Object Oriented Programming- Assignment 04

01. "Attributes in a class don't override, they will hide in the subclass", Explain this using appropriate examples.

```
class Parent {
   int x = 10;
}

class Child extends Parent {
   int x = 20;
}

public class Main {
   public static void main(String[] args) {
      Parent parentObj = new Parent();
      Child childObj = new Child();

      System.out.println(parentObj.x); // Output: 10
      System.out.println(childObj.x);  // Output: 20
   }
}
```

In this Java example, we have a Parent class with an attribute x set to 10, and a Child class that extends Parent and defines its own x attribute set to 20.

When we create instances of both classes (parentObj and childObj) and access the x attribute, we see that the value of x depends on the type of the reference variable used to access it. However, note that the actual type of the object determines which attribute is accessed at runtime.

Even though both Parent and Child classes have an attribute named x, they are distinct and separate attributes. When we access x through a Child object, it refers to the x attribute defined in the Child class, hiding the x attribute of the Parent class.

02. 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());
}
}
```

We create objects of type Rectangle, Triangle, and Figure.

We then assign these objects to a reference variable of type Figure, which is the superclass for both Rectangle and Triangle.

When we call the area() method using the figref reference variable, Java determines which area() method to execute based on the actual type of the object being referred to at runtime.

In the first call figref.area() with figref referencing a Rectangle object, Java resolves the call to the area() method in the Rectangle class, and the output reflects the calculation for the area of a rectangle.

Similarly, in the second call figref.area() with figref referencing a Triangle object, Java resolves the call to the area() method in the Triangle class, and the output reflects the calculation for the area of a triangle.

In the third call figref.area() with figref referencing a Figure object, Java resolves the call to the area() method in the Figure class, and the output is from the default area() method which prints "undefined" and returns 0.

03. 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]
}
}
import java.util.ArrayList;
import java.util.List;

class Customer {
    private int code;
    private String name;

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

    public int getCode() {
        return code;
    }

    public String getName() {
        return name;
    }
}
```

```
class CustomerStack {
    private List<Customer> stack;

    public CustomerStack() {
        stack = new ArrayList<>();
    }

    public void push(Customer customer) {
        stack.add(customer);
    }

    public void pop() {
        if (!stack.isEmpty()) {
            stack.remove(stack.size() - 1);
        }
    }

    public void printCustomerStack() {
        System.out.print("[");
        for (int i = stack.size() - 1; i >= 0; i--) {
            Customer customer = stack.get(i);
            System.out.print(customer.getCode() + "-" + customer.getName());
            if (i != 0) {
                System.out.print(", ");
            }
        }
        System.out.println("]");
    }
}

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-Somapala, 1002-Gunapala, 1001-Danapala]
        stack.pop();
        stack.printCustomerStack();
        //[1003-Somapala, 1002-Gunapala, 1001-Danapala]
    }
}
```

04. 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]
}
}
import java.util.LinkedList;
import java.util.Queue;

// Vehicle class (parent class)
class Vehicle {
    private String id;

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

    public String getId() {
        return id;
    }

    public String getType() {
        return "Vehicle";
    }
}

// Car class (subclass of Vehicle)
class Car extends Vehicle {
    public Car(String id) {
        super(id);
    }

    @Override
    public String getType() {
```

```
        return "Car";
    }
}

// Bus class (subclass of Vehicle)
class Bus extends Vehicle {
    public Bus(String id) {
        super(id);
    }

    @Override
    public String getType() {
        return "Bus";
    }
}

// Van class (subclass of Vehicle)
class Van extends Vehicle {
    public Van(String id) {
        super(id);
    }

    @Override
    public String getType() {
        return "Van";
    }
}

// VehicleQueue class
class VehicleQueue {
    private Queue<Vehicle> queue;

    public VehicleQueue() {
        queue = new LinkedList<>();
```

```
    }

    public void enQueue(Vehicle vehicle) {
        queue.add(vehicle);
    }

    public void deQueue() {
        queue.poll();
    }

    public void printVehicleQueue() {
        System.out.println(queue);
    }

    public void callPark() {
        for (Vehicle vehicle : queue) {
            if (vehicle instanceof Car) {
                System.out.println("Car Parking " + vehicle.getId());
            } else if (vehicle instanceof Bus) {
                System.out.println("Bus Parking " + vehicle.getId());
            } else if (vehicle instanceof Van) {
                System.out.println("Van Parking " + vehicle.getId());
            }
        }
    }
}

// Demo class
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
           Car Parking V002
           Bus Parking B003 */

        queue.deQueue();
        queue.printVehicleQueue();
        //[B001, B002, C002, C003, V001, V002, B003]
    }
}
```

05. Complete the following program to obtain outputs for the line 8 as "Customer code-1001"

```
class Customer{
int code;
Customer(int code){this.code=code;}
}
class Demo{
public static void main(String args[]){
Customer c1=new Customer(1001);
System.out.println(c1); //Line 8
}
}
```

```
class Customer {
   int code;

   Customer(int code) {
      this.code = code;
   }

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

class Demo {
   public static void main(String args[]) {
      Customer c1 = new Customer(1001);
      System.out.println(c1); //Line 8
   }
}
```

06. What are the reasons for line1, and line2 are compiled and line3 is a compile error of the following program?

```
class Customer{
int code;
Customer(int code){this.code=code;}
}
class Demo{
public static void main(String args[]){
Customer c1=new Customer(1001);
c1.hashCode(); //Line 1
c1.toString(); //Line 2
c1.myMethod(); //Line 3
}
```

}

The hashCode() method is a public method defined in the Object class, which is the superclass of all Java classes.

Since every class in Java implicitly extends the Object class, the hashCode() method is inherited by the Customer class.

Therefore, calling hashCode() on an object of the Customer class (c1) is valid and compiles successfully.

07. Insert codes to the class "Customer" to get the correct outputs for the following program.

```
class Customer{
private int code;
private String name;
Customer(int code, String name){
this.code=code;
this.name=name;
}
}
class Demo{
public static void main(String args[]){
Customer c1=new Customer(1001,"Danapala");
Customer c2=new Customer(1001,"Danapala");
Customer c3=new Customer(1002,"Gunapala");
System.out.println("Hashcode c1 : "+c1.hashCode()); //1001
System.out.println("Hashcode c2 : "+c2.hashCode()); //1001
System.out.println("Hashcode c3 : "+c3.hashCode()); //1002
}
}
```

```
class Customer {
    private int code;
```

```
    private String name;

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

    @Override
    public int hashCode() {
        return code;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null || getClass() != obj.getClass())
            return false;
        Customer customer = (Customer) obj;
        return code == customer.code && name.equals(customer.name);
    }
}
```

08. Which of the following lines are legal? Explain your answer.

```
import javax.swing.*;
import java.util.*;
class Super{}
class Sub extends Super{}
class Demo{
public static void main(String args[]){
Super sup;
Sub sub;
Super []supArray;
Object ob;
```

```
ob=new Super(); //Line 1
ob=new Sub(); //Line 2
ob=new Object(); //Line 3
ob=new Object[10]; //Line 4
sub=new Sub(); //Line 5
sup=new Sub(); //Line 6
sub=new Super(); //Line 7
sup=new Super[10]; //Line 8
sub=new Sub[10]; //Line 9
supArray=new Sub[10]; //Line 10
supArray=new Super[10]; //Line 11
ob=new Super[10]; //Line 12
}
}
```

ob=new Super(); (Line 1):

Legal. An object of type Super is assigned to a reference variable of type Object. Java allows assignment of a superclass object to a subclass reference.

ob=new Sub(); (Line 2):

Legal. An object of type Sub (which is a subclass of Super) is assigned to a reference variable of type Object. This is also valid due to polymorphism.

ob=new Object(); (Line 3):

Legal. An object of type Object is assigned to a reference variable of type Object. This is straightforward and allowed.

ob=new Object[10]; (Line 4):

Illegal. An array of type Object is assigned to a reference variable of type Object. Arrays in Java are covariant, meaning they only allow assignments to their declared type or its subtypes. Here, Object is not a subtype of Object[].

sub=new Sub(); (Line 5):

Legal. An object of type Sub is assigned to a reference variable of type Sub. This is valid because Sub is a subclass of Super.

sup=new Sub(); (Line 6):

Legal. An object of type Sub (a subclass of Super) is assigned to a reference variable of type Super. This is legal due to polymorphism.

sub=new Super(); (Line 7):

Illegal. An object of type Super cannot be assigned to a reference variable of type Sub. A superclass reference variable cannot point to a subclass object.

sup=new Super[10]; (Line 8):

Illegal. An array of type Super is assigned to a reference variable of type Super. However, it is trying to assign an array of Super objects, not a single Super object, which is invalid.

sub=new Sub[10]; (Line 9):

Illegal. An array of type Sub is assigned to a reference variable of type Sub[]. This is an attempt to create an array of Sub objects, not a single Sub object.

supArray=new Sub[10]; (Line 10):

Legal. An array of type Sub is assigned to a reference variable of type Super[]. This is legal because arrays in Java are covariant.

supArray=new Super[10]; (Line 11):

Legal. An array of type Super is assigned to a reference variable of type Super[]. This is legal because arrays in Java are covariant.

ob=new Super[10]; (Line 12):

Legal. An array of type Super is assigned to a reference variable of type Object. This is allowed in Java because arrays are treated as objects and can be assigned to an Object reference.

09. Which of the following lines are legal? Explain your answer

```
import javax.swing.*;
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(),new B(),new C()}; //Line 1
B[] br={new A(), new B(), new C()}; //Line 2
C[] cr={new C(),new D(), new B()}; //Line 3
Object[] ob={new A(), new D(), //Line 4
new String(),new JFrame()};


}
}
```

A[] ar={new A(),new B(),new C()}; (Line 1):

Legal. An array of type A is being assigned with instances of classes A, B, and C. Java allows polymorphic behavior in arrays, meaning you can store subclass instances in a superclass array.

B[] br={new A(), new B(), new C()}; (Line 2):

Illegal. In Java, arrays are covariant, meaning you can only assign arrays of a subtype to a reference variable of the supertype. Since A is a superclass of B, you can't assign A objects to a B[] array.

C[] cr={new C(),new D(), new B()}; (Line 3):

Illegal. Similar to Line 2, you can't assign objects of types C, D, and B to an array of type C[]. Only instances of C or its subclasses should be assigned to a C[] array.

Object[] ob={new A(), new D(), new String(), new JFrame()}; (Line 4):

Legal. An array of type Object is being assigned with instances of classes A, D, String, and JFrame. Since Object is the superclass of all Java classes, it can hold references to any object. This is a common use case when you want to create an array that can hold any type of object.

10. Complete the class "Vehicle" to get the output as follow

```
class Customer{
private int code;
private String name;
Customer(int code, String name){
this.name=name;
this.code=code;
}
}
```

```
class Demo{
public static void main(String args[]){
Customer c1=new Customer(1001, "Danapala");
Customer c2=new Customer(1002, "Gunapala");
System.out.println(c1); //[1001-Danapala]
System.out.println(c2); //[1002-Gunapala]
}
}


class Customer {
    private int code;
    private String name;

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

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

11. Given:

```
class Vehicle{
String getName() { return "Vehicle"; }
Vehicle getType() { return this; }
}
class Car extends Vehicle{
// insert code hereLine 6
```

}

class Toyota extends Car{ }

Which statement(s), inserted at line 6, will compile?

A. Vehicle getType() { return this; }

B. String getType() { return "this"; }

C. Car getType() { return this; }

D. Toyota getType() { return new Toyota(); }

12. If a base class has a method defined as

void method() { }.

Which of the following are legal prototypes in a derived class of this class? Select all correct answers.

A. void method() { }

B. int method() { return 0;}

C. void method(int i) { }

D. private void method() { }

13. Which of the following statements are true?

A. method cannot be overloaded to be less public in a child class

B. To be overridden a method must have the same name and parameter types

C. To be overridden a method must have the same name, parameter and return types

D. An overridden method must have the same name,

14. Given:

class Super{

//Insert Code here Line 2

}

class Sub extends Super{

void myMethod(){};

}

Which of the following code fragments could be inserted at line 2 and still allow the code to compile?

A. static void myMethod(){}

B. final void myMethod(){}

C. private void myMethod(){}

D. private static void myMethod(){}

E. private final void myMethod(){}

F. static void myMethod(int i){}

G. public void myMethod(){}

H. protected void myMethod(){}

15. Given:

1. class Plant {

2. String getName() { return "plant"; }

3. Plant getType() { return this; }

4. }

5. class Flower extends Plant {

6. // insert code here

7. }

8. class Tulip extends Flower { }

Which statement(s), inserted at line 6, will compile?

A. Flower getType() { return this; }

B. String getType() { return "this"; }

C. Plant getType() { return this; }

D. Tulip getType() { return new Tulip(); }

16. Which of the following statements are true?

A. A method cannot be overloaded to be less public in a child class

B. To be overridden a method must have the same name and parameter types

C. To be overridden a method must have the same name, parameter and return types

D. An overridden method must have the same name,

17. Which of the following statements are true?

A. A final method cannot be overridden.

B. All methods declared in a final class are implicitly final.

C. The methods declared in a final class must be explicitly declared final or a compile-time error occurs.

D. It is a compile-time error if a private method is declared final.

E. A machine code generator can inline the body of a final method.

F. None of the above.

18. Given:

```
1. class Programmer {
2. Programmer debug() { return this; }
3. }
4. class SCJP extends Programmer {
5. // insert code here
6. }
```

Which, inserted at line 5, will compile?

A. Programmer debug() { return this; }

B. SCJP debug() { return this; }

C. Object debug() { return this; }

D. int debug() { return 1; }

E. int debug(int x) { return 1; }

F. Object debug(int x) { return this; }

19. Given:

```
class G {
String s1 = "G.s1";
void printS1(){
System.out.print("G.printS1," + s1);
}
G() { printS1();}
}
class H extends G {
String s1 = "H.s1";
void printS1(){
System.out.print("H.printS1," + s1);
}
}
class Demo{
public static void main(String[] args) {
H h = new H();
}
}
```

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

A. Prints: G.printS1,G.s1 B. Prints: G.printS1,H.s1

C. Prints: G.printS1,null D. Prints: H.printS1,G.s1

E. Prints: H.printS1,H.s1 F. Prints: H.printS1,null

20. Given

```
class E{
void m(){
System.out.print("A"+" ");
}
static void m1(){
```

```
System.out.print("B"+" ");
}
}
class F extends E{
void m(){
System.out.print("AAA"+" ");
}
static void m1(){
System.out.print("BBB"+" ");
}
public static void main(String args[]){
E e=new F();
e.m();
e.m1();
}
}
```

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

A. Prints: AB B. Prints: AAAB

C. Prints: ABBB D. Prints: AAABBB

21. Given

```
class Super{
static int i=10; //Line 1
int j=20; //Line 2
void m1(){} //Line 3
static void m2(){} //Line 4
}
class Sub extends Super{
int i=5; //Line 5
static int j=10; //Line 6
static void m1(){} //Line 7
void m2(){} //Line 8
```

}

A. Line 1 B. Line 2

C. Line 3 D. Line 4

E. Line 5 F. Line 6

G. Line 7 H. Line 8

22. Given

```
class Account{
private int balance;
final int MAX;
Account(){
//Line 1
}
Account(int balance){
//Line 2
this.balance=balance;
//Line 3
}
static{
//Line 4
}

{
//Line 5
}
void setMax(int max){
//Line 6
}
}
```

Which the following code fragment(s) can be inserted at each line to remove the completion errors of the above program?

A. Replaces as ‘final int MAX=10000;’ at line 1

B. Inserts ‘MAX=1000;’ at Line 2

C. Inserts ‘MAX=1000;’ at Line 3

D. Inserts ‘MAX=1000;’ both at Line 2 and Line 3

E. Inserts ‘MAX=1000;’ both at Line 2 and Line 4

F. Inserts ‘MAX=1000;’ at Line 2 and inserts ‘this();’ at Line 3

G. Inserts ‘MAX=1000;’ at Line 2 and inserts ‘this();’ at Line 4

H. Inserts ‘MAX=1000;’ at Line 5

I. Inserts ‘MAX=1000;’ at Line 6

J. Inserts ‘MAX=1000;’ at Line 7

K. Inserts ‘MAX=1000;’ both at Line 5 and Line 2

23. Which of the following statements are true about a variable created with the static modifier?

A. Once assigned the value of a static variable may not be altered.

B. A static variable created in a method will keep the same value between calls.

C. Only one instance of a static variable will exist for any amount of class instances.

D. The static modifier can only be applied to a primitive value.

24. According to Question 22 if the final variable is static, what is your decision? Clearly explain your answer.

```
class Account{
static final int MAX; //Line 1
static int MIN; //Line 2
Account(){
//Line 3
}
Account(int balance){
//Line 4
```

```
}
static{
//Line 5
}
{

//Line 6
}
}
```

25. What is the output? Explain your answer.

```
class Super{
int a=10;
private int b=30;
final int c=30;
static int d=40;
final static int e=50;
void printValues(){
System.out.print("Super : ");
System.out.println(+a+" "+b+" "+c+" "+d+" "+e);
}
Super(){
printValues();
}
}
class Sub extends Super{
int a=100;
private int b=300;
final int c=300;
static int d=400;
final static int e=500;
void printValues(){
```

```
System.out.print("Sub : ");
System.out.println(+a+" "+b+" "+c+" "+d+" "+e);
}
}
class Demo{
public static void main(String args[]){
new Sub();
}
}
```

26. Keyword 'final' is checking only compile time, not runtime. Explain it using the following example.

```
class Super{
final static int a=10;
void printValues(){
System.out.print("Super : ");

System.out.println(a);
}
Super(){
printValues();
}
}
class Sub extends Super{
final int a;
Sub(){
super();
a=100;
}
void printValues(){
System.out.print("Sub : ");
System.out.println(a);
}
```

```
}
class Demo{
public static void main(String args[]){
Sub s1=new Sub();
System.out.println(s1.a);
}
}
```

27. Which of the following lines are legal, explain your answers.

```
class Stack{
}
class Queue<T>{
}
class Demo{
public static void main(String args[]){
Stack <String>strStack=new Stack<>(); //Line 1
Stack objStack=new Stack(); //Line 2
Queue <String>strQueue=new Queue<>(); //Line 3
Queue objQueue=new Queue(); //Line 4
}}
```