7E2002

Part 1
SC2002

abstract class X {
  public abstract void y();
}

- abstract class = a restricted class that cannot be use to create object (to access, must inherited from another class)
- abstract method = used only in abstract class

main concept of OO :

- Abstraction → abstract classes / interfaces

hide unwanted info from user
denotes essential characteristic of object that distinguish

↳ object = : attribute
: actions

↳ state of object = data/variables
↳ attributes = data/variables = • Methods = • services • Operation • Interfaces
↳ behaviour of object =
- doesn't have implementation

abstraction = superclass with no meaningful object
= cannot used to create object

- Encapsulation / info hiding

- protect object's private data
  ↳ still can be accessed through public methods
- hides details / implementation of class

X Polymorphism

interface X {
  Public void y();
}

- Java interface
  ↳ doesn't have body → class a implements X {
  ↳ access by implements }
  ↳ contain only abstract methods and constants (static final)

! Class can implements multiple interface but extend only 1

- Inheritance (is a) (Public A extends B { } ; generalisation)

SuperClass = parent
SubClass = child (inherits one/more classes) specialisations

↳ can override method
↳ have all parent function + own function

method Overrides
↳ refinement = with super keyword
↳ replacement = override

- Constructor B use in A

Public Class A extend B {
  Public A() {
    super();
  }
}

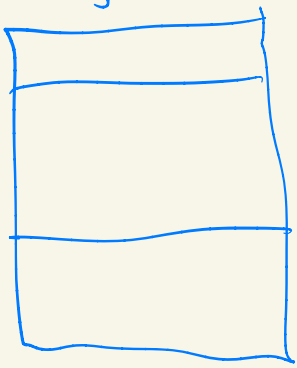
(A B = new A();)

- Polymorphism

↳ create class object that performs operation appropriate to its class

↳ make prgm extensible
↳ method overriding

Class diagram



→ Class name

} attribute / data / properties / variables
$ name : type

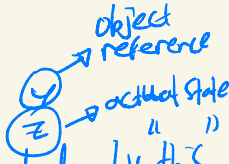
} methods
$ name(constructor) : return type

} Class members

instance

$		Class	Package	Subclass	World
+	public	✓	✓	✓	✓
−	private	✓	×	×	×
#	protected	✓	✓	✓	×
~	Package (default) ↳ for visibility	✓	✓	×	×

instantiation: a b = new a(); — memory: x, y



! Static = method is not attached to specific instance and run by this
↳ for assigning it as class method ↳ access class method
↳ can access without instance of class (Class name.method)
↳ cannot reference instance variable/method

! Static final mean are fixed and immutable

! java destructor = finalize()

! java copyObjects() function to copy object

! Public class a (b) {
int b;
this.b = b;
}

! Accessor = get method
! Mutators = set method (maintain data consistency)
! application class use
Public static void main (String[] args) { }

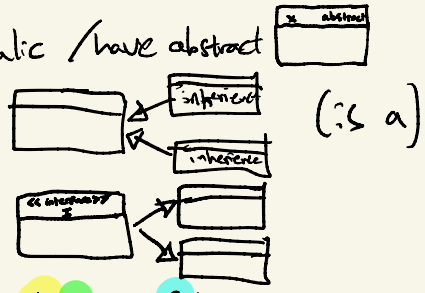
- object composition (has-a)
  - object as data member

- Final method = Cannot be overridden in subclasses
- Final class = Cannot be superclass / inherited by others
  - Functions: - improve security
    - improve efficiency

Java Naming Convention
- Lowercase : package name
- Uppercase : exams, constants
- CamelCase (Cap then Lower): Classes n interface (Cap next word)
- Mixed case : (Lower then Cap next word) : methods, variables

Java supports multiple interface

- Class diagram
- abstraction = italic / have abstract
- inheritance
- interface (can be multiple)
- polymorphism



public x extends I

public interface I { only constant and abstract method; only public
public x implements I

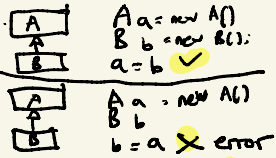
```
A b = new C();
b.go();
```
→ show this!

```
A = class A {
    public void go() { }
}
C = class C extends A {
    public void go() { }
}
```

binding = occurs at bound @ compile time for static
occurs at execution time/run time for dynamic ← polymorphism use this
↳ all except private, final, and static methods

exception = occurs at run time

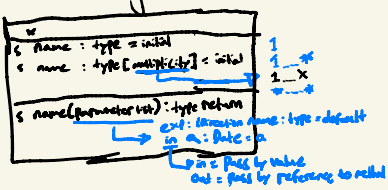
Type casting → upcasting
downcasting



Part 2

- Inheritance = is-A
- association = Has-A
  - Aggregation
  - Composition
- dependency = uses-A

Class diagram :



association

1 = 1

1..* = >=1

1..x = 1 to x

* = a lot

Example

Shelf 1 ——> 1..* books

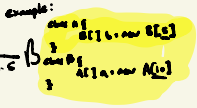
- 1 shelf can have >=1 books
- book doesn't look @ shelf

multiplicity only for ↓

Has A

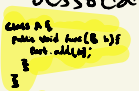
A 1..10 ——> 1..c B

example:

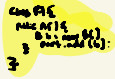


A ——> B association x or Link rolename

association = Has A



A ◇—— parts B aggregation A part of B, A can be alone

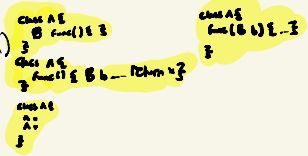


A ◆—— parts B composition A part of B, A belong to B

A ——▷ B is a A inherit from B

A ..--▷ B realises B = interface, A inherit from B

A ---> B dependency (uses-A) «x»



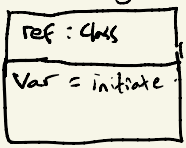
A ⌐ Reflexive association

= ——✕——> Unidirection

= ——▶

= ——→

Object diagram :



Stereotypes

- Entity class (store data)
- Boundary class (interfaces, input/output)
- Logic class (application)

Rigidity = tendency of hard to change (1 change cone another)
Fragility = tendency of breaks in many place
Immobility = inability to reuse

inherience VS association (composition)
inherience VS role (enum)

SOLID:

- Single responsibility Principle (SRP)
  ↳ each class have 1 responsibility

!collection
- Open-closed principle (OCP)
  ↳ able to change without changing source code

!abstraction is key
- Liskov Substitution principle (LSP)
  ↳ Subclass should be not expect more/providing less
    Funct of parent
    - subclass must do all things super class do
    - subclass must not bring trouble super class dont
- Interface segregation principle (ISP)
  ↳ Classes should not depend on interfaces they do not need

- Dependency injection principle (DIP)
  ↳ high lvl module shouldn't depend upon low lvl module both should depend upon abstraction
  ↳ Details should depend upon abstraction

C VS C++
↳ use std::cout<< —<<endl; instead of printf
↳ use cin >> x instead of scanf

interface/implementation

h file (") — cpp file (") — in main.cpp (")

h file:
```
class a {
private:
    int x
public:
    void foo();
};
```

cpp file:
```
#include "a.h"
void a::foo() {
}
```

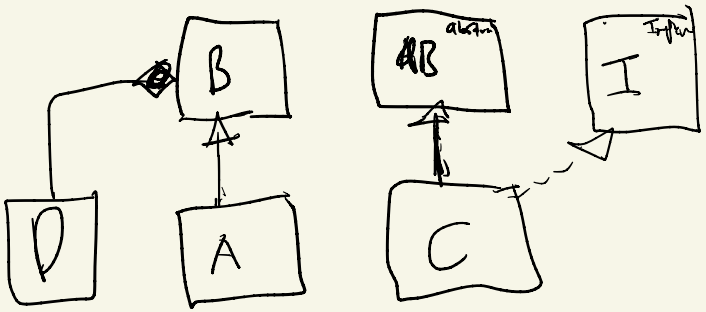
in main.cpp:
```
#include "a.h"
int main() {
    a a;
    return 0;
}
```

Constructor `x()` : ~~~
destructor `~x()` ;

h file benefits:
↳ speeds up compile time
↳ keep code organised

code testing:

- private
  final
  static
- inherience
- composition
- polymorphism → super.printInfo();
- Combine
- abstraction
- Upcast
  downcast
- method overriding (Polymorphism)
  - SuperClass method
  - abstract method
  - interface
    - method of ConcreteClass using interface
-



- String Compare
- array

A Wrapper class is a class whose object wraps or contains primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store primitive data types. In other words, we can wrap a primitive value into a wrapper class object.

Need of Wrapper Classes

1. They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).
2. The classes in java.util package handles only objects and hence wrapper classes help in this case also.
3. Data structures in the Collection framework, such as ArrayList and Vector, store only objects (reference types) and not primitive types.
4. An object is needed to support synchronization in multithreading.

Primitive Data types and their Corresponding Wrapper class

Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

Autoboxing and Unboxing

Autoboxing: Automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing. For example – conversion of int to Integer, long to Long, double to Double etc.

TUT 1

1) Class = Student, NTU, Book, person, Engine, Liquid Transformer, Force
Object = NTU, Michael Jackson, Person1
Property/Attribute/behaviour = Age, Color, work, result, , Force, Shoot

2) School → Student (has), Lecturer (has), Course (teach), Classroom, academic result

Student	Lecturer	Course	Classroom
name	name	Course code	TUT Room ID
age	age	resources	
course	course		
Student ID	Staff ID		
Faculty	faculty		

take

3)
```
public bubbleSort {
    Public Void bubble (int a[], int n) {
        int i, j, t;
        for(i=n-2; i>=0; i--) {
            for(j=0; j<=i; j++){
                if(a[j] > a[j+1]) {
                    t = a[j];
                    a[j] = a[j+1];
                    a[j+1] = t;
                }
            }
        }
    }
    Public static Void main (String args[]) {
        int a[100], n, i;
        System.out.println ("la~~~ );
        Scanner sc = new Scanner(System.in);
        n = sc.nextInt();
        for(i=0; i<=n-1; i++) {
            System out.println (~~~.
            a[i] = sc.nextInt();
        }
        bubble(a, n);
        system.out.println ~~~);
        for( i=0; i<=n-1; i++) {
            system.out.println (          ,a[i]);
        }
    }
}
```

TUT 2..

1]

```
public class Circle {
    private double radius;

    public Circle (double rad){
        this.radius = rad;
    }
    public Void SetRadius (double rad) {
        radius = rad;
    }
    public double area() {
        return  Math.pi * radius * radius;  printArea();
    }
    public double circumference () {
        return                          printCircumference();
    }
    Public Void printArea() {
        System.out.println("~~" + Area);
    }
    Public Void printCircumference() {
        System.out.println("~~" + ~~);
    }
}

public static Void main (String args[]) {
    System.out.println(~~~~);
    Scanner sc = new Scanner(System.in);
    int choice = sc.nextInt();
    do {case 1:

    }(while choice != 4)

}
}
```

2)

```
Public class Dice {
    Private int valueOfDice;
    Public Dice() {

    }
    Public Void setDiceValue() {

    }
    Public int getDiceValue() {
        return ValueOfDice;
    }
    public Void printDiceValue() {
        System.out.println("~~~" + ValueOfDice);
    }
}
```

```
public DiceApp {
    Public Static Void main(String args[]) {
        System.out.println("~~~~~");
        Scanner sc = new Scanner(System.in);
        sc.next();
        Dice di = new Dice();
        System.out.println("~~~~~");
        sc.next();
    }
}
```

TUT 3

1)

```
Public class VendingMachine {
    Public VendingMachine() {

    }
    Public double selectDrink() {

    }
    Public double insertCoins(double drinkCost){

    }
    public Void checkChange(double amount, double drinkCost){

    }
    Public void printReceipt() {

    }
}
```

```
Public class VendingMachineApp {
    Public Static Void main(String Args[]) {
        System.out.print("~~~~~");
        Scanner sc = new Scanner(System.in);
        int choice = sc.nextInt();
        double sum = 0;
        Switch(choice){
            case 1:
            sum=0;
            while(sum < 3.0) {
                System.out.println(~~~);
                char c = sc.next().charAt();
                if(~~~)
                    sum += ~~
            }
        }
    }
}
```

2)

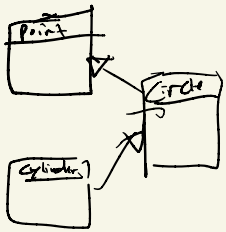
```
Public Class Point {
    protected int x;
    protected int y;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
    Public String toString() {
        return "X:"+x.toString() + " " +y.toString();
    }
    Public Void setPoint(int x, int y) {
        this.x = x;
        this.y = y;
    }
    public int getX() { return x; }
    public int getY() { return y; }
}
```

```
Public class Circle extends Point {
    Private double radius;

    Public Circle(int x, int y){
        Super(x, y);
    }
    Public Void setRadius(double radius) {
        this.radius = radius;
    }
    Public double getRadius() {
        return radius;
    }
    Public double area() {
        return ~~~;
    }
}
```

```
public class Cylinder extends Circle {
private int h;
    Public Cylinder(int x, int y, int h){
        Super(x, y); this.h = h;
    }

}
```



TUT5

1) a, b, c, d

2) d Class D will have error

b) i) C c = D D
ii) A a = C not OK (C obtained) error
iii) A a = F error Class A don't have

Upcasting = implicit
downcasting = explicit

c) 4) B b = E error
5) A a = F A
6) A a = F error
E e = C error

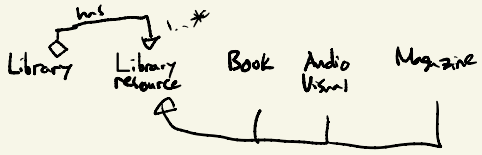
3)

Compilation → declared type

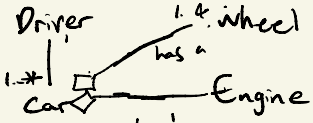
object type

TUT6

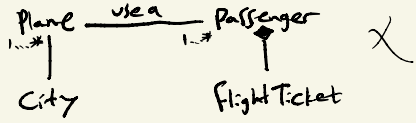
1) i)



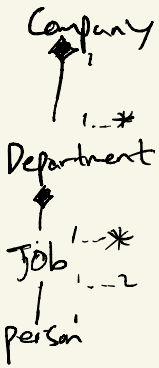
ii)



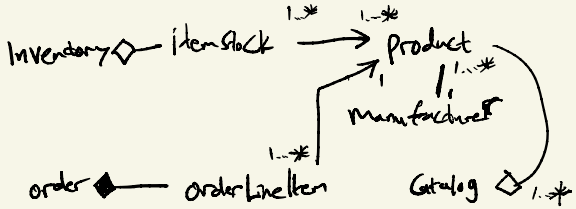
iii)



iv)

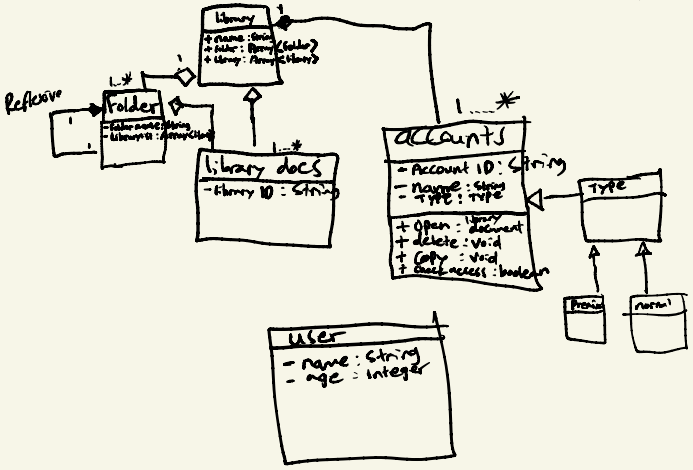


v)



2) Class
- library
accounts
users
- library docs
- Folder

method
checkedaccess
open
delete
copy



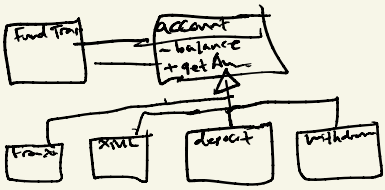
TUT 7

1) Open-Close

3)

2) a) alot

b)



c) need to modify account class

d) ~~Liskov~~ X DIP and ISP

TUT 8

1) c)

```
#include<iostream>
using namespace std;
class bubble (int a[], int n){
public:
    int i,j,t;
    for(i=0;i<n;i++){
        for(j=0;j<n-i-1;j++){
            if(a[j]>a[j+1]){
                t=a[j];
                a[j]=a[j+1];
                a[j+1]=t;
            }
        }
    }
};
int main(){
    int n;
    cout<<"Enter the number of elements: ";
    cin>>n;
    int a[n];
    cout<<"Enter the elements: ";
    for(int i=0;i<n;i++){
        cin>>a[i];
    }
    bubble b;
    b.bubble(a,n);
    cout<<"Sorted array: ";
    for(int i=0;i<n;i++){
        cout<<a[i]<<" ";
    }
    return 0;
}
```

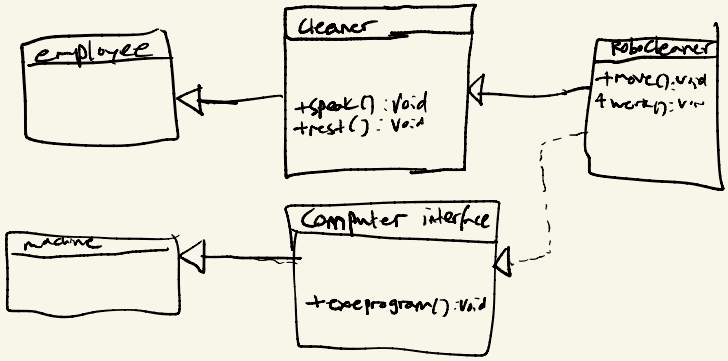
TUT 9

1) a) error compilation

2) I am in base class;

2020-2021 S2

1) a) i)



b) Constructing Class B → Constructor A, Constructor B ✓

C → Constructor A, Constructor B ✓, Constructor C

ob.method(100) → A 1, C 1, B 1, B 1

oC.method(100)

2) a)

```
Class TestClass {
    private double price;
    private String name;
    static int numberOfProduct=0;
    Class TestClass(double price,String name){
        this.price =price;
        this.name = name;
    }
    greetings() {
        System
    }
    update
        this.price = price
    this.name = name
    print() {
    System.out.println("the price is " +price + " for "+name);
    }
    numberof product
    }
```

```
public class TestClassApp
{
    public static void main (String[] args){
        TestClass.greetings();
        TestClass myproduct = new TestClass(0,"Aiga");
        myproduct.print();
        myproduct.update(3,"Ammy");
        myproduct.print();
        myproduct.printNumberOfProduct();
        TestClass bmwproduct = new TestClass(0, "tom");
        bmwproduct.print();
    }
}
```

b) 1: upcasting, compile OK!
2: equal in 2
3: downcasting, OK!
4: diferent 2

3. (a) **Study the following description of an application :**

MOVPUR is an application to computerize the processes of making online purchase of movie tickets. It will be used by the movie-goers and cinema staff in each cineplex in different locations. Each cineplex will have 3 or more cinema theatres. The movie ticket price is charged according to the age group of movie-goer, day of the week or public holiday. The age group can be adult, senior citizen or child. MOVPUR allows movie-goers to query the movie listings and show times. From the movie listings, movie-goer can also view the information about the movie like title, showing status, classification, synopsis, directors, casts and reviewers' ratings. The showing status can be "Coming Soon", "Preview" or "Now Showing". The movie classification can be PG, NC16, M18 or R21. Only registered movie-goer with a valid identification and password can make online purchases. Each movie ticket will indicate the movie title, show time and date, seat number and the cinema. Registered movie-goer can also check their history of bookings.

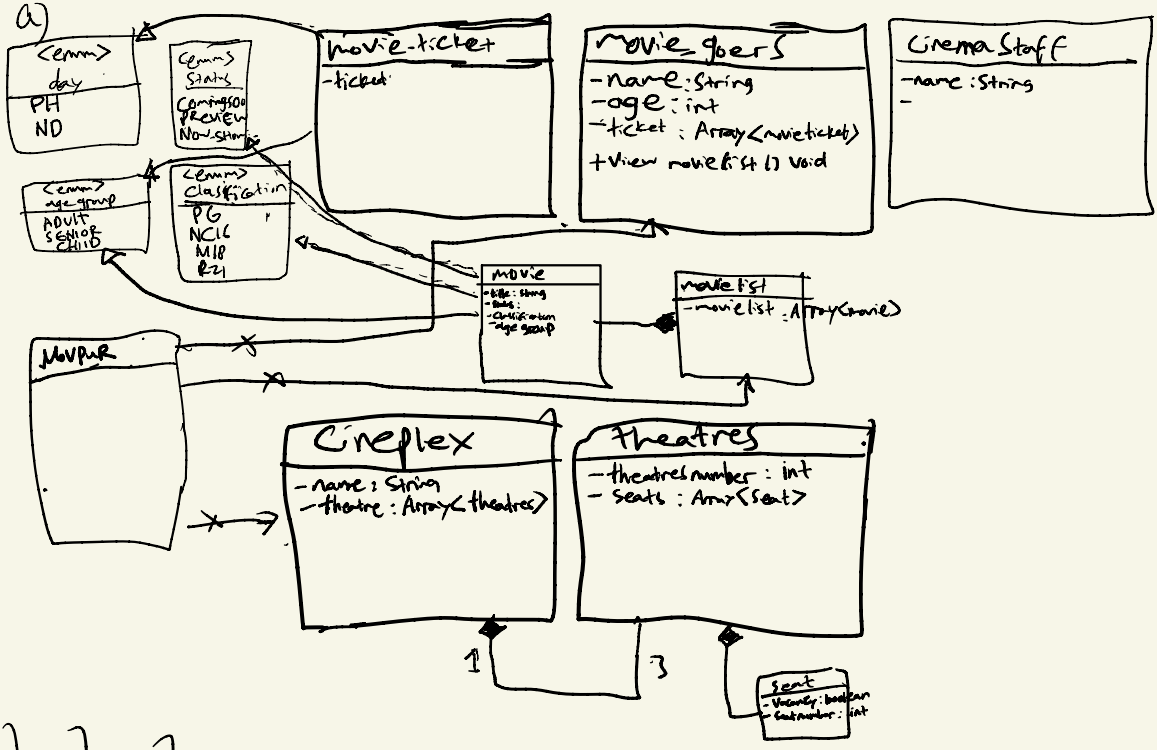
You are tasked to identify the **entity** classes needed to build the application based on the description above.

Show your design in a Class Diagram. Your Class Diagram should show clearly the relationships between classes, enumeration, relevant attributes (at least TWO), logical multiplicities, meaningful role names, association names and constraint(s), if any. You need not show the class methods.

(13 marks)

(b) The UML **Sequence Diagram** in Appendix A (page 9) shows the objects' interactions of a scenario flow in a particular application. Using the details depicted in the diagram, write the preliminary Java code for

3) a)



4) a) i)

account: Owner, accountID, amount

Owner: withdraw, deposit

i) h file

```
#include<iostream>
#IFNDEF BANK_H
#DEFINE BANK_H
Using namespace std;
Class Account {
    Private:
        String name;
        String accountNO;
        float  amount;
    Public:
        Account(){};
        Account(String owner){};
        Virtual ~Account(){};
        Virtual float getAmount(){ return amount; }
        Virtual void deposit(float amount){ amount += amount; }
        Virtual float withdraw(float x ){
            if(x>amount){
                cout<<"Overdraft!!!" << endl;
                return 0;
            }
            amount = amount - x;
            return x;
        }
};
```

ii)

```
#include<iostream>
Using namespace std;
class PrivilegedAccount: Public Account{
    Private:
        String name;
        float amount;
        float limit;
    Public:
        PrivilegedAccount(String name):Account(name)
        {}
        Void withdraw
```

```
class TestPolygon {

public :
    static void printArea(Polygon& poly) {
        float area = poly.calArea( );
    cout <<"Reference The area of the " << StringKindofPolygon[poly.getPolytype()]  <<  " is " << area << endl;
    }

    static void printArea(Polygon* poly) {
        float area = poly->calArea( );
    cout <<"Pointer The area of the " << StringKindofPolygon[poly->getPolytype()]  <<  " is " << area << endl;
    }
};


int main() {

        Rectangle rect1("Rect1", 3.0f, 4.0f);
        rect1.printWidthHeight();
        TestPolygon::printArea(rect1);
        TestPolygon::printArea(&rect1);

        Triangle *triang1= new Triangle("Triang1", 3.0f, 4.0f);
        triang1->printWidthHeight();
        TestPolygon::printArea(*triang1);
        TestPolygon::printArea(triang1);

        delete triang1;
    }
```

if Triagle* triang

both pass by ref

```
Width = 3 Height = 4
Reference The area of the POLY_RECT is 12
Pointer The area of the POLY_RECT is 12
Width = 3 Height = 4
Reference The area of the POLY_TRIANG is 6
Pointer The area of the POLY_TRIANG is 6
```