IJSE

Institute of Software Engineering

Comprehensive Master Java Developer

Batch - CMJD 103

Module – Object Oriented Programming

Name : Shimara Appuhami

Nic : 200362310434

Assignment 04

01. “Attributes in a class don't override, they will hide in the subclass”, Explain this using

appropriate examples.

- When a subclass defines a variable with the same name as a variable in its superclass, the variable in the subclass hides the variable in the superclass. This is called "variable hiding." The subclass does not override the variable; instead, it creates a new variable that coexists with the superclass's variable.

```
class SuperClass {
    int x = 10;
}
class SubClass extends SuperClass {
    int x = 20;
   void printX() {
        System.out.println("x in SubClass: " + x);
        System.out.println("x in SuperClass: " + super.x);
    }
}

public class Main {
    public static void main(String[] args) {
        SubClass obj = new SubClass();
        obj.printX();
    }
    }
```

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());
}
}
```

- Runtime polymorphism, also known as dynamic method dispatch, is the mechanism by which a call to an overridden method is resolved at runtime rather than at compile-time. This is achieved through method overriding and the use of a reference variable of a superclass type to refer to an object of a subclass type.

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]

}

}
```

- class CustomerStack {
- private Stack<Customer> stack = new Stack<>();
- public void push(Customer customer)
- {
- stack.push(customer);
- }
- public void pop() {
- if (!stack.isEmpty()) {
- stack.pop();
- }
- }

//

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]

}

}
```

- class VehicleQueue {
- private Queue<Vehicle> queue = new LinkedList<>();
- public void enQueue(Vehicle vehicle) {
- queue.add(vehicle);
- }
- public void deQueue() {
- if (!queue.isEmpty()) {
- queue.remove();
- }
- }

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);  // Output: Customer code-1001
•      }
•  }
```

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
}
}
```

- Line 1: hashCode() is a method of the Object class, which is the superclass of all Java classes. Since Customer is a subclass of Object, it inherits the hashCode() method.
- Line 2: toString() is also a method of the Object class. Similarly, Customer inherits the toString() method.
- Line 3: myMethod() is not defined in the Customer class or its superclass Object. Therefore, trying to call myMethod() on a Customer object results in a compile-time error.

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
}
}
```

- Hashcode c1 : 918221580
- Hashcode c2 : 468121027
- Hashcode c3 : 1804094807

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**. You can assign an instance of `Super` to a variable of type `Object`.

ob = new Sub(); // Line 2

- **Legal**. You can assign an instance of `Sub` (which extends `Super`) to a variable of type `Object`.

ob = new Object(); // Line 3

- **Legal**. You can assign an instance of `Object` to a variable of type `Object`.

ob = new Object[10]; // Line 4

- **Legal**. You can assign an array of `Object` to a variable of type `Object`.

sub = new Sub(); // Line 5

- **Legal**. You can assign an instance of `Sub` to a variable of type `Sub`.

sup = new Sub(); // Line 6

- **Legal**. You can assign an instance of `Sub` (which extends `Super`) to a variable of type `Super`.

sub = new Super(); // Line 7

- **Illegal**. You cannot assign an instance of `Super` to a variable of type `Sub` because `Super` is not a subclass of `Sub`.

sup = new Super[10]; // Line 8

- **Legal**. You can assign an array of `Super` to a variable of type `Super[]`.

sub = new Sub[10]; // Line 9

- **Illegal**. You cannot assign an array of `Sub` to a variable of type `Sub`. You would need to declare `sub` as `Sub[] sub`.

supArray = new Sub[10]; // Line 10

- **Legal**. You can assign an array of `Sub` (which is a subclass of `Super`) to a variable of type `Super[]`.

supArray = new Super[10]; // Line 11

- **Legal**. You can assign an array of `Super` to a variable of type `Super[]`.

ob = new Super[10]; // Line 12

- **Legal**. You can assign an array of `Super` to a variable of type `Object`.

//
//////////////

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**. This line is valid because `B` and `C` are subclasses of `A`. Thus, an array of `A` can hold instances of `A`, `B`, and `C`.
- **Object[] ob = {new A(), new D(), new String(), new JFrame()}; // Line 4**
  - **Legal**. This line is valid because all objects are subclasses of `Object`. Thus, an array of `Object` can hold instances of `A`, `D`, `String`, and `JFrame`.

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]
}
}
```

- Customer@36baf30c
- Customer@7a81197d

//

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

C. 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();
}
}
```

- Sub : 0 0 300 400 500

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);
}
}
```

- First printValues() call (from Super constructor): "Super : 10"
- Second printValues() call (from Sub instance method): "Sub : 100"
- Final System.out.println(s1.a);: Prints 100

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
}
}
```

- Line 2: Legal — Stack is a non-generic class, and you can instantiate it without type parameters.
- Line 3: Legal — Queue is a generic class, and specifying <String> is correct.

****