//

IJSE

Institute of Software Engin

Graduate Diploma in Software Engineering

Batch - GDSE69

Module - Object Oriented Programming

Name : Shimara Appuhami

Nic : 200362310434

///
/

Assignment 02

01. Considering the class diagram given bellow, identify classes, super classes, sub classes and implement them using java OOP.

- A is a superclass
- B and C are subclasses of A
- D and E are subclasses of C

```
class A{}

class B extends A{}
class C extends A{}
class D extends C{}
class E extends C{}
class oop {
   public static void main(String args[]){
   E b1 =new E();

    }
}
```

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

- Inheritance is a mechanism in which one class acquires the property of another class.

Eg -: child inherits the traits of his/her parents.
car, bus, bike – all of these come under a broader category called Vehicle.

03. Describe the ways in which inheritance promotes software reuse, saves time during program development and helps prevent errors.

Code Reusability:

- Inheritance allows you to create a new class by reusing and extending the functionality of an existing class. The existing class (superclass or base class) serves as a template, and the new class (subclass or derived class) inherits its attributes and methods.

Time Savings:

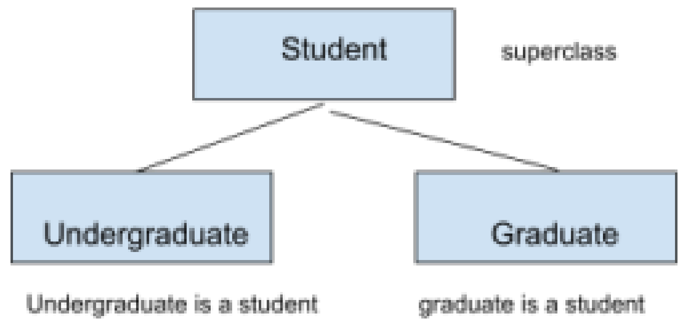
- By inheriting from existing classes, developers can leverage pre-existing code and focus on extending or customising the functionality rather than starting from scratch.

Prevention of Redundant Code:

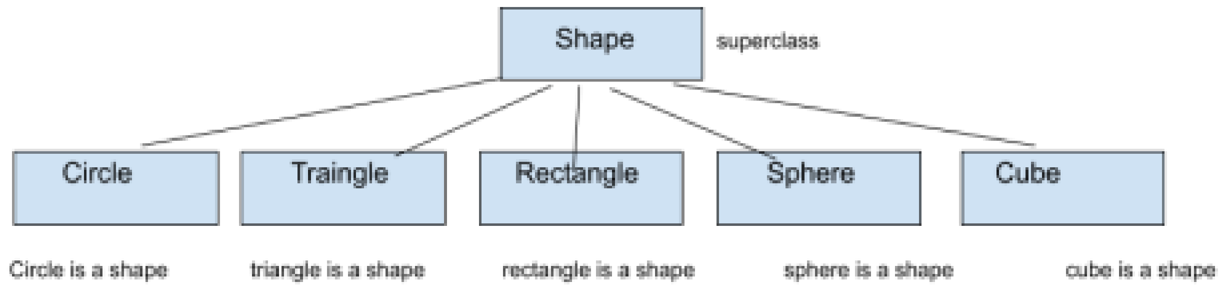
- Without inheritance, similar classes might need to duplicate common functionality, leading to redundant code. This redundancy can introduce inconsistencies and increase the likelihood of errors during maintenance or updates.

//

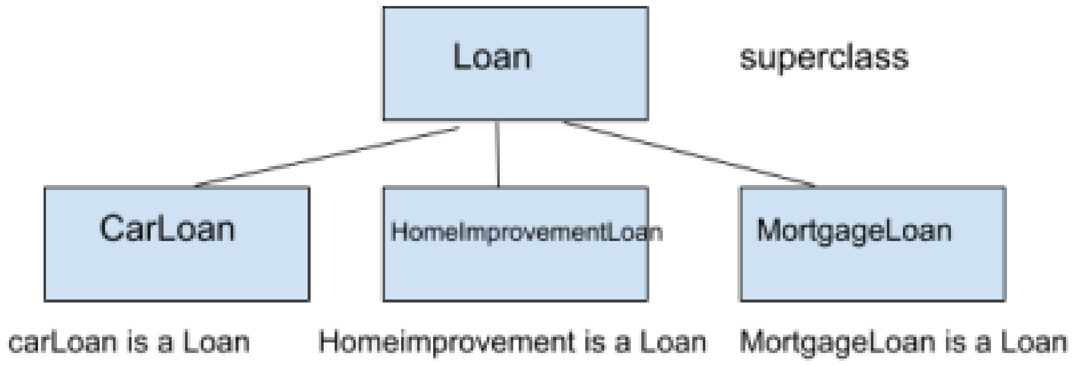
04. Create java class hierarchies for following cases:



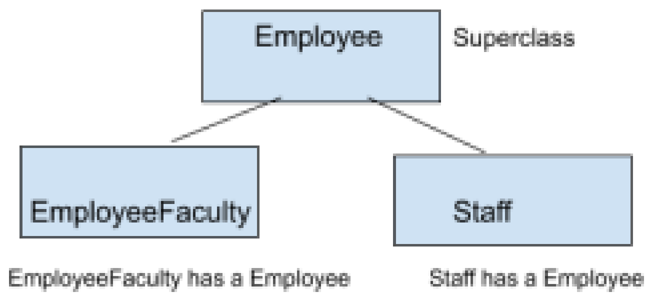
==



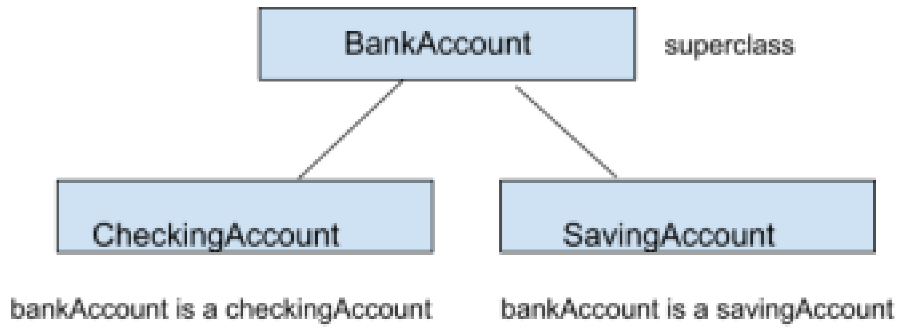
===



===



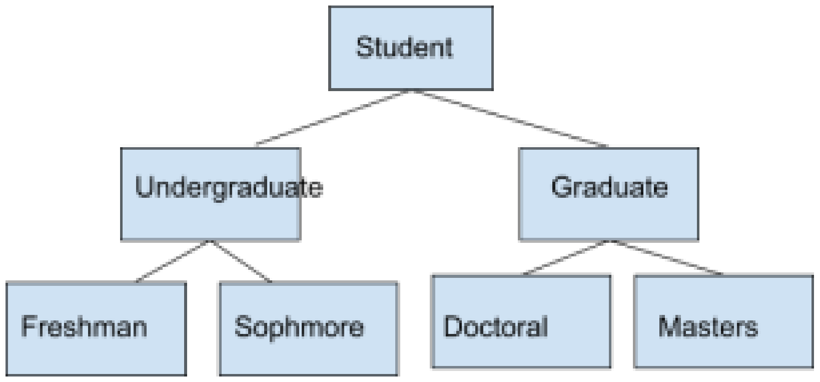
==


BankAccount
superclass
CheckingAccount
SavingAccount
bankAccount is a checkingAccount
bankAccount is a savingAccount

05. Expand the class "Shape" of exercise 04 with the following class hierarchy.

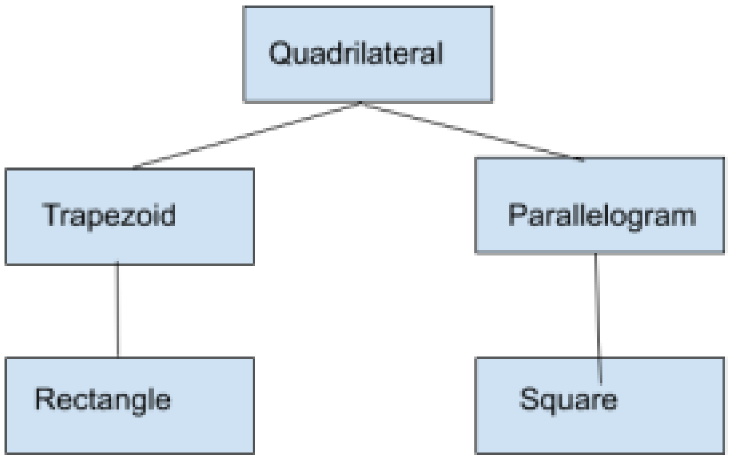
- the Cylinder and Pyramid subclasses each with their specific attributes and methods. The draw method is overridden in each subclass to provide specific drawing behaviour for each shape. The ShapeDemo class demonstrates creating instances of each shape and calling the draw method to show polymorphic behaviour.

06. Draw an inheritance hierarchy for students at a university. Use Student as the superclass of the hierarchy, then extend Student with classes UndergraduateStudent and GraduateStudent. Continue to extend the hierarchy as deep (i.e.,as many levels) as possible. For example, Freshman, Sophomore, Junior and Senior might extend UndergraduateStudent, and DoctoralStudent and MastersStudent might be subclasses of GraduateStudent. After drawing the hierarchy, implements all classes with suitable attributes and Methods.



//

07. 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).



// 08.
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.

- The CalculatorView class inherits various methods and attributes from the JFrame class, such as setSize, setTitle, setDefaultCloseOperation. This reuse of code allows the developer to leverage the existing functionality provided by the JFrame class without having to rewrite it.

//

09. Which of the following statements are true?
A. Inheritance reduces program-development time.
B. Java does not support multiple inheritances.
C. In single inheritance, a class is derived from one direct super class.
D. A subclass is more specific than its superclass and represents a smaller group of object.
E. In an ‘is-a’ relationship, an object of a subclass also can be treated as an object of its superclass.
F. A ‘has-a’ relationship represents composition. In a ‘has-a’ relationship, a class object contains references to objects of other classes.

- All the statements are true

//

10. Which of these code lines will cause compile error?

Explain your answer.

```
//-----------------------A.java-------------------------------
class A{
int a;
void printA(){
System.out.println("a : "+a);
}
}
// B.java
class B{
int b;
void printB(){
System.out.println("b : "+b); //Line 1
}
void callPrintAB(){
printA(); //Line 2
printB(); //Line 3
}
void printAB(){
System.out.println("a : "+a); //Line 4
System.out.println("b : "+b); //Line 5
}
}
```

- Line 2 >>>The printA method is not defined in the class B. It seems like it's intended to call the printA method from class A.
- Line 4>>>The variable a is not defined in class B.

//

11. 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.
B. The relationship between a class and its super class is an example of an "is-a" relationship.
C. The relationship between a class and field within the class is an example of a "has-a" relationship.
D. The relationship between a class and a field within the class is an example of an "is-a" Relationship.

//

12. Which the following statements describe relationship between Super classes 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.

//

13. Given:

```
class Super{
public void aMethod(int a){}
}
class Sub extends Super{
//Line 12
}
```

Which statement(s), inserted at line 12, will compile?
A. public void aMethod(int a){}
B. public void aMethod(int data){}
C. public void aMethod(double a){}
D. public void aMethod(int[] a){}

//

14. What will happen when you compile and run the program? Explain your answer.

```
class A{
A(int i){}
}
class B extends A{}
class Demo{
public static void main(String []args){
new B();
}
}
```

- error>>>A cannot be applied to given types;
- The issue arises because class B does not have a constructor, and there is no default constructor in class A.

//

15. Which of these code lines will cause compile error?
Explain your answer.

```
class A{}
class B extends A{}
class C extends A{}
class D extends B{}
class Demo{
public static void main(String args[]){
A a1=new A(); //Line 1
B b1=new B(); //Line 2
C c1=new C(); //Line 3
D d1=new D(); //Line 4
A a2=a1; //Line 5
a2=b1; //Line 6
a2=c1; //Line 7
a2=d1; //Line 8
B b2=a1; //Line 9
b2=c1; //Line 10
b2=d1; //Line 11
C c2=a1; //Line 12
c2=b1; //Line 13
c2=d1; //Line 14
D d2=a1; //Line 15
d2=b1; //Line 16
d2=c1; //Line 17
}
}
```

- Line 9>>>Trying to assign an object of type A to a reference of type B. ● Line 10>>>Trying to assign an object of type C to a reference of type B. ● Line 11>>>Trying to assign an object of type D to a reference of type B. ● Line 12>>>Trying to assign an object of type A to a reference of type C. ● Line 13>>>Trying to assign an object of type B to a reference of type C. ● Line 14>>>Trying to assign an object of type D to a reference of type C. ● Line 15>>>Trying to assign an object of type A to a reference of type D. ● Line 16>>>Trying to assign an object of type B to a reference of type D. ● Line 17>>>Trying to assign an object of type C to a reference of type D.

//

16. Which of the following most closely describes the process of overriding?
A. A class with the same name replaces the functionality of a class defined earlier in the hierarchy
B. A method with the same name completely replaces the functionality of a method earlier in the hierarchy
C. A method with the same name but different parameters gives multiple uses for the same method name
D. A class is prevented from accessing methods in its immediate ancestor.

//

17. Given:
```
class Animal{}
class Shape{}
//Line 12
```
Which statement(s), inserted at line 12, will compile?
A. class Animal extends Shape{}
B. class Shape extends Animal{}
C. class Lion extends Animal, Shape{}
D. class AnimalShape{}

//

18. Given:
```
class A{
public void aMethod(int a){
System.out.println("aMethod of A");
}
}
class B extends A{
public void aMethod(int a){
System.out.println("aMethod of B");
}
}
class C extends B{
public void aMethod(int a){
System.out.println("aMethod of C");
}
}
class Demo{
public static void main(String args[]){
A a1=new A();
A a2=new B();
A a3=new C();
B b1=new B();
B b2=new C();
C c1=new C();
//Line 12
}
}
```

Which of the following can be inserted at line 12 to prove the mechanism of “dynamic method dispatch” in OOP.

A. a1.myMethod(100); B. a2.myMethod(100);
C. a3.myMethod(100); D. b1.myMethod(100);
E. b2.myMethod(100); B. c1.myMethod(100);

//

19. What will happen when you compile and run the following program? Explain your answer.

```
class Super{
public Super(int i){
System.out.println("Super(int)");
}
}
class Sub extends Super{
public Sub(int i){
System.out.println("Sub(int)");
}
}
class Demo{
public static void main(String []args){
Sub s1=new Sub(100);
}
}
```

- When an object of class Sub (s1) is created in the main method with the statement Sub s1 = new Sub(100);, the following sequence of events occurs:
- The constructor of class Sub is invoked, which prints "Sub(int)" due to the statement

System.out.println("Sub(int)");.

- The constructor of the superclass Super is implicitly called with super(i). This calls the constructor public Super(int i) and prints "Super(int)".

//

20. Which of 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);

//

21. Describe the mechanism of invoking super class constructor at a sub class with appropriate example.

- class Vehicle {
- private String make;
- private String model;

```
●
● public Vehicle(String make, String model) {
● this.make = make;
● this.model = model;
● System.out.println("Vehicle constructor invoked. Make: " + make + ", Model: " + model);
● }
●
● public void start() {
● System.out.println("Vehicle is starting.");
● }
● }
●
● class Car extends Vehicle {
● private int year;
● private int numberOfDoors;
●
● public Car(String make, String model, int year, int numberOfDoors) { ●
super(make, model);
●
● this.year = year;
● this.numberOfDoors = numberOfDoors;
● System.out.println("Car constructor invoked. Year: " + year + ", Number of Doors: " +
  numberOfDoors);
● }
●
● public void drive() {
● System.out.println("Car is driving.");
● }
● }
●
● public class InheritanceExample {
● public static void main(String[] args) {
● Car myCar = new Car("Toyota", "Camry", 2022, 4);
●
● myCar.start();
● myCar.drive();
● }
● }
```

1. The Vehicle class has a parameterized constructor that takes a make and a model and initializes the corresponding attributes.
2. The Car class extends Vehicle and has its own constructor that takes additional

parameters (year and numberOfDoors).
3. Inside the Car constructor, super(make, model) is used to invoke the constructor of the Vehicle superclass. This ensures that the Vehicle constructor is executed before the Car constructor.
4. The Car constructor then initializes its own attributes (year and numberOfDoors) and prints a message specific to the Car class.

//

22. “Attributes in a class doesn’t override, they will hide in

the subclass”, Explain this using appropriate examples.

- The Student class has a variable name and a method to write.
- The graduate class extends Student and has its own variable name and a method write that overrides the write method in the superclass.

When you create an instance of graduate and access the name variable or the write method, the variables and methods in the subclass take precedence over those in the superclass.

//

23. Assume that class A extends class B, which extends
class C. Also all the three classes implement the method
test(). How can a method in class A invoke the test()
method defined in class C?
Select the one correct answer.
A. test();
B. super.test();
C. super.super.test();
D. ::test();
E. C.test();
F. It is not possible to invoke test()
G. method defined in C from a method in A.

//

24. Given:
class Mixer {
Mixer() { }
Mixer(Mixer m) {

```
 m1 = m;
}
Mixer m1;
public static void main(String[] args) {
Mixer m2 = new Mixer();
Mixer m3 = new Mixer(m2); m3.go();
Mixer m4 = m3.m1; m4.go();
Mixer m5 = m2.m1; m5.go();
}
void go() {
System.out.print("hi ");
}
}
```

What is the result?

A. hi

B. hi hi

C. hi hihi

D. Compilation fails

E. hi, followed by an exception

F. hi hi, followed by an exception

//

25. 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
}
}
```

//

26. Describe “Runtime Polymorphism” using the following example.

```
class Figure {
double dim1;
double dim2;
Figure(double a, double b) {
dim1 = a;
dim2 = b;
}
double area() {
System.out.println("undefined");
return 0;
}
}
class Rectangle extends Figure {
Rectangle(double a, double b) {
super(a, b);
}
// override area for rectangle
double area() {
System.out.println("Inside Area for Rectangle.");
return dim1 * dim2;
}
}
class Triangle extends Figure {
Triangle(double a, double b) {
super(a, b);
}
// override area for right triangle
double area() {
System.out.println("Inside Area for Triangle.");
return dim1 * dim2 / 2;
}
}
class FindAreas {
public static void main(String args[]) {
Figure f = new Figure(10, 10);
Rectangle r = new Rectangle(9, 5);
Triangle t = new Triangle(10, 8);
Figure figref;
figref = r;
System.out.println("Area is " + figref.area());
figref = t;
System.out.println("Area is " + figref.area());
figref = f;
System.out.println("Area is " + figref.area());
}
}
```

- an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.
- Three classes (Figure, Rectangle, and Triangle) form an inheritance hierarchy. ● The area() method is overridden in the subclasses (Rectangle and Triangle). ● In the main method, objects of types Figure, Rectangle, and Triangle are created. ● The reference figref of type Figure is used to point to objects of types Rectangle, Triangle, and Figure in different steps.
- The area() method is called using the figref, and the actual method executed is determined dynamically at runtime based on the actual type of the object referred to by figref.

//

27. Given:

```
class A{
static{System.out.print("A ");}
}
class B extends A{
static{System.out.print("B ");}
}
class C extends B{
static{System.out.print("C ");}
}
class D extends B{
static{System.out.print("D ");}
}
class E extends C{
static{System.out.print("E ");}
}
class Demo{
public static void main (String[] args) {
new D();new F();
}
}
```

What is the output?

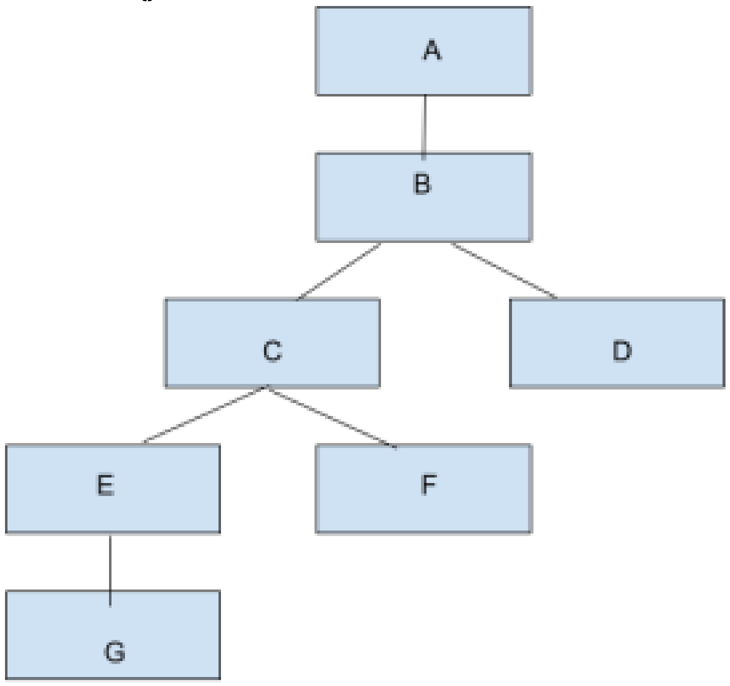
A. A B D A B C E

B. A B C D E

C. A B D C F

D. A B D

28. Draw a class Diagram for the following classes.
class A{}
class B extends A{}
class C extends B{}
class D extends B{}
class E extends C{}
class F extends C{}
class G extends E{}



29. You are given a class hierarchy with an instance of the class Dog. The class Dog is a child of Mammal and the class Mammal is a child of the class Vertebrate. The class Vertebrate has a method called move which prints out the string "move". The class Mammal overrides this method and prints out the string "walks". The class Dog overrides this method and prints out the string "walks on paws". Given an instance of the class Dog, how can you access the ancestor method move in Vertebrate so it prints out the string "move";
A. d.super().super().move();
B. d.parent().parent().move();
C. d.move();

D. none of the above;

//

30. Which of the following statements are true?
A. A static method is also known as a class method.
B. A class method is not associated with a particular instance of the class.
C. The keyword "this" cannot be used inside the body of a static method.
D. The keyword "super" may be used in the body of a static method

//

31. Assume that the superclass constructor invocation statement ‘super’ appears explicitly in a subclass constructor. If a compile-time error is to be avoided then the arguments for the superclass constructor invocation statement, ‘super’, cannot refer to which of the following?
A. Static variables declared in this class or any Superclass.
B. Instance variables declared in this class or any Superclass.
C. Static methods declared in this class or any superclass.
D. Instance methods declared in this class or any superclass.
E. The keyword "this".
F. The keyword "super".

//

32.Given the following class definition, which of the following 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();
B. super();
C. this("Hello",10);
D. Base b = new Base(10);
//

33. 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

///

34. Which statement(s) are true?
A. Has-a relationships always rely on inheritance.
B. Has-a relationships always rely on instance variables.
C. Has-a relationships always require at least two class types.
D. Has-a relationships always rely on polymorphism.
E. Has-a relationships are always tightly coupled.

///

35. Create class “CustomerStack”

```
class Customer{
private int code;
private String name;
public Customer(int code, String name){
this.code=code;this.name=name;
}
}
class Demo{
public static void main(String args[]){
CustomerStack stack=new CustomerStack();
stack.push(new Customer(1001,"Danapala"));
stack.push(new Customer(1002,"Gunapala"));
stack.push(new Customer(1003,"Somapala"));
stack.push(new Customer(1004,"Siripala"));
stack.printCustomerStack();
//[1004-Siripala, 1003-Gunapala, 1002-Gunapala, 1001-Danapala]
stack.pop();
stack.printCustomerStack();
//[1004-Siripala, 1003-Gunapala, 1002-Gunapala, 1001-Danapala]
}
}
```

```java
class Customer {
   private int code;
   private String name;

   public Customer(int code, String name) {
       this.code = code;
       this.name = name;
    }

   @Override
   public String toString() {
       return code + "-" + name;
    }
}

class CustomerStack {
   private Customer[] stack;
   private int top;

   public CustomerStack(int maxSize) {
       stack = new Customer[maxSize];
       top = -1;
    }

   public void push(Customer customer) {
       if (top < stack.length - 1) {
           stack[++top] = customer;
        }
    }

   public void pop() {
       if (top >= 0) {
           top--;
        }
    }

   public void printCustomerStack() {
       System.out.print("[");
       for (int i = top; i >= 0; i--) {
            System.out.print(stack[i]);
            if (i > 0) {
                System.out.print(", ");
            }
        }
        System.out.println("]");
    }
}

class oop {
   public static void main(String args[]) {
       CustomerStack stack = new CustomerStack(10);
       stack.push(new Customer(1001, "Danapala"));
       stack.push(new Customer(1002, "Gunapala"));
       stack.push(new Customer(1003, "Somapala"));
       stack.push(new Customer(1004, "Siripala"));
       stack.printCustomerStack();

       stack.pop();
    }
}
```

//

36. Create class “VehicleQueue”

```
class Demo{
public static void main(String args[]){
VehicleQueue queue=new VehicleQueue();
queue.enQueue(new Car("C001"));
queue.enQueue(new Bus("B001"));
queue.enQueue(new Bus("B002"));
queue.enQueue(new Car("C002"));
queue.enQueue(new Car("C003"));
queue.enQueue(new Van("V001"));
queue.enQueue(new Car("V002"));
queue.enQueue(new Bus("B003"));
queue.printVehicleQueue();
//[C001, B001, B002, C002, C003, V001, V002, B003]
queue.callPark();
/* Car Parking C001
Bus Parking B001
Bus Parking B002
Car Parking C002
Car Parking C003
Van Parking V001
Van Parking V001
Bus Parking B003*/
queue.deQueue();
queue.printVehicleQueue();
//[B001, B002, C002, C003, V001, V002, B003]
}
}
class Vehicle {
   private String registrationNumber;

   public Vehicle(String registrationNumber) {
   this.registrationNumber = registrationNumber; }

   @Override
   public String toString() {
       return registrationNumber;
    }
}

class Car extends Vehicle {
   public Car(String registrationNumber) {
          super(registrationNumber);
    }
}

class Bus extends Vehicle {
   public Bus(String registrationNumber) {
          super(registrationNumber);
    }
}

class Van extends Vehicle {
    public Van(String registrationNumber) {
        super(registrationNumber);
    }
}

class VehicleQueue {
    private Vehicle[] queue;
    private int front;
    private int rear;
    private int maxSize;

    public VehicleQueue(int maxSize) {
        this.maxSize = maxSize;
        this.queue = new Vehicle[maxSize];
        this.front = -1;
        this.rear = -1;
    }

    public void enQueue(Vehicle vehicle) {
        if (rear == maxSize - 1) {
            System.out.println("Queue is full. Cannot enqueue " + vehicle);
        } else {
            if (front == -1) {
                front = 0;
            }
            queue[++rear] = vehicle;
        }
    }

    public void deQueue() {
        if (front == -1) {
            System.out.println("Queue is empty. Cannot dequeue.");
        } else {
            System.out.println("Dequeued: " + queue[front]);
            if (front == rear) {
                front = rear = -1;
            } else {
                front++;
            }
        }
    }

    public void printVehicleQueue() {
        System.out.print("[");
        for (int i = front; i <= rear; i++) {
            System.out.print(queue[i]);
            if (i < rear) {
                System.out.print(", ");
            }
        }
        System.out.println("]");
    }

    public void callPark() {
        for (int i = front; i <= rear; i++) {
            String vehicleType = "Unknown";
            if (queue[i] instanceof Car) {
                vehicleType = "Car";
            } else if (queue[i] instanceof Bus) {
                vehicleType = "Bus";
            } else if (queue[i] instanceof Van) {
                vehicleType = "Van";
            }
            System.out.println(vehicleType + " Parking " + queue[i]);
        }
    }
}

public class oop {
    public static void main(String args[]) {
        VehicleQueue queue = new VehicleQueue(10);
        queue.enQueue(new Car("C001"));
        queue.enQueue(new Bus("B001"));
        queue.enQueue(new Bus("B002"));
        queue.enQueue(new Car("C002"));
        queue.enQueue(new Car("C003"));
        queue.enQueue(new Van("V001"));
        queue.enQueue(new Car("V002"));
        queue.enQueue(new Bus("B003"));
        queue.printVehicleQueue();
        queue.callPark();
            queue.deQueue();
        queue.printVehicleQueue();
      }
}
```

//

37. 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
Prints Sub : null -> Panadura