OOP

FAHAM AKHTAR(15772)

6/29/2020

M.Ayub

Department of Computer Science

Final Term Exam Spring 2020

Subject: **Object Oriented Programming**

BS (CS,SE) Instructor: M.Ayub Khan

There are total **5** questions in this paper. Max Marks: 50

<u>Note:</u>

At the top of the answer sheet there must be the ID, Name and semester of the concerned Student.

Students must have to provide the output of their respective programs. Students have same answers or programs will be considered fail. Programs in Java or codes should be explained clearly.

As this paper is online so incase of any ambiguity my Whatsapp no. is 034499121116.

Each question carry equal marks.
Please answer briefly.

Q1. a. Why access modifiers are used in java, explain in detail Private and Default access modifiers?

ANS:

You must have seen public, private and protected keywords while practising java programs, these are called access modifiers. An access modifier restricts the access of a class, constructor, data member and method in another class. In java we have four access modifiers:

1. default
2. private
3. protected
4. public

1. Default access modifier

When we do not mention any access modifier, it is called default access modifier. The scope of this modifier is limited to the package only. This means that if we have a class with the default access modifier in a package, only those classes that are in this package can access this class. No other class outside this package can access this class. Similarly, if we have a default method or data member in a class, it would not be visible in the class of another package. Lets see an example to understand this:

Default Access Modifier Example in Java

In this example we have two classes, Test class is trying to access the default method of Addition class, since class Test belongs to a different package, this program would throw compilation error, because the scope of default modifier is limited to the same package in which it is declared.

```
package abcpackage;


public class Addition {

   /* Since we didn't mention any access modifier here, it would

    * be considered as default.

    */

   int addTwoNumbers(int a, int b){

            return a+b;

   }

}
```

Test.java

```
package xyzpackage;

import abcpackage.*;

public class Test {

   public static void main(String args[]){

            Addition obj = new Addition();

        /* It will throw error because we are trying to access

         * the default method in another package

         */

            obj.addTwoNumbers(10, 21);

   }

}
```

b. Write a specific program of the above mentioned access modifiers in java.

ANS:

program # 1

```
public class PrivateDefault{

    public static void main(String []args){
      PrivateEg privateObj = new PrivateEg();
      privateObj.setData(30);
      System.out.println("Output Private Object: " + privateObj.getData());
      System.out.println("------------------------");
      DefaultEg defaultObj = new DefaultEg();
      System.out.println("Setting data using default scope in same package");
      defaultObj.data = 20;
      defaultObj.output();
      System.out.println("------------------------");
      System.out.println("End of program");
    }
}

class PrivateEg{
   private int data;
   public void setData(int iData)  {
      System.out.println("Setting data for private class");
      data = iData;
   }
   public int getData(){
      return data;
   }
}

class DefaultEg{
   int data;
   void output(){System.out.println("Outut data from default class: " + data);}
}

/*Output*/
/*
Setting data for private class
Output Private Object: 30
------------------------
Setting data using default scope in same package
Output data from default class: 20
------------------------
End of program

*/
```

Q2. a. Explain in detail Public and Protected access modifiers?
ANS:
Private access modifier

The scope of private modifier is limited to the class only.

Private Data members and methods are only accessible within the class

Class and Interface cannot be declared as private

If a class has private constructor then you cannot create the object of that class from outside of the class.

Let's see an example to understand this:

Private access modifier example in java

This example throws compilation error because we are trying to access the private data member and method of class ABC in the class Example. The private data member and method are only accessible within the class.

```
class ABC{

  private double num = 100;

  private int square(int a){

        return a*a;

  }

}

public class Example{

  public static void main(String args[]){

        ABC obj = new ABC();

        System.out.println(obj.num);

        System.out.println(obj.square(10));

  }

}
```

3. Protected Access Modifier

Protected data member and method are only accessible by the classes of the same package and the subclasses present in any package. You can also say that the protected access modifier is similar to default access modifier with one exception that it has visibility in sub classes. Classes cannot be declared protected. This access modifier is generally used in a parent child relationship.

Protected access modifier example in Java

In this example the class Test which is present in another package is able to call the addTwoNumbers() method, which is declared protected. This is because the Test class extends class Addition and the protected modifier allows the access of protected members in subclasses (in any packages).

Addition.java

```
package abcpackage;

public class Addition {


   protected int addTwoNumbers(int a, int b){

            return a+b;

   }

}
```

4. Public access modifier

The members, methods and classes that are declared public can be accessed from anywhere. This modifier doesn't put any restriction on the access.

```
package abcpackage;


public class Addition {


   public int addTwoNumbers(int a, int b){

            return a+b;

   }

}
```

b. Write a specific program of the above mentioned access modifiers in java.

ANS:

```
 public class PublicProteccted extends Protected{

     public static void main(String []args){
        PublicProteccted obj = new PublicProteccted();
        obj.msg();
        System.out.println("------------------------");
        PublicEg pubOjb = new PublicEg();
        pubOjb.msg();
        System.out.println("End of program");
     }
}

class Protected{
    protected void msg(){
        System.out.println("In Protected");
    }
}

class PublicEg{
    public void msg(){
        System.out.println("In Public");
    }
}

/*Output*/
/*
In Protected
------------------------
In Public
End of program

*/
```

Q3. a. What is inheritance and why it is used, discuss in detail ?
ANS:
Inheritance is a mechanism in which one class acquires the property of another class. For example, a child inherits the traits of his/her parents. With inheritance, we can reuse the fields and methods of the existing class. Hence, inheritance facilitates Reusability and is an important concept of OOPs.
JAVA INHERITANCE is a mechanism in which one class acquires the property of another class. In Java, when an "Is-A" relationship exists between two classes, we use Inheritance. The parent class is called a super class and the inherited class is called a subclass. The keyword **extends** is used by the sub class to inherit the features of super class.
Inheritance is important since it leads to the reusability of code.

```
class subClass extends superClass
{
  //methods and fields
}
```

But one may argue that across all classes, you have a repeated pieces of code.
To overcome this, you create a parent class, say "account" and implement the same function of deposit and withdraw. And make child classes inherited "account" class. So that they will have a ccess to withdraw and deposit functions in account class.The functions are not required to be i mplemented individually. This is Inheritance in java. .

b. Write a program using Inheritance class on Animal in java.

ANS:

```
public class AnimalInheritance{

    public static void main(String []args){
      BabyDog d=new BabyDog();
      d.weep();
      d.bark();
      d.eat();
      System.out.println("------------------------");
      System.out.println("End of program");
    }
}

class Animal{
   void eat(){
      System.out.println("All animals eat.");
   }
}

class Dog extends Animal{
   void bark(){
      System.out.println("All dogs bark.");
   }
}

class BabyDog extends Dog{
   void weep(){
      System.out.println("Baby dog weeps.");
   }
}

/*Output*/
/*
Baby dog weeps.
All dogs bark.
All animals eat.
-----------------------
End of program

*/
```

Q4. a. What is polymorphism and why it is used, discuss in detail ?
ANS:
Answer: The word polymorphism is used in various contexts and describes situations in which something occurs in several different forms. In computer science it describes the concept that objects of different types can be accessed through the same interface. Each type can provide its own, independent implementation of this interface. It is one of the core concepts of object oriented programming If you're wondering if an object is polymorphic, you can perform a simple test. If the object successfully passes multiple is a or instance of tests it's polymorphic. As Ive described in my post about inheritance all Java classes extend the class *object*. Due to this, all objects in java are polymorphic because they pass at least two instance of checks.

Types of polymorphism

Java supports 2 types of polymorphism:

- static or compile-time
- dynamic

Static polymorphism

Java, like many other object-oriented programming languages, allows you to implement multiple methods within the same class that use the same name but a different set of parameters. That is called method overloading and represents a static form of polymorphism.

The parameter sets have to differ in at least one of the following three criteria:

- They need to have a different number of parameters one method accepts 2 and another one 3 parameters
- The types of the parameters need to be different one method accepts a *String* and another one a *Long*

- They need to expect the parameters in a different order one method accepts a *String* and a *Long* and another one accepts a *Long* and a *String* This kind of overloading is not recommended because it makes the API difficult to understand.

A simple example for static polymorphism

I use the same Coffee Machine project as I used in the previous posts of this series. The *Basic CoffeeMachine* class implements two methods with the name *brewCoffee*. The first one accepts one parameter of type *CoffeeSelection*. The other method accepts two parameters, a *CoffeeSelection*, and an *int*.

```
public class BasicCoffeeMachine {

  // ...

  public Coffee brewCoffee(CoffeeSelection selection) throws CoffeeException {

    switch (selection) {

    case FILTER_COFFEE:

      return brewFilterCoffee();

    default:

      throw new CoffeeException(

        "CoffeeSelection ["+selection+"] not supported!");

    }

  }



  public List brewCoffee(CoffeeSelection selection, int number) throws CoffeeException {

    List coffees = new ArrayList(number);

    for (int i=0; i<number; i++) {

      coffees.add(brewCoffee(selection));
```

This form of polymorphism doesn’t allow the compiler to determine the executed method. The JVM needs to do that at runtime.

Within an a subclass can override a method of its superclass. That enables the developer of the subclass to customize or completely replace the behavior of that method.

It also creates a form of polymorphism. Both methods, implemented by the super- and subclass, share the same name and parameters but provide different functionality.

Let’s take a look at another example from the Coffee Machine project.

Method overriding in an inheritance hierarchy

The Basic Coffee Machine class is the superclass of the Premium CoffeeMachine class.

b. Write a program using polymorphism in a class on Employee in java.

Ans:

```
class Child{
   void Print()
   {
      System.out.println("parent class");
   }
}
 class Son extends Child{
  void Print(“ALi”)
   {
      System.out.println("is a son of Atif");
   }
}
 class Brother  extends Child{
   void Print()
   {
      System.out.println("is a brother of Ayesha");
   }
}
 class TestPolymorphism3 {
   public static void main(String[] args)
   {
      Parent a;
      a = new Son();
      a.Print();
```

```
        a = new Brother();
        a.Print();
    }
}
```

```
OUT PUT:
ALi is a son of Atif
Ali is a brother of Ayesha
```

EXPLANATION

Polymorphism is Well explained in above Question as we know its like a same person doing or performing some different task at the same time or have some many and different relation as shown in above program.
Here we are just using the parent class and child class.
The child class is basically acting as a parent class while the other class Son and Brother are acting as a Child Class.

Q5. a. Why abstraction is used in OOP, discuss in detail ?
ANS:
Abstraction:Abstraction is selecting data from a larger pool to show only the relevant details of the object to the user. Abstraction shows only the essential attributes and hides unnecessary information. It helps to reduce programming complexity and effort. It is one of the most important concepts of OOPs. **Abstraction in JAVA** shows only the essential attributes and hides unnecessary details of the object from the user. In Java abstraction is accomplished using Abstract classes Abstract methods and interfaces. Abstraction helps in reducing programming complexity and effort.

Abstract class

A class which is declared abstract is called as an abstract class It can have abstract methods as well as concrete methods A normal class cannot have abstract methods

Abstract Method

A method without a body is known as an Abstract Method It must be declared in an abstract class The abstract method will never be final because the abstract class must implement all the abstract methods

Rules Of Abstract Method

- Abstract methods do not have an implementation it only has method signature
- If a class is using an abstract method they must be declared abstract. The opposite cannot be true This means that an abstract class does not necessarily have an abstract method
- If a regular class extends an abstract class then that class must implement all the abstract methods of the abstract parent

Advantages Of Abstract Method

- The main benefit of using an abstract class is that it allows you to group several related classes as siblings.

- Abstraction helps to reduce the complexity of the design and implementation process of software.

b. Write a program on abstraction in java.

Ans:

```
abstract class Shape
{
abstract void draw();
}
class Rect extends Shape{
void draw()
{
System.out.println("Four Corners ");

}
}
class Circle extends Shape{
```

```
void draw()
{
System.out.println("Zero Corners");
}
}
class TestAbstractClass{
public static void main(String args[]){
Shape s=new Circle();
Shape S=new Rect();
S.draw();
s.draw();
}
}
```

OUT PUT:

```
Four Corners
Zero Corners
```

EXPLANATION

An abstract class can have a data member, abstract method, method body ,non-abstract method, constructor, and even main() method.
Shape is the abstract class, and its implementation is provided by the Rect and Circle classes.
we don't know about the implementation class (which is hidden to the end user), and an object of the implementation class is provided by the factory method.
if you create the instance of Rect class, draw() method of Rect class will be invoked.
While in the Last part of program we just make a Main class So we Can perform some action So in the main class we just make 2 OBJECTS of the abstract class so we can access the Rect and circle Class as we have extended the both classes with abstract class so we can access both of the class data by using the Abstract class.