Object-Oriented Programming – Assignment 03

1. What is the purpose of the super keyword in Java?

super keyword allows the subclass to reference a superclass constructor.
When creating a subclass the constructor of the subclass should call the constructor of the super class.we use super keyword to call the constructor we need of the superclass.

2. Describe the concept of method overriding in Java.

When we have the same method in superclass and subclass changing only the body of the method that belongs to the superclass in the subclass is method overriding.
The method has the same method name and same parameter list in both the superclass and the sub class.

3. Explain the difference between super and this in Java.

This keyword refers to the current object that we put this keyword in.
Super keyword refers to the super class.

4. Can a subclass access protected member of its superclass? Provide an example.

Yes. Protected means it can be accessed within the same package.
But even if the subclass is in another package it can access a protected member.

```
class B {
    protected int a = 10;
        protected void method1() {
        System.out.println("a protected method.");
    }
}

class C extends B {
    public void accessProtected() {
            System.out.println("Protected member : " + a);
            method1();
    }
}

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

```
            C c1 = new C();
            c1.accessProtected();
    }
}
```

5. Write a Java program to demonstrate method overriding.

```
class Vehicle {
    public void park() {
        System.out.println("Vehicle park");
    }
}

class Car extends Vehicle {

    public void park() {
        System.out.println("Car park");
    }
}

class Bike extends Vehicle {

    public void park() {
        System.out.println("Bike park");
    }
}

public class Main5 {
    public static void main(String[] args) {
        Vehicle Vehicle1 = new Vehicle();
        Vehicle Car1 = new Car();
        Vehicle Bike1 = new Bike();

        Vehicle1.park();
        Car1.park();
        Bike1.park();
    }
}
```

6. Explain how a subclass can call a superclass constructor with parameters with

examples.

```
class A {
    int a;


    A(int a) {

        System.out.println("A constructor: " + a);
    }


    A(int a, int a2) {

        System.out.println("A constructor: " + a + ", " + a2);
    }
}

class B extends A {

    B() {

        super(10,20);
        System.out.println("B constructor");
    }
}

public class Main5 {
    public static void main(String args[]) {
        B b1 = new B();
    }
}
```

By using super(10,20); argument we are calling , A(int a, int a2) constructor of the super class.
If we use a constructor like super(10); we can call , A(int a) constructor of the super class.

7. Inheritance is one of the key concepts in Object Oriented Programming. Describe “inheritance” with real world examples.

Inheritance is the process where new classes(sub classes) inherit properties and behaviors of an existing class(super class) .
This helps code reusability.

Example:

Super class > Vehicle
Sub classes > Car
Bike
Bus
Ship

```
class Vehicle {
    public void park() {
        System.out.println("Vehicle park");
    }
}

class Car extends Vehicle {

    public void park() {
        System.out.println("Car park");
    }
}

class Bike extends Vehicle {

    public void park() {
        System.out.println("Bike park");
    }
}

class Bus extends Vehicle {

    public void park() {
        System.out.println("Bus station");
    }
}

class Ship extends Vehicle {

    public void park() {
        System.out.println("Harbour");
    }
}

public class Main5 {
    public static void main(String[] args) {
```

```
        Vehicle Vehicle1 = new Vehicle();
        Vehicle Car1 = new Car();
        Vehicle Bike1 = new Bike();
        Vehicle Bus1 = new Bus();
        Vehicle Ship1 = new Ship();

        Vehicle1.park();
        Car1.park();
        Bike1.park();
        Bus1.park();
        Ship1.park();
    }
}
```

8. Given:

```
import javax.swing.JFrame;
class CalculatorView extends JFrame{
//Just a window
}
class Demo{
public static void main(String []args){
CalculatorView v1=new CalculatorView();
v1.setSize(300,300);
v1.setTitle("Calculator");
v1.setDefaultCloseOperation(

CalculatorView.EXIT_ON_CLOSE);

v1.setVisible(true);
}
}
```

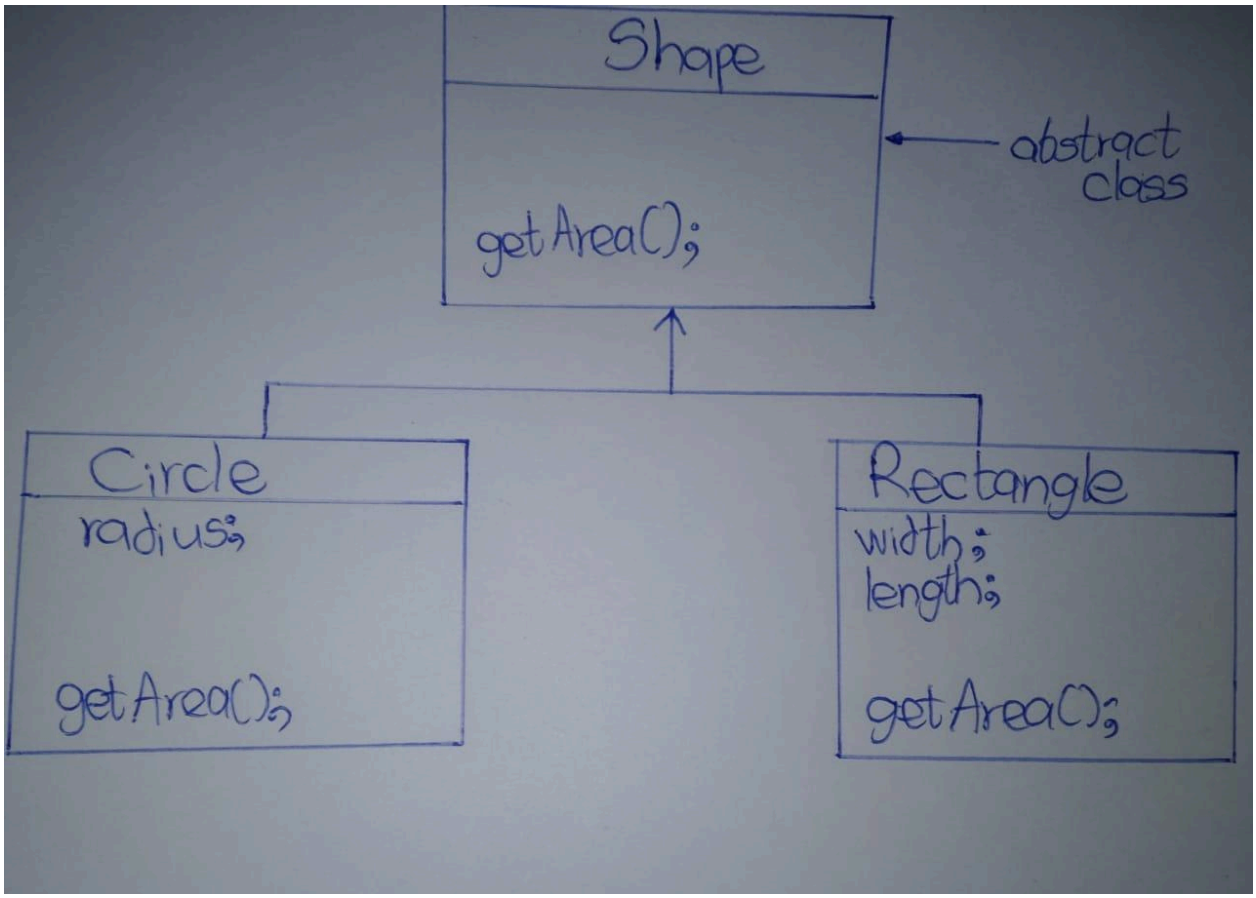
Describe the software reusability in OOP using above java application.

Because of Inheritance (Calculator is a JFrame) CalculatorView can reuse functions(methods , attributes) of JFrame.

9. Write an inheritance hierarchy for classes Quadrilateral, Trapezoid, Parallelogram, Rectangle and Square. Use Quadrilateral as the superclass of the hierarchy. Create and use a Point class to represent the points in each shape. Make the hierarchy as deep (i.e., as many levels) as possible. Specify the instance variables and methods for each class. The private instance variables of Quadrilateral should be the x-y coordinate pairs for the four endpoints of the Quadrilateral. Write a program that instantiates objects of your

classes and outputs each object's area (except Quadrilateral).

10. Draw a class diagram illustrating a scenario where an abstract class Shape is extended by classes Circle and Rectangle.



11. What is the purpose of the dynamic method dispatch in Java? Explain using real world examples.

Dynamic method dispatch (runtime polymorphism) allows java to support method overriding.

Compiler thinks the method in the super class is the one running but what actually runs is the overriding method in the subclass.

```
class Vehicle{
        public void park(){
                System.out.println("Vehicle Parking");
        }
```

```
}
class Car extends Vehicle{
        public void park(){
                System.out.println("Car Parking");
        }
}
class Example{
        public static void main(String[] args) {
                Vehicle v1 = new Car();
                v1.park();
        }
}
```

12. Given:

```
class A {
A() {
System.out.println("A");
}
}
class B extends A {
B() {
System.out.println("B");
}
}
class C extends B {
C() {
System.out.println("C");
}
}
public class Test {
public static void main(String[] args) {
new C();
}
}
```

What will be the output of this code and why? Provide explanation.

Output > A
B
C

B extends A (B is a A) and C extends B (C is a B) ,so by calling the class C we can access both B and A methods too.

13. Given the following classes:

```
class Super {
protected void method() {
System.out.println("Super method");
}
}
class Sub extends Super {
private void method() {
System.out.println("Sub method");
}
}
public class Test {
public static void main(String[] args) {
Super s = new Sub();
s.method();
}
}
```

What will be the output of this code, and why?

Compile error.
Method in subclass can't override method in super class because method is private.

14. Given:

```
class Base {
static void show() {
System.out.println("Base show");
}
}
class Derived extends Base {
static void show() {
System.out.println("Derived show");
}
}
public class Test {
public static void main(String[] args) {
Base.show();
Derived.show();
}
}
```

What will be the output of this code, and why?

Output > Base show
Derived show

Derived extends Base(Derived is a Base) both of them have a static method named show so we do not have to create an object(Base.show; / Derived.show;) to call the method show.

15. Given:

```
class Base {
private int value = 100;
private void showValue() {
System.out.println("Value in Base: " + value);
}
}
class Derived extends Base {
void accessBase() {
System.out.println(value); // Error
showValue(); // Error
}
}
public class Test {
public static void main(String[] args) {
Derived d = new Derived();
d.accessBase();
}
}
```

Explain why the commented-out lines in Derived class cause errors.

Because we can not access private attributes of a superclass through a subclass.

16. Which statement(s) are true?

a. Has-a relationships always rely on inheritance.
false
b. Has-a relationships always rely on instance variables.
true
c. Has-a relationships always require at least two class types.
true
d. Has-a relationships always rely on polymorphism.
false
e. Has-a relationships are always tightly coupled.
false

17. Which of the following statements are true?

a. Inheritance reduces program-development time.
true
b. Java does not support multiple inheritances.
true
c. In single inheritance, a class is derived from one direct super class.
true
d. A subclass is more specific than its superclass and represents a smaller group of objects.
true
e. In an 'is-a' relationship, an object of a subclass also can be treated as an object of its superclass.
true
f. A 'has-a' relationship represents composition. In a 'has-a' relationship, a class object contains references to objects of other classes.
true

18. Which of the following are true statements?

A. The relationship between a class and its super class is an example of a "has-a" Relationship.
false
B. The relationship between a class and its super class is an example of an "is-a" Relationship.
true
C. The relationship between a class and field within the class is an example of a "has-a" relationship.
true
D. The relationship between a class and a field within the class is an example of an "is-a" relationship.
false

19. Which of the following statements describe the relationship between Superclasses and Subclasses?

A. A subclass cannot access the private members of its super class, but it can access the non-private members.
B. Superclass constructors are not inherited by subclasses.
C. A subclass can invoke a constructor of its superclass by using the keyword super, followed by a set of parentheses containing the superclass constructor

arguments. This must appear as the first statement in the subclass constructor's body.
D. A superclass method can be overridden in a subclass to declare an appropriate implementation for the subclass.
E. When a subclass redefines a superclass method by using the same signature, the subclass is said to overload that superclass method.

Answer > A
B
C
D

20. Given:

```
class Parent {
final void finalMethod() {
System.out.println("Final method in Parent");
}
}
class Child extends Parent {
void finalMethod() { // Error:
System.out.println("Trying to override final method");
}
}
public class Test {
public static void main(String[] args) {
Child c = new Child();
c.finalMethod();
}
}
```

Explain why the commented-out lines in Derived class cause errors.

We can't access , manipulate constant methods or constant attributes of a superclass through a subclass.

21. Which of the following lines can be inserted at line 12 still code will compile? Explain your answer.

```
class Super{
int a=10;
static int x=100;
Super(){
System.out.println("Super()");
}
Super(int i){
System.out.println("Super(int)");
```

```
}
}
class Sub extends Super{
int b=20;
static int y=200;
Sub(int i){
//Insert code Line 12
System.out.println("Sub(int)");
}
}
```

A. super(a); B. super(b); C. super(x);
D. super(y); E. super(i); F. super(100);

Answer > C. super(x); // x is static so class can access it before the instance is created
D. super(y); // y is static so class can access it before the instance is created
E. super(i); // i is a valid argument for int i parameter
F. super(100); // 100 is a valid argument for int i parameter.

22. How do access modifiers affect inheritance in Java? Discuss the access levels of methods and fields in the context of inheritance with example code.

We can't access private attributes or private methods of a superclass through a subclass.
We can access protected attributes or protected methods of a superclass through a subclass.
We can access public attributes or public methods of a superclass through a subclass.
We can't access default attributes or default methods of a superclass through a subclass.

```
class Vehicle{
        private void park(){
        System.out.println("Vehicle Parking");
        }

        public void callPark(){
        park();
        }
}

class Car extends Vehicle{
        public void park(){
        System.out.println("Car Parking");
        }
}

class Main5{
        public static void main(String[] args) {
```

```
                Vehicle v1 = new Car();
                v1.callPark();
        }
}
```

23. How does method overloading interact with inheritance in Java? Can overloaded methods be inherited? Provide an example.

A subclass can inherit overloaded methods of a super class.
Also we can overload methods that belong to the superclass within the subclass.

```
class SuperClass {
        void display(int a) {
        System.out.println("SuperClass: display(int)");
        }

        void display(double a) {
        System.out.println("SuperClass: display(double)");
        }
}

class SubClass extends SuperClass {

        void display(String a) {
        System.out.println("SubClass: display(String)");
        }

        void test() {
                display(5);
                display(5.5);
                display("Hello");
        }
}

public class Main5 {
        public static void main(String[] args) {
                SubClass sub = new SubClass();
                sub.test();
        }
}
```

24. Explain the role of the default constructor in inheritance. What happens if a class does not explicitly define any constructors?

Default constructor is created by the compiler when we do not put a constructor intentionally when creating a class.It is created with the null values.
When a subclass is initiated , normally the default constructor is called.
But if the superclass does not have a default constructor we should use super keyword and call a specific constructor when initiating the sub class.

25. Given:

```
class A{
void m(A a){
System.out.print("A");
}
}
class B extends A{
void m(B b){
System.out.print("B");
}
}
class C extends B{
void m(C c){
System.out.print("C");
}
}
class D{
public static void main(String args[]){
A a=new A();
B b=new B();
C c=new C();
c.m(a);
c.m(b);
c.m(c);
}
}
```

What is the result of attempting to compile and run the program?

a. ABC
b. ACC
c. AAA
d. BCC
e. BBC
f. Compiler error

g. None of the above

Answer > a. ABC

26. Given:

```
class Customer{
String name="Danapala";
static String city="Galle";
public Customer(){
printCustomer();
}
void printCustomer(){
System.out.println("Super : "+name+" -> "+city);
}
}
class RegularCustomer extends Customer{
String name="Somapala";
static String city="Panadura";
void printCustomer(){
System.out.println("Sub : "+name+" -> "+city);
}
}
class Demo{
public static void main(String arg[]){
new RegularCustomer();
}
}
```

What is the output? Explain your answer:
a. Prints Customer : Danapala -> Galle
b. Prints Customer : null -> Galle
c. Prints Regular Customer : null -> Panadura
d. Prints Sub : null -> Panadura

Answer > d. Prints Sub : null -> Panadura

The output will be from the printCustomer() method in RegularCustomer, but name will be null because it has not been initialized when printCustomer() is called from the Customer constructor.

27. Given the following class definition, which of the following can be legally placed after the comment line?

```
//Here ?
class Base{
```

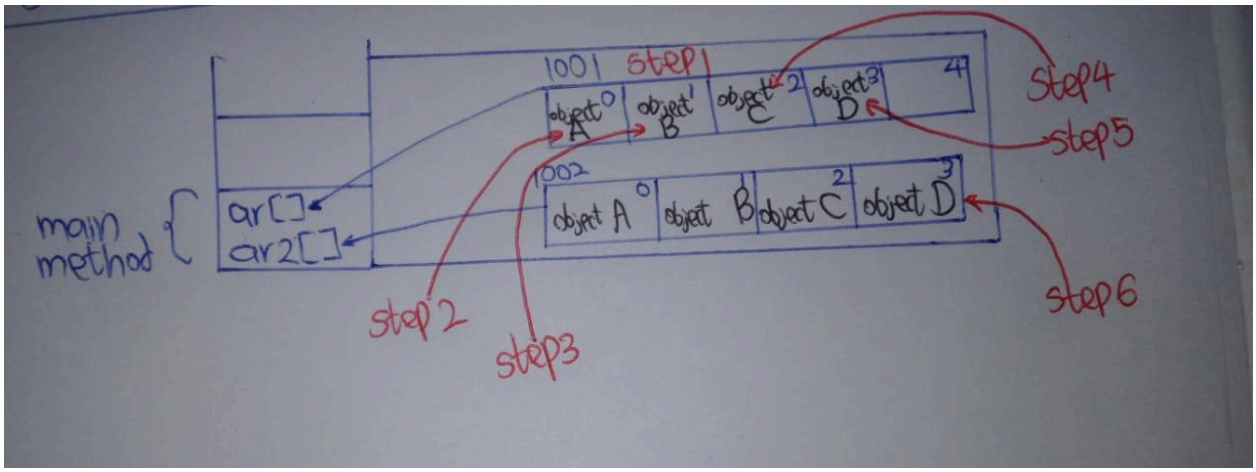
```
public Base(int i){
}
}
class MyOver extends Base{
public static void main(String arg[]){
MyOver m = new MyOver(10);
}
MyOver(int i){
super(i);

}
MyOver(String s, int i){
this(i);
//Here

}
}
```

a. MyOver m = new MyOver();// no suitable constructors
b. super(); // must be first statement in the constructor , no arguments
c. this("Hello",10); // must be first statement in the constructor
d. Base b = new Base(10);

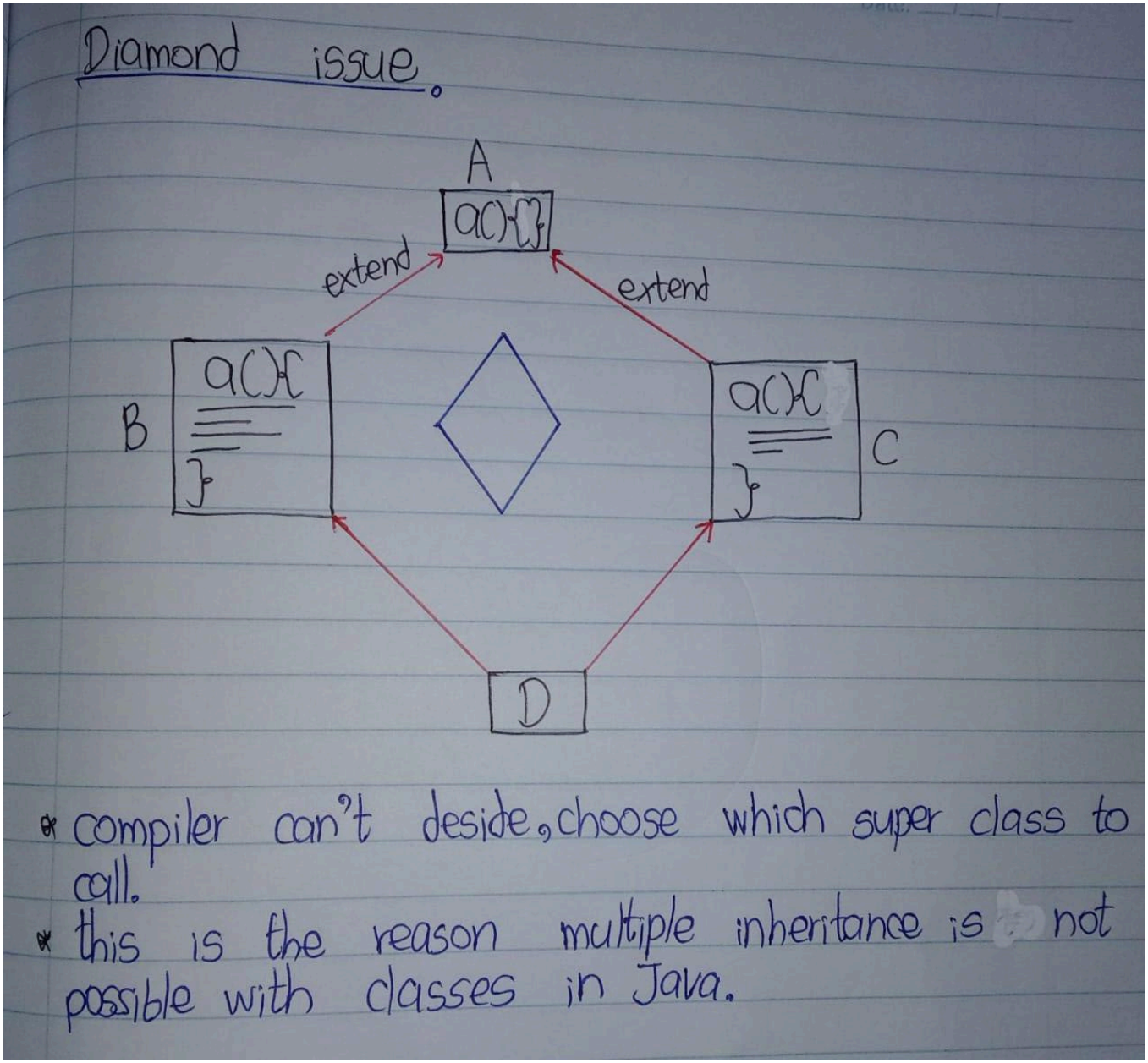
Answer > d. Base b = new Base(10);
Answer d line creates a new Base object using its constructor. This line can be put after this(i); in the constructor because it doesn't break any rules. It creates an object and doesn't interfere with how the constructors need to start with super() or this().

28. Draw object diagrams for the following steps:

```
class A{}
class B extends A{}
class C extends B{}
class D extends B{}
class Demo{
public static void main(String args[]){
A[] ar=new A[5];//Step 1
ar[0]=new A(); //Step 2
ar[1]=new B(); //Step 3
ar[2]=new C(); //Step 4
ar[3]=new D();//Step 5
A[] ar2={new A(),new B(), new C(),new D()}; //St6
}
}
```



Explain the Diamond Problem in the C


Diamond issue.
A
a()
extend
extend
B
a()
C
a()
D
* compiler can't deside, choose which super class to call.
* this is the reason multiple inheritance is not possible with classes in Java.