**Final Term Assignment**

**Name: Abubakar Siddique**

**ID: 16343**

Subject: **Object Oriented Programming**

Instructor: **M.Ayub Khan**

**BS SE (section B)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

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

**access modifiers?**

**Ans:**

# Access Modifiers in Java

There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

## Private Access Modifier - Private

Methods, variables, and constructors that are declared private can only be accessed within the declared class itself.

Private access modifier is the most restrictive access level. Class and interfaces cannot be private.

Variables that are declared private can be accessed outside the class, if public getter methods are present in the class.

Using the private modifier is the main way that an object encapsulates itself and hides data from the outside world.

### Example

The following class uses private access control −

public class Logger {

private String format;

public String getFormat() {

return this.format;

}

public void setFormat(String format) {

this.format = format;

}

}

Here, the *format* variable of the Logger class is private, so there's no way for other classes to retrieve or set its value directly.

So, to make this variable available to the outside world, we defined two public methods: *getFormat()*, which returns the value of format, and *setFormat(String)*, which sets its value.

**Default Access Modifier:**

Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.

A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public.

## Example

Variables and methods can be declared without any modifiers, as in the following examples -

String version = "1.5.1";

boolean processOrder() {

   return true;

}

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

**Ans:**

**Coding**

|  |
| --- |
| //Java program to illustrate error while  //using class from different package with  //private modifier  package p1;    class A  {     private void display()      {          System.out.println("GeeksforGeeks");      }  }    class B  {     public static void main(String args[])        {            A obj = new A();            //trying to access private method of another class            obj.display();        }  } |

Output:

error: display() has private access in A

obj.display();

**Q2. a. Explain in detail Public and Protected access modifiers?**

**Ans:**

**Protected access modifier**

The **protected access modifier** is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

It provides more accessibility than the default modifer.

**Example of protected access modifier**

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. }
6. //save by B.java
7. **package** mypack;
8. **import** pack.\*;
10. **class** B **extends** A{
11. **public** **static** **void** main(String args[]){
12. B obj = **new** B();
13. obj.msg();
14. }
15. }

Output:Hello

4) Public access modifier

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
3. **package** pack;
4. **public** **class** A{
5. **public** **void** msg(){System.out.println("Hello");}
6. }
7. //save by B.java
9. **package** mypack;
10. **import** pack.\*;
12. **class** B{
13. **public** **static** **void** main(String args[]){
14. A obj = **new** A();
15. obj.msg();
16. }
17. }

Output:Hello

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

**Ans:**

**Coding protected access**

|  |
| --- |
| //Java program to illustrate  //protected modifier  package p1;    //Class A  public class A  {     protected void display()      {          System.out.println("GeeksforGeeks");      }  } |
| //Java program to illustrate  //protected modifier  package p2;  import p1.\*; //importing all classes in package p1    //Class B is subclass of A  class B extends A  {     public static void main(String args[])     {         B obj = new B();         obj.display();     }    } |
| **Coding public access modifier**  //Java program to illustrate  //public modifier  package p1;  public class A  {     public void display()        {            System.out.println("GeeksforGeeks");        }  }  package p2;  import p1.\*;  class B  {      public static void main(String args[])        {            A obj = new A;            obj.display();        }  } |

**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.

**Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of [OOPs](https://www.javatpoint.com/java-oops-concepts) (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new [classes](https://www.javatpoint.com/object-and-class-in-java) that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a parent-child relationship.

### Why use inheritance in java

* For [Method Overriding](https://www.javatpoint.com/method-overriding-in-java) (so [runtime polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java) can be achieved).
* For Code Reusability.

### 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.

### The syntax of Java Inheritance

1. **class** Subclass-name **extends** Superclass-name
2. {
3. //methods and fields
4. }

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

**Java Inheritance Syntax:**

class subClass extends superClass

{

//methods and fields

}

## Java Inheritance Example

class Doctor {

void Doctor\_Details() {

System.out.println("Doctor Details...");

}

}

class Surgeon extends Doctor {

void Surgeon\_Details() {

System.out.println("Surgen Detail...");

}

}

public class Hospital {

public static void main(String args[]) {

Surgeon s = new Surgeon();

s.Doctor\_Details();

s.Surgeon\_Details();

}

}

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

**Ans:**

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...

**Q4. a. What is polymorphism and why it is used, discuss in detail ?**

**Ans:**

## Polymorphism in OOPS

The literal meaning of Polymorphism would mean” a state of having many shapes.” When this concept is applied to the Object Oriented Programming System (OOPS) language such as Java or C++, it would be able to process in a simple manner. The language would be able to process various kinds of classes and objects through a uniform and single interface.

Basically, Polymorphism is one of the abilities of OOPS for redefining methods for Derived Classes. The behavior of the polymorphic function would depend on the types of data that are being used in the programming. Polymorphism is generally used for implementing inherence in the programming.

For the example of the polymorphism in the OOPS language, it would be the relationship between the parent class object and child class objects. Any object that is able to satisfy the relationship of IS-A for more than one time would be Polymorphic in nature.

There are two major types of polymorphisms in Object Oriented Programming (OOPS) languages. They are Static Binding (Compile time Polymorphism) and Dynamic Binding (Runtime Polymorphism). Method overriding would be the example of Dynamic Polymorphism and Method Overloading would be the example of Static Polymorphism.

### Compile time Polymorphism

By using the method of overloading, one can reach the state of static polymorphism in Object Oriented Programming languages. This method would allow the programmer to implement various methods. They might use the same names but their parameters are different. Method overloading is an example of Static Polymorphism. There are certain conditions that are essential for static polymorphism. They are:

* All Parameters would have to be different in types.
* The order of the Parameter might have to different.
* The number of parameters of one method would have to be different from the other method.

During the run time, compiler of the language would identify various methods by identifying signatures of these methods. The compiler would first identify the method signature and then decide the method for a specific method call while compiling the program.  The execution time for Compile time Polymorphism is much faster but this process is not very flexible.

### Runtime Polymorphism

This process is also known as Dynamic method dispatch. Under this process, a call to a single overridden method is solved during the runtime of the program. Method overriding is the prime example of Runtime Polymorphism. In this process, the call is not solved by the compiler. The overriding is achieved through pointers and virtual functions.

The Method Overriding is the process of declaring a single method in a sub-class that is present in a parent class. Through this process, the child class would gain a method for implementation. This method is given by the parent class of the child class. This process of polymorphism is slow but it is more flexible than the static polymorphism.

The main advantage of Runtime Polymorphism is the ability of the class to offer the specification of its own to another inherited method. This transfer of implementation of one method to another method is possible without changing or modifying the codes of the parent class object. Thus, if a single child class needs the implementation method of the parent class while other child class might use the overriding feature to have different implementation methods.

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

**Ans:**

**coding**

public class Employee {

private String name;

private String address;

private int number;

public Employee(String name, String address, int number) {

System.out.println("Constructing an Employee");

this.name = name;

this.address = address;

this.number = number;

}

public void mailCheck() {

System.out.println("Mailing a check to " + this.name + " " + this.address);

}

public String toString() {

return name + " " + address + " " + number;

}

public String getName() {

return name;

}

public String getAddress() {

return address;

}

public void setAddress(String newAddress) {

address = newAddress;

}

public int getNumber() {

return number;

}

}

**Q5. a. Why abstraction is used in OOP, discuss in detail ?**

**Ans:**

**Abstraction:**

Abstraction is one of the [key concepts](https://stackify.com/oops-concepts-in-java/) of object-oriented programming (OOP) languages. Its main goal is to handle complexity by hiding unnecessary details from the user.

That’s a very generic concept that’s not limited to object-oriented programming. You can find it everywhere in the real world.

**Example:**

Making coffee with a coffee machine is a good example of abstraction.

**Abstraction in oop:**

objects in an OOP language provide an abstraction that hides the internal implementation details. Similar to the coffee machine in your kitchen, you just need to know which methods of the object are available to call and which input parameters are needed to trigger a specific operation. But you don’t need to understand how this method is implemented and which kinds of actions it has to perform to create the expected result.

Let’s implement the coffee machine example in Java. You do the same in any other object-oriented programming language. The syntax might be a little bit different, but the general concept is the same.

**With Detail:**

 “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

**b. Write a program on abstraction in java.**

**Ans:**

**coding**

1. **abstract** **class** Bike{
2. **abstract** **void** run();
3. }
4. **class** Honda4 **extends** Bike{
5. **void** run(){System.out.println("running safely");}
6. **public** **static** **void** main(String args[]){
7. Bike obj = **new** Honda4();
8. obj.run();
9. }
10. }

**Output:**

**Compile by: javac Honda4.java**

**Run by: java Honda4**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_