CORE JAVA

MANUAL V8.3

MODULE CODE:

ANUDIP FOUNDATION


ICONS AND THEIR MEANING
HINTS:
Get ready for helpful insites on difficult topics and questions.
STUDENTS:
This icon symbolize important instrcutions and guides for the students.
TEACHERS/TRAINERS:
This icon symbolize important instrcutions and guides for the trainers.

Module 2: Object Oriented Programming and Package

Chapter 4

Objective: After completing this lesson you will be able to : * Understand the concepts of public, private and protected access modifiers * Gain an understanding of polymorphism in Java	**Materials Required:** 1. Computer 2. Internet access
Theory Duration: 60 minutes	**Practical Duration:** 60 minutes
Total Duration: 120 minutes	

Chapter 4

4.1 Access Modifier (Public, Private, Protected),

What is an access modifier?

An access modifier in Java specifies the classes which can access a particular class, its constructors, fields and methods. Separate access modifiers can be assigned for a class and its fields, methods and constructors. Programmers can use designated access modifiers to modify the level of access for a class, method, field or constructor.

Access modifiers enable improved code encapsulation. Encapsulation is a wrapping process that wraps methods and variables as single units.

Access modifier types include –

* Default
* Public
* Private
* Protected

Public access modifier

A public access modifier is used to declare methods, classes, fields and constructors as 'public' or accessible from everywhere. Components assigned a public access modifier can be accessed from inside the class, outside the class, and from both inside and outside of a package.

Example of public access modifier in Java

Logger.java

```
public class Logger {
public int debugLevel = 1;
public void debug(String logLine){
System.out.println('Debug: '+logLine);

}
public void info(String logLine){
System.out.println('Info: '+logLine);
  }
}
```

LoggerImp.java

```
public class LoggerImp {
public static void main( String[] args ) {
Logger logger = new Logger( );
logger.debug('debug level ' + logger.debugLevel);
logger.debugLevel = 6;
logger.info('info at level' + logger.debugLevel);
   }
}
```

Debug: debug level 1

Info: info at level 6

Private access modifier

If a private access modifier is assigned to methods, constructors and variables of a Java class, those can only be accessed within the class. Subclasses code cannot access the variables that are marked with a private access modifier. A private access modifier is used to achieve encapsulation, to keep data hidden from access. Interface and class cannot be assigned a private modifier.

Example of private access modifier in Java

```
public class Data {
private String name;
public String getName() {
return this.name;
}
public void setName(String name) {
this.format = name;
  }
}
public class Main {
Public static void main(String[] main){
      Data d = new Data()
      d.setName( "Wiz" );
     System.out.println(d.getName());
  }
}
```

Output: Wiz

Protected Access Modifier

A protected access modifier is visible to classes inside a package and also subclasses belonging to other packages. Methods, variables, and constructors, declared as protected in a superclass can be accessed by other package subclasses or protected member class packages. This modifier type cannot be used for interfaces or classes.

Example of protected access modifier in Java

```
Logger1.java
package package1;
public class Logger {protected void debug(String logLine){
System.out.println('Debug line: '+logLine);
```

```
  }
}

// Main.java
package package2;
import package1.Logger1;
public class Main extends Logger1 {
public static void main(String [] args){
Main logger = new Main();

// invokes debug() from Logger1 class

logger.debug('main is hello')
  }
}
```

Output:

Debug line: main is hello

4.2 Polymorphism

Polymorphism in Java refers to the ability of objects to transform into different forms. Objects which pass upwards of one IS-A test are called polymorphic. Polymorphism enables developers to perform individual actions in varying ways. Programmers can define a single interface and have multiple executions.

The two types of polymorphism are -

i) **Compile time polymorphism** – Compile time polymorphism or static polymorphism is enabled by operator or function overloading. Overloading method calls are settled within the compilation time in this type of polymorphism.

Example of compile time polymorphism –

```
class ClassMain{
        void disp(int number){
                System.out.println ('method:' + number);
        }
        void disp(int number1, int number2){
                System.out.println ('method:' + number1 + ',' + number2);
        }
        double disp(double number) {
                System.out.println('method:' + number);
                return num;
        }
}

class CompileTimePolymorphismDemo
{
  public static void main (String args [])
  {
     ClassMain obj = new ClassMain();
     double result;
     obj.disp(60);
     obj.disp(70, 50);
     result = obj.disp(6.1);
     System.out.println('Answer is:' + result);
  }
}
```

Output is:

```
method:60
method:70, 50
method:6.1
Answer is:6.1
```

ii) Runtime polymorphism – Runtime polymorphism or dynamic method dispatch is a method in which an overridden method call can be processed at runtime. In this polymorphism type, a superclass reference variable is used to call an overridden method.

Example of runtime polymorphism –

```
class ClassA{
    public void disp( ){
        System.out.println (‘I am within Class A‘ );
    }
}
class ClassB extends ClassA{
    public void disp( ){
        System.out.println (‘I am within Class B‘ );
    }
}
public class RunTimePolymorphismDemo{
   public static void main (String args [] ) {
       ClassA obj1 = new ClassA( );
       ClassA obj2 = new ClassB( );
       obj1.disp( );
       obj2.disp( );
   }
}
```

Output:

I am within Class A
I am within Class B

Major pillars of OOPS

*** Abstraction –** It deals with showing only the necessary features of an object.

Real life example – To switch on a ceiling fan all one needs to do is press a power switch. There is no need to know about the inner workings of a fan's components.

* **Encapsulation –** Data and method are encapsulated (or wrapped) into a class.

Real life example – Consider an ATM machine as a class containing the data (money) and method (software and hardware mechanism). You can obtain money from the machine, or data contained within the class.

* **Inheritance –** One class (subclass) is created from another class (super-class).

Real life example – We can take the example of a son and his father. The son is the subclass while the father is the super-class. The son inherits the properties of his father.

* **Polymorphism –** A subclass can have its own behaviour while having the same functions of its parent class.

Real life example – We can take the example of a human being (parent class). The human being has all characteristics of a human being, but can also have properties like being a father, a professional, a sportsman, and a son to his father. So he is playing many roles and exhibiting polymorphism (multiple forms).

Minor pillars of OOPS

* **Persistence –** State of an object does not change with space or time.

Real life example – A Nokia phone is a Nokia phone irrespective of where it is, and irrespective of time.

* **Concurrency –** Occurs when multiple operations are running simultaneously and interacting between each other.

Real life example – A computer can be used for performing multiple tasks at the same time. You can perform actions like listening to music and surfing the Internet simultaneously.

Practical (60 minutes)

See the example programme for Java compile time polymorphism below. Write the same programme with integer values 40, 70 and 30. Execute the programme again for integer values 80, 75 and 60. Show the resulting outputs.

```
class ClassMain{
   void disp(int number){
      System.out.println ('method:' + number);
   }
   void disp(int number1, int number2){
      System.out.println ('method:' + number1 + ',' + number2);
   }
   double disp(double number) {
      System.out.println('method:' + number);
      return num;
   }
}

class CompileTimePolymorphismDemo
{
  public static void main (String args [])
  {
     ClassMain obj = new ClassMain();
     double result;
```

```
    obj.disp(60);
     obj.disp(70, 50);
     result = obj.disp(6.1);
     System.out.println('Answer is:' + result);
  }
}
```

Instructions: The progress of students will be assessed with the exercises mentioned below.

MCQ

1. What does an access modifier specify?

a) class

b) array

c) table

d) None of the mentioned

2. Level of access to a method can be modified with _____ _____.

a) method modifier

b) access modifier

c) data modifier

d) None of the mentioned

3. Access modifiers improve code _____.

a) discovery

b) implementation

c) encapsulation

d) moderation

4. To make classes accessible from everywhere programmers can use a _____ access modifier.

a) public

b) protected

c) default

d) private

5. Which modifier enables access to components from both inside and outside a package?

a) public

b) protected

c) default

d) private

6. Subclasses code cannot access the variables that are marked with a _____ access modifier.

a) public

b) protected

c) default

d) private

7. Class and instance cannot be assigned a _____ modifier.

a) public

b) protected

c) default

d) private

8. A _____ access modifier is visible to classes inside a package.

a) public

b) protected

c) default

d) private

9. Objects which pass more than one _____ test are called polymorphic

a) IS-A

b) Has-A

c) Have-A

d) None of the mentioned

10. Operator overloading enables _____ polymorphism.

a) dynamic

b) static

c) hybrid

d) None of the mentioned

Answers: 1.a, 2.b, 3.c, 4.a, 5.a, 6.d, 7.d, 8.b, 9.a, 10.b