The LNM Institute of Information Technology

CSE

Advance Programming

Mid term Exam-2017-18

Time: 1.30 hrs **Date:**25/09/2017 **Max. Marks:** 30

Q1. You are asked to design a software system for the restaurant chain. The restaurant chain has three kinds of restaurants – Indian, Chinese and Italian. Each of them have menu that contain the names and prices of the dishes that they serve. Each restaurant has a waiter who gives the menu to the customers and takes the orders. For each order, a bill is generated according to the price mentioned in the menu. Identify the following:

1. Classes to be used**(5 marks)**
2. Class in which main method is present**(1 mark)**
3. The inheritance pattern**(2 marks)**
4. Which classes use the objects of which other classes as their attribute**(1 mark)**
5. Which classes use the objects of which other classes in their functions?**(1 mark)**

(space for answer1)

1. **Classes to be used (5 marks):**
   1. Restaurant_Chain
   2. Restaurant
   3. Indian_Restaurant
   4. Chinese_Restaurant
   5. Italian_Restaurant
   6. Dish
   7. Menu
   8. Waiter
   9. Order
   10. Bill
2. **Class in which main method is present(1 mark)**

   Restaurant_Chain

3. **The inheritance pattern (2 marks)**

   Indian_Restaurant, Chinese_Restaurant and Italian_Restaurant inherit from Restaurant

4. **Which classes use the objects of which other classes as their attribute (1 mark)**

   Menu contains objects of Dish

5. **Which classes use the objects of which other classes in their methods?(1 mark)**

   Waiter uses Order in its methods

   Order uses Menu in its methods

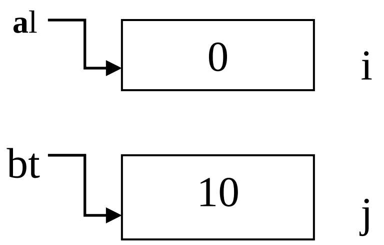
(Space for Answer1)

Q2. Correct the compile time error (if any) and find the output of the following Java codes:
(Also show the workout otherwise the answer will not be evaluated) [10]

Q1. [2]

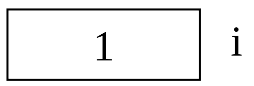
```
class Alpha
{     int i;
      public void display()
      {
         System.out.println(i);
      }
  }
class Beta extends Alpha
  {  int j;
      Beta()
      {
        j=10;
      }
     public void display()
      {
         System.out.println(super.i+this.j);
      }
  }
class RunPolymorphism
{  public static void main(String args[])
      {
         Alpha al = new Alpha();
         Beta  bt = new Beta();
         al.display();
         bt.i = 1;
         al = bt;
         al.display();
      }
  }
```

Show the workout here

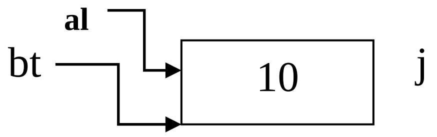


call to al.display(); will invoke the Alpha class (super class) display() so output: 0

bt.i = 1; due to this statement the value of i become 1.



al = bt; due to this statement object al (super class) will refer to sub class(Beta)



al.display(); call to this statement will invoke the sub class display() so

Output: (super.i+this.j) = 1+10 = 11

Q2. [2]

```
class Garbage
{   public static void main(String args[])
     {
       Garbage gb = new Garbage();
       h.methodA();
     }
    Object methodA()
     {   Object obj1 = new Object();
         Object obj2[] = new Object[1];
         obj2[0] = obj1;
         obj1 = null;
         return obj2[0];
   }
}
```

Show the workout here

Compile time error in: h.methodA();
So correction is : gb.methodA();

In this code we have to find how many objects will be eligible for garbage collection

1. **obj1 = null; this object obj1 should be eligible for garbage collection but right now referred by obj2[0].**
2. **return obj2[0]; obj2[0] is returned but not received by calling method main() so this object will also be eligible for garbage collection.**

Output: 2 (after completion of program)

Q3. **[2]**

```
class Equality
{
   int x;
   int y;
   boolean isequal(){
      return(x == y);
   }
}
class Output {
   public static void main(String args[])
   {
      equality obj = new equality();
      obj.x = 5;
      obj.y = 6;
      System.out.println(obj.isequal());
   }
}
```

Show the workout here

5	x
6	y

System.out.println(obj.isequal());// will call to isequal() method of class Equality.

```
boolean isequal()
{
   return(x == y);
}
```

Will return false because the value of x=5 and y=6;

Output: false

Q4. **[2+2]**

```
class Overload
  {     int x;
        double y;
         static int z=10;
     void add(double c , double d){
        x = (c+d)*z;
        y = (c + d)*x;
        z = x+y;
     }
     Overload() {
              this.y = 10.0;
               }
  }
  class Output {
     public static void main(String args[]){
        Overload obj = new Overload();
        Overload obj1 = new Overload();
        int a = 2; double b = 3.5;
        obj.add(a, a);
        obj1.add(b, b);
       System.out.println(obj.x + " " + obj.y);
       System.out.println(obj1.x + " " + obj1.y);
     }
 }
```

Show the workout here

Compile time error in the statements:

x = (c+d)*z;

z = x+y;

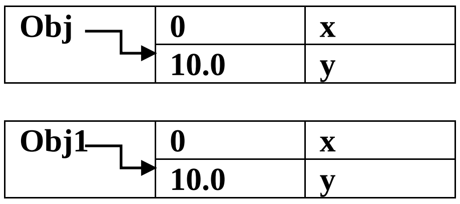
so proper casting is needed.

x = (int)(c+d)*z;

z = (int)(x+y);

$\boxed{10}$ **Z**

Two objects share the same z (static)



Two local variables:

a $\boxed{2}$ b $\boxed{3.5}$

now call to **obj.add(a, a);**

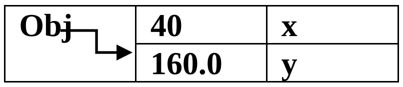
c=2.0 d =2.0

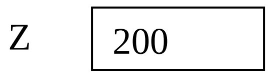
x = (2.0+2.0)*10 = 40.0 so **x=40**

y= (2.0+2.0)*40 = 160.0 so y= 160.0

z=(40+160.0) = 200.0 so z=200

after execution object **obj** become





Now call to **obj1.add(b, b);**

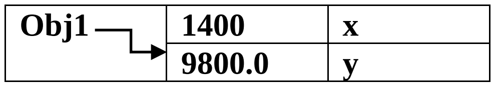
c=3.5 d =3.5

x = (3.5+3.5)*200 = 1400.0 so **x=1400**

y= (3.5+3.5)*1400 = 9800.0 so y= 9800.0

z=(1400+9800.0) =10 200.0 so z=10200

after execution object **obj1**become



Now call to

```
System.out.println(obj.x + " " + obj.y);
System.out.println(obj1.x + " " + obj1.y);
```

Output:
```
40 160.0
1400 9800.0
```

Q3. Differentiate the following: (Write precise and to the point) [5X2]

Top-down Programming

In this approach, programming begins from the top level of hierarchy and progresses towards the lower levels. The implementation of modules starts with the main module. After the implementation of the main module, the subordinate modules are implemented and the process follows in this way. In top-down programming, there is a risk of implementing data structures as the modules are dependent on each other and they nave to share one or more functions and procedures. In this way, the functions and procedures are globally visible. In addition to modules, the top-down programming uses sequences and the nested levels of commands.

Bottom-up Programming

The description begins at the bottom of the hierarchy of modules and progresses through higher levels until it reaches the top. It is easier to construct functions in bottom-up manner. This is because bottom-up programming requires a way of passing complicated arguments between functions. It takes the form of constructing abstract data types in languages such as C++ or Java, which can be used to implement an entire class of applications and not only the one that is to be written. It therefore becomes easier to add new features in a bottom-up approach than in a top-down programming approach.

2) System testing and User acceptance testing?

System Testing	Acceptance Testing
1. User(client) is not involved in System Testing.	1. User (client or customer) is completely involved in Acceptance Testing.
2. It is performed before Acceptance Testing.	2. It is performed after System Testing.
3. System Testing refers to the testing of complete system as a whole that is carried out by testers and sometimes by developers to check whether it meets the system specifications or not.	3. Acceptance Testing is carried out by the users (client) to determine whether they accept the delivery of the system or not.
4. System Testing determine the developer and tester for satisfaction with system specifications.	4. Acceptance Testing determine the customer for satisfaction with software product.

3) Abstraction and Encapsulation.

Both **Abstraction** and **Encapsulation** are two of the four basic OOP concepts which allow you to model real-world things into objects so that you can implement them in your program and code. In other words, Abstraction hides details at the **design** level, while Encapsulation hides details at the **implementation** level.

Encapsulation is the packing of *data* and *functions operating on that data* into a single component and restricting the access to some of the object's components..

Abstraction is a mechanism which represents the essential features without including implementation details.

Encapsulation: Information hiding.
Abstraction: Implementation hiding.

4) Static binding and Dynamic binding (in context to Function overloading and overriding)

Static binding is a binding which happens during compilation. It is also called early binding because binding happens before a program actually runs.

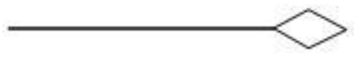
Dynamic binding is a binding which happens during run time. It is also called late binding because binding happens when program actually is running.

Basis for comparison	Static Binding	Dynamic Binding
Event Occurrence	Events occur at compile time are "Static Binding".	Events occur at run time are "Dynamic Binding".
Information	All information needed to call a function is known at compile time.	All information need to call a function come to know at run time.
Advantage	Efficiency.	Flexibility.
Time	Fast execution.	Slow execution.
Alternate name	Early Binding.	Late Binding.

5) Aggregation and Specialization (Inheritance).

Aggregation

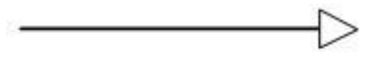
When an object **'has-a'** another object, then you have got an aggregation between them. Direction between them specified which object contains the other object..



Consider that A Course contains students. A student cannot exist without a Course.

Specialization

Specialization means creating new subclasses from an existing class**.** **Specialization** uses a “is-a” relationship from a specialization to the generalization class.



Consider there exists a class named Person. A student is a person. A faculty is a person. Therefore here the relationship between student, faculty and person is specialization.