Name: Muhammad Amir. ID: 16372 Semester: 2nd

Q1: a: why access modifiers are used in java, explain Default and private.

b: write a specific program for the above mentioned access modifiers?

Ans: There are numbers of access modifiers which help you give access you want for your created classes, methods and fields and more from other classes. These modifiers are also known as a scope of a class. There are four types of access modifiers in java but here are going to explain two of them.

1: Default:

If you don't specify any access level it will be called the default modifiers. It can't be accessed by the outside the package. This modifier can be accessed within the same pakage.

2: Protected:

The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

Example of default access modifier

Default:

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

```
1.  //save by A.java
2.  package pack;
3.  class A{
4.    void msg(){
5.  System.out.println("Hello");
6.  }
7.  }
```

```
1.  //save by B.java
2.  package mypack;
3.  import pack.*;
4.  class B{
5.    public static void main(String args[]){
6.     A obj = new A();//Compile Time Error
7.     obj.msg();//Compile Time Error
8.    }
9.  }
```

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

Protected:

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is a compile-time error.

```
1.  class A{
2.  private int data=40;
3.  private void msg(){System.out.println("Hello java");}
4.  }
5.
6.  public class Simple{
7.   public static void main(String args[]){
8.     A obj=new A();
9.     System.out.println(obj.data);//Compile Time Error
10.    obj.msg();//Compile Time Error
11.    }
12. }
```

Ans2 a: Public:

The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Protected:

The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

Public:

The public access modifier is accessible everywhere. It has the widest scope among all other modifiers.

Example of public access modifier

```
1.  //save by A.java
2.
3.  package pack;
4.  public class A{
5.  public void msg(){System.out.println("Hello");}
6.  }
```

```
1.  //save by B.java
2.
```

```
3.  package mypack;
4.  import pack.*;
5.
6.  class B{
7.    public static void main(String args[]){
8.     A obj = new A();
9.     obj.msg();
10.   }
11. }
```

```
Output:Hello
```

In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

```
1.  //save by A.java
2.  package pack;
3.  public class A{
4.  protected void msg(){System.out.println("Hello");}
5.  }
```

```
1.  //save by B.java
2.  package mypack;
3.  import pack.*;
4.
5.  class B extends A{
6.    public static void main(String args[]){
7.     B obj = new B();
8.     obj.msg();
9.    }
10. }
```

```
Output:Hello
```

Ans3: The mechanism by which a class allows to inherit the features like fields and methods of another class. The class can add its own fields and methods in addition to its superclass fields and methods. The new class created is known as derived class from the existing class and that existing class is known as Base class, there can be single inheritance, multi level and hierarchical. Inheritance allows us to reuse of code, it improves reusability in your java application.

Terms used in Inheritance

- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.

- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

Single Inheritance Example

When a class inherits another class, it is known as a *single inheritance*. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

File: TestInheritance.java

```
1.  class Animal{
2.  void eat(){System.out.println("eating...");}
3.  }
4.  class Dog extends Animal{
5.  void bark(){System.out.println("barking...");}
6.  }
7.  class TestInheritance{
8.  public static void main(String args[]){
9.  Dog d=new Dog();
10. d.bark();
11. d.eat();
12. }}
```

Output:

```
barking...
eating...
```

Ans 4:

Polymorphism in Java is a concept by which we can perform a *single action in different ways*. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

If you overload a static method in Java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.

Example of Java Runtime Polymorphism

In this example, we are creating two classes Bike and Splendor. Splendor class extends Bike class and overrides its run() method. We are calling the run method by the reference variable of Parent class. Since it refers to the subclass object and subclass method overrides the Parent class method, the subclass method is invoked at runtime.

Since method invocation is determined by the JVM not compiler, it is known as runtime polymorphism.

```
1.  class Bike{
2.    void run(){System.out.println("running");}
3.  }
4.  class Splendor extends Bike{
5.    void run(){System.out.println("running safely with 60km");}
6.
7.    public static void main(String args[]){
8.      Bike b = new Splendor();//upcasting
9.      b.run();
10.   }
11. }
```

Output:

```
running safely with 60km.
```

Ans 5: Abstract class in Java

A class which is declared as abstract is known as an abstract class**.** It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

Points to Remember

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

Understanding the real scenario of Abstract class

In this example, Shape is the abstract class, and its implementation is provided by the Rectangle and Circle classes.

Mostly, 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.

A factory method is a method that returns the instance of the class. We will learn about the factory method later.

In this example, if you create the instance of Rectangle class, draw() method of Rectangle class will be invoked.

File: TestAbstraction1.java

```
1.  abstract class Shape{
2.  abstract void draw();
3.  }
4.  //In real scenario, implementation is provided by others i.e. unknown by end user
5.  class Rectangle extends Shape{
6.  void draw(){System.out.println("drawing rectangle");}
7.  }
8.  class Circle1 extends Shape{
9.  void draw(){System.out.println("drawing circle");}
10. }
11. //In real scenario, method is called by programmer or user
12. class TestAbstraction1{
13. public static void main(String args[]){
14. Shape s=new Circle1();//In a real scenario, object is provided through method, e.g., getShape() method
15. s.draw();
16. }
17. }
```

```
drawing circle
```