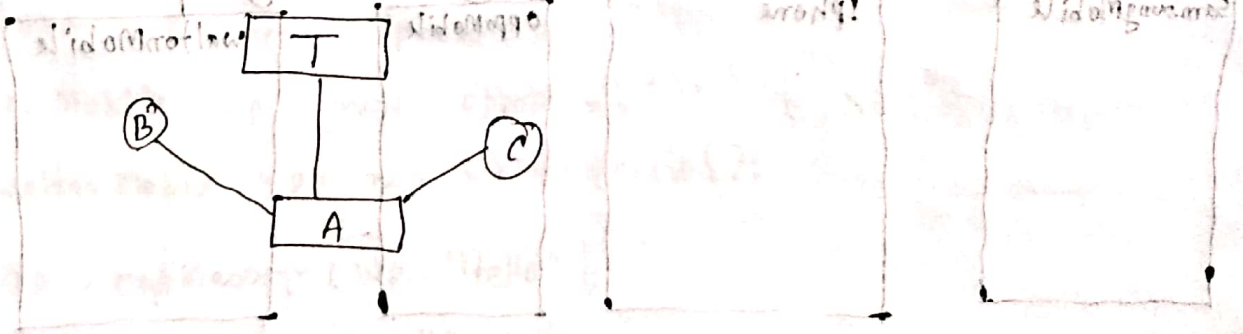
Access (Public or Private) interface interface_name {

return_type method_name1 (Parameter_list);

return_type method_name2 (Parameter_list);

type final_variable_1 = value;

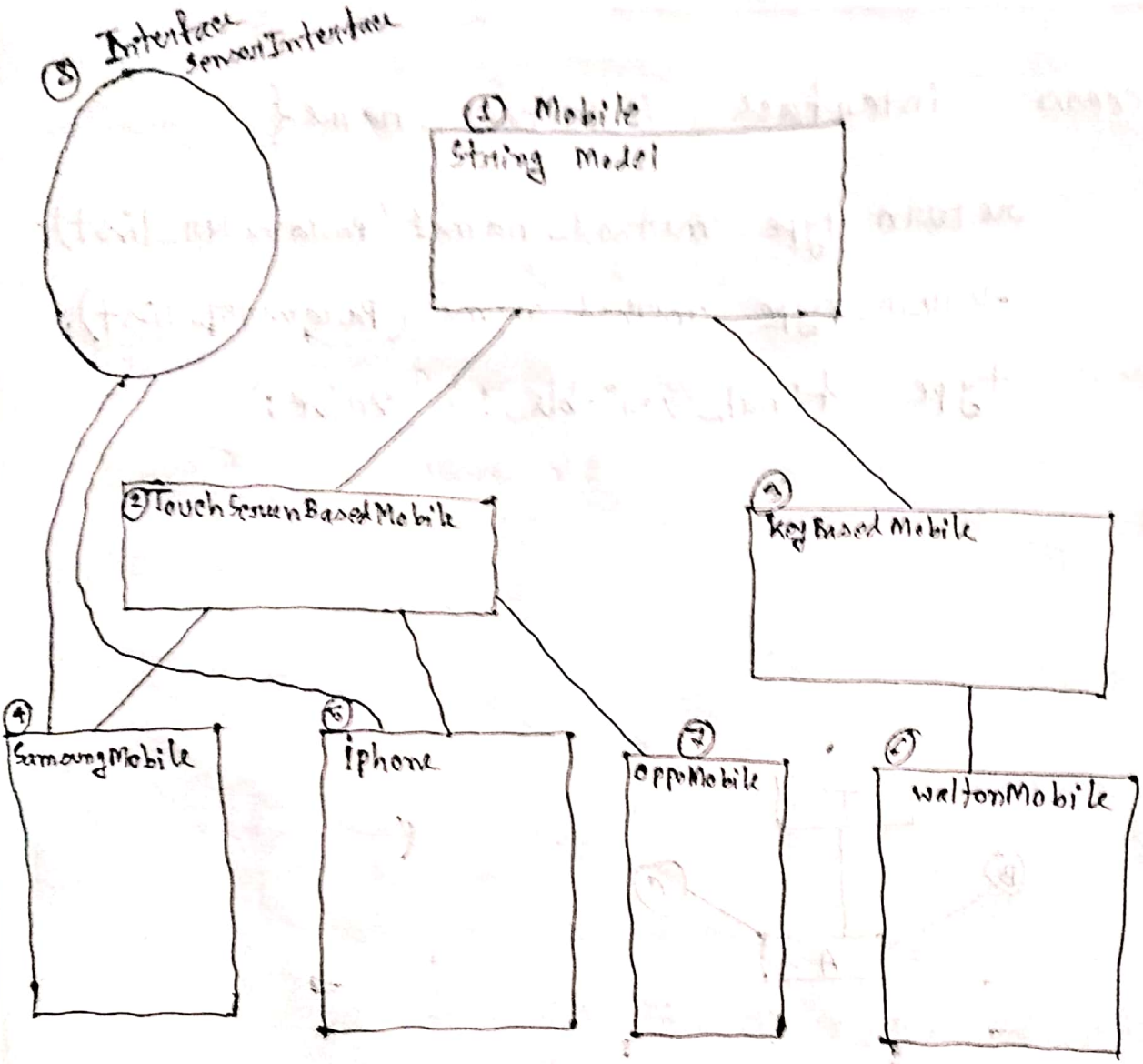
*



class A extends T implements B, C {

* interface এর method কে implement করার সময় method এর আগে public লিখতে হবে must. (Body লেখার সময়)

Lecture - 20 (6-3-19)



(1) ⇒ void sendingMessage(Mobile m, String message);
void receivingMessage(String message);
get Model();
constructor();

Sensor Methods:

1] getGPSPosition.

2] sendGPSPosition.

3] receiveGPSPosition.

4] getVelocity.

(2) ⇒ getGPSPosition();

sendGPSPosition(); SensorInterface m, int gpsdata);

receiveGPSPosition(int gpsdata);

getvelocity();

SamsungMobile sp = new Samsung();

iphone ip = new iphone();

oppoMobile op = new oppomobile();

waltonMobile wp = new waltonMobile();

sp.sendMessage(wp, "Hello");

wp.sendMessage(ip, "Hello");

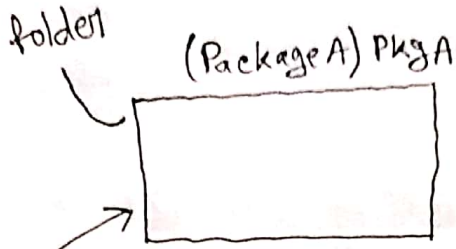
X op.sendGPSPosition(sp, 10); → oppo sensor interface ক implement করে নাই।

X ip.sendGPSPosition(op, 1);

ip.sendGPSPosition(sp, 10);

Leetusre- 21 (10-3-19)

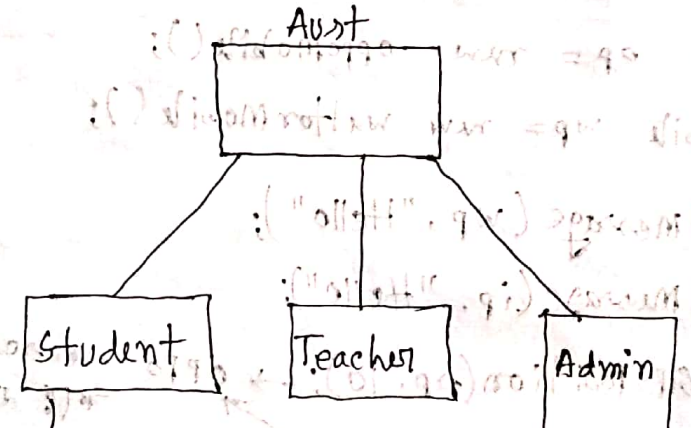
* Packages:



A.java

```
Package pkgA;
class A{

}
```

→ folder



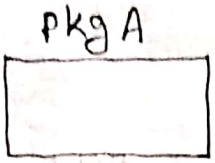
Package Aust.Student

(.) operatosr

Aust नाम sub-folder Student

Package.Aust.Admin

*




```
package pkgA;
public class A{
    public method(){
    }
}
```

```
package pkg B;
class B{
    public . . . . main(){
        A ob = new A();
    }
}
```

⊛ একই Package এ নাই

⊛ A পাবে না

*

```
package pkg B;
import pkgA.A;
class B{
    public . . . . (. . .){
        A = ob = new A();
    }
}
```

→ এই Package এর সব ক্লাস Access করতে চাইলে pkgA.*

import না করে:

```
Java.util.Scanner sc = new Java.util.Scanner
                       (System.in);
```

	Private	No Modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same Package Sub class	No	Yes	Yes	Yes
Same Package Non Sub class	No	Yes	Yes	Yes
Different Package Sub class	No	No	Yes	Yes
Different Package Non Sub class	No	No	No	Yes

No modifier public না।

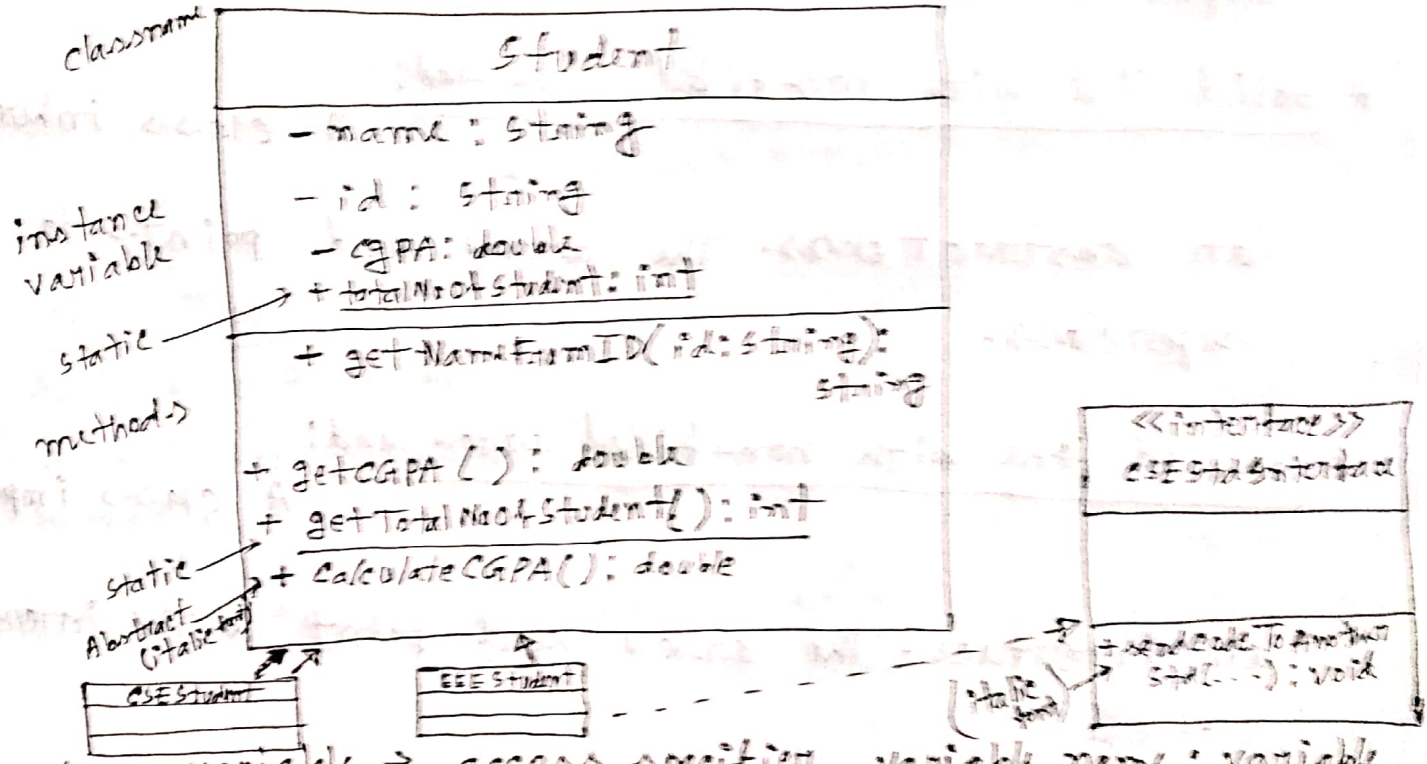
Same package এ থাকলে অথন public

Protected – নিজের Package এ যাবে।

Different এও যাবে যদি inheritance এর সম্পর্ক থাকে।

Lecture - 22 (12-3-19)

* Unified modeling language (UML):



instance variable → access specifier ␣ variable name : variable_data_type

+ : public

- : private

: protected

~ : default

methods → access specifier ␣ method name( ) : return type

variable_name : type

* Solid line with filled arrow head: A normal class inherits another class. The arrow head points to super class.

* Solid line with non-filled arrow head: A class inherits an abstract class. The arrow head points to superclass.

* dashed line with non-filled arrow head: A class implements an interface. The arrow head point to the interface.

* italic:
  - all abstract methods
  - all interface methods
  - Name of abstract class.

** Quiz #3:

Date:

Syllabus: abstract class + interface + UML.

Lecture-23(13-3-19)

* Exception Handling:

(Handled by object oriented concept)

* Exception Type:

(i) Arithmetic Exception
(ii) Array Index Out Of Bound Exception
(iii) formating Exception.

* When an exceptional condition arise:

i)

ii)

iii)

iv)

* try {

} catch(. . . .) { ——→ (ExceptioType reference)
(ArithmeticException e)

}

* Multiple catch clauses:

```
try {

} catch(. . . .){

} catch (. . . .) {

}
```

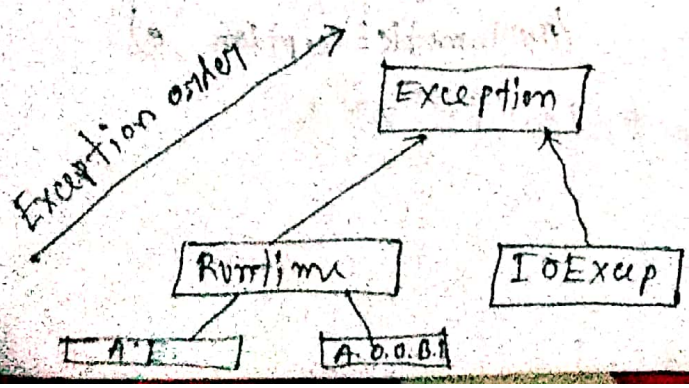
* lecture-24 (19-3-19)

* Java Exception Hierarchy:

i) All java exception class inherits, either directly or indirectly from class Exception.

ii) Programmers can extend this hierarchy to create their own exception class.

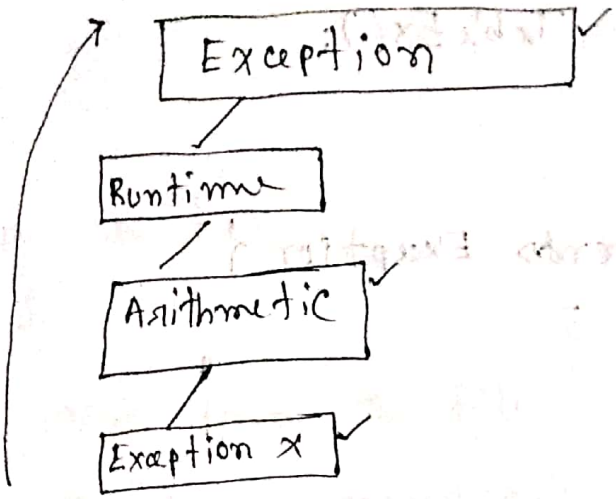
* catch ব্লক এর multicatch এ প্রথমে subclass লিখতে হবে, এবং Superclass লিখতে হবে।



* Lecture - 25 (27-3-19)

* try {
  ≡
  } catch (Exception e){
  ≡
  } catch (ArithmeticException e){ → will give compilation error. → Unreachable code
  ≡
  }

*



Key words:

try
catch
throw
finally
throws

try {

} catch (Exception x){
  ≡
} catch (ArithmeticException e){
  ≡
} catch (Exception e)
  ≡
}

* catch blocks should be written from subclass to super-class.

* throw :

throw ⎵ an object of a exception;

* ExceptionX ex = new ExceptionX();
  throw ex;

  or, throw ⎵ new ExceptionX();

* TableEx t = new TableEx();
  throw ⎵ t;

class TableEx extends Exception {
  ≡
}

* যেন class Exception class কে Inherit না করলে throw করা যাবে না।

* finally:

```
try{
    =
}catch(-  -){
    =
}catch(-  -){
    =
}finally{
    =
}
```

* Open an file

```
try{
    read from the file
    do some operations
    save the result to the file
}catch(-  -){
    handle the exception that occurs during
        operation.
}finally{
    close the file.
}
```

* throws:

* double division(int a, int b) throws ArithmeticException{

return a/b;

}

---o---

try{

division(3, 2);

}catch(ArithmeticException e){

=

}

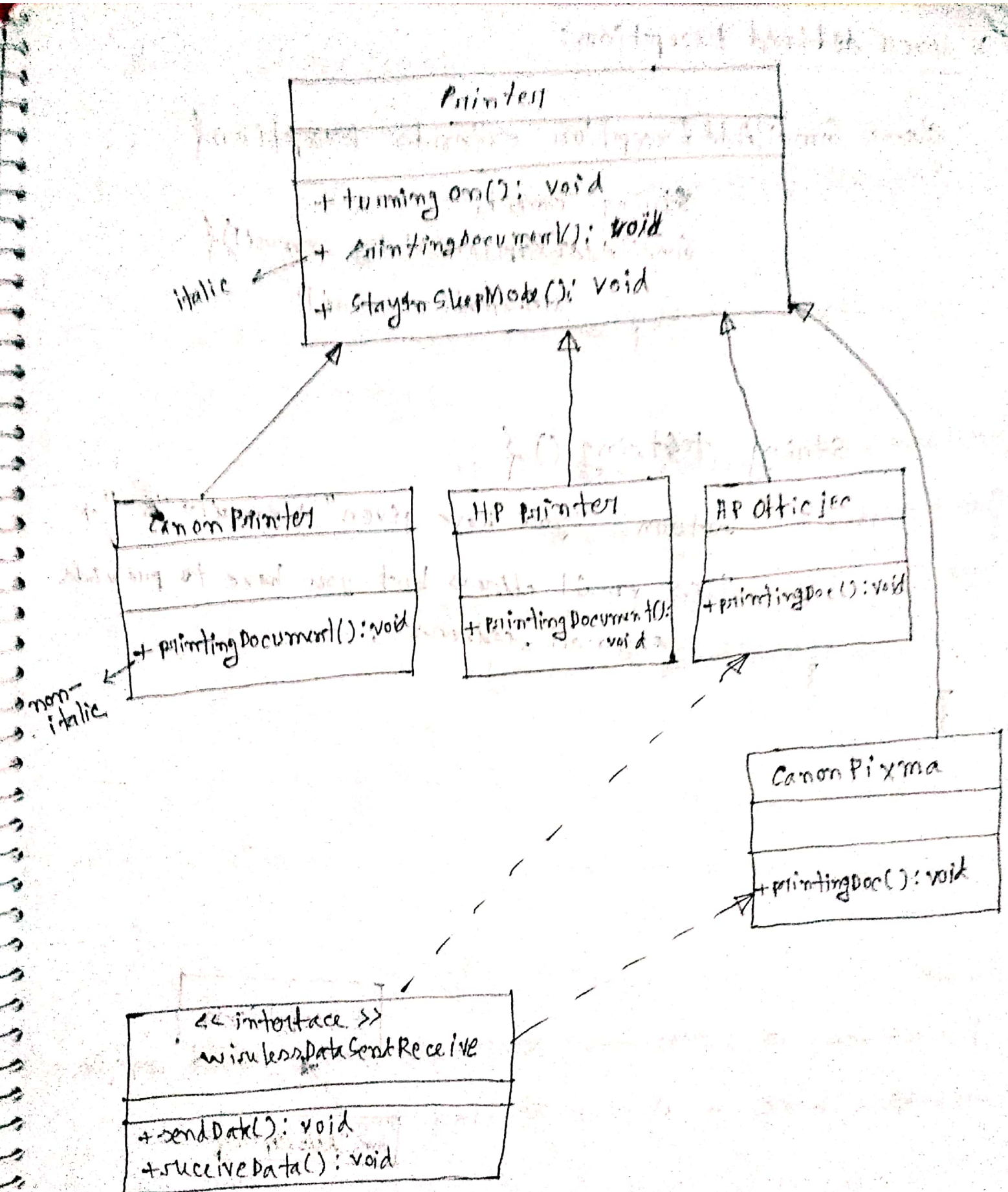
* Lecture-26 [(31-3-19)]

* Quiz question solve:

Canon Printer → turning on
HP " printing Documents
stay on sleep mode

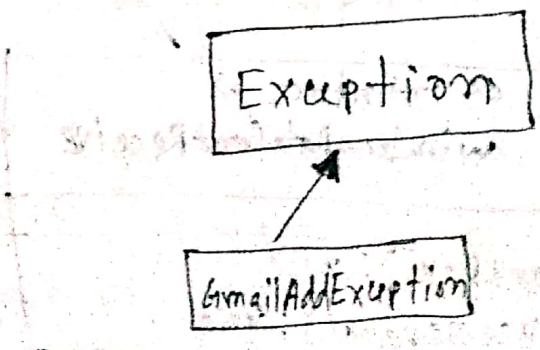
Specific printer → HP OfficeJet
→ Canon Pixma → wireless sends receive.

design a UML diagram.



* Users defined Exception:

```
class GmailAddException extends Exception{
        String email;
        GmailAddException(String email){
            this.email = email;
        }

        String toString(){
            return "you have given" + email + " " +
            " is email address but you have to provide
                a gmail address";
        }
}
```



```
Class StudentFormDemo{
    public ... void main(...){
        Scanner sc=new Scanner(System.in);
        ≡ → (whatever you do)
        String inEmailAdd = sc.nextLine();
        try{
            if( !inEmailAdd.endwith("@gmail.com")
                throw new GmailAddException(inEmailAdd);
                        (object)
            }catch(GmailAddException e){
                System.out.println(e);
            }
        }
}
```

*If you have given shahrian@yahoo.com. as an email address but you have to provide a gmail address.

Lecture-27 (2-4-13)

* Multithreading:

* A multithreaded program contain two or more parts that can run concurrently.
* Each part of such a program is called thread.
* There are two kinds of multitasking:
  i) Process-based multitasking.
  ii) Thread-based multitasking.
* Process-based multitasking allows you to run two or more program concurrently.
* In thread-based multitasking a single program can perform two or more task simultaneously.

* The Thread class and Runnable Interface:

* Java's multithreading systems is built upon Thread class.
* To create a thread, you will either extends thread class or implements the Runnable interface.

* The thread class defines several methods:

i] getName → obtain a thread's name.

ii] getPriority → " " " priority.

iii] isAlive → Determine if a thread is still running

iv] join → wait for a thread to terminate

v] run → Entry point for the thread

vi] sleep → Suspend a thread for a period of time.

vii] start →

* class A implements Runnable{

public void run(){

for(int i=0; i<100; i++)

{

System.out.println(

(Thread.currentThread().getName()+"printing"+

i);

try{

Thread.sleep(10);

}catch(InterruptedException e){

e.printStackTrace();

}

}

}

```
class ThreadDemo{
    public . . . main(. . .){

        Thread t1, t2;

        A ob = new A();
        B ob = new B();
        t1 = new Thread(ob, "t1");
        t2 = new Thread(ob, "t2");
        t1.start();
        t2.start();
        t3 = new Thread(ob, "t3");

    }

}
```

class B doesn't implement runnable

ob → class object which implements Runnable.

"t1" → Thread name (optional)

* main thread by default run করে,

Lecture-21 (3-4-19)

Lecture-22 (8-4-19)

* Thread का priority 1-10 तक होता है।
* default thread का priority 5 (पांच) होता है।

```
class CurrentThreadDemo{
    public static ... (. . .) {
        Thread t = Thread.currentThread();
        System.out.println("current thread:" +t);
    // change the name of the thread
        t.setName("My thread");
        System.out.println("After name change:" +t);
    try{
        for(int n=5; n>0; n--)
        {
```

* isAlive(): final boolean isAlive();

- Return true if the thread upon which it is called still running, otherwise return false.

* join():

```
final void join() throws InterruptedException
{
}
```

lecture-30 (10-4-19)

* Synchronization:

* When two or more threads need to access to a shared resource, it needs to ensure that the shared resource is

* Inter-thread communication:

* wait();
* notify();
* notifyAll();