26/9/23

ASSIGNMENT 2

Unit -3 & 4 Class, Object and Methods & Inheritance, Packages and Interfaces.

Q.1 Explain class and object with respect to Java.

→ A class is a blueprint or template for creating objects which defines the structure and behaviour of objects of that type. It specifies what attributes an object of that class will have and what method can be called on objects of that class.

Syntax : class class_name {
    // methods
    // fields
} class body

→ An Object is an instance of a class. It is a concrete, tangible entity created based on the blueprint ~~provided~~ by a class. It can interact with each other by invoking methods defined in their respective classes.

Syntax : className Object = new ClassName();

example:
```
public Class Main {
    int x = 5, y = 11;
    public static void main (String args[]) {
        Main Obj1 = new Main();
        Main Obj2 = new Main();
        System.out.println(Obj1.x);
        System.out.println(Obj2.y); } }
```

Output : 5
11

Q.2 Discuss various access modifiers available in JAVA? How access modifier affects the visibility of a member in different access locations?

→ There are four access modifiers which are used to control the visibility or accessibility of class members from other classes or packages.

i) Public : Members declared as 'public' are accessible from anywhere, both within the same class, in other classes within the same package, and in classes from different packages. It has the highest level of visibility.

```
eg: public class MyClass {
      public int publicVar;
      public void publicMethod ()
```

ii) Protected : Members declared as 'protected' are accessible within the same class, in subclass (even if they are in different packages), and within the same package.

```
eg: public class MyClass {
      protected int protectedVar;
      protected void protectedMethod ()
```

iii) Private : Members declared as 'private' are accessible only within the same class. They are not accessible or visible from any other class.

eg: public class MyClass {
    private int privateVar;
    private void privateMethod()

ii) Default: When no access modifier is specified, members are accessible only within the same package.

eg: class MyClass {
    int defaultVar;
    void defaultMethod()

* Visibility	Default	Private	Protected	Public
same class	Yes	Yes	Yes	Yes
same package subclass	Yes	No	Yes	Yes
same package Non-Subclass	Yes	No	Yes	Yes
Diff^{n} Package Subclass	No	No	Yes	Yes
Diff^{n} Package Non-Subclass	No	No	No	Yes

Q.3 What is constructor? Explain constructor overloading with example.

→ A Constructor is a type of method that is called when an object of a class is created. It is used to initialize the newly created object. They have same name as the class and do not have a return type, not even 'void'. They are used to set initial values for the objects attributes.

→ Constructor Overloading: It is a concept where a class can have multiple constructors with

different parameter lists. Each of them provides a different way to create an object of the class, allowing you to initialize the object with different sets of values. The choice of which constructor to use depends on the arguments.

example:

```
class Student {
    private String name;
    private int age;
    public Student() {
        this.name = "Unknown";
        this.age = 0;
    }
    public Student(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public String getName() {
        return name;
    }
    public int getAge() {
        return age;
    }
}
public class Main {
    public static void main(String args[]) {
        Student student1 = new Student();
        Student student2 = new Student("Tony", "23");
        System.out.println(student1.getName() + ", " +
```

```
        Student1.getAge());
        System.out.println(Student2.getName() + "," +
        Student2.getAge());
    }
}
```

Output : Unknown , 0
Tony, 23

Q.4 With the help of program show how to pass object as an argument of method.

—>
```
class Rectangle {
    private int length;
    private int width;
    public Rectangle(int length, int width) {
        this.length = length;
        this.width = width;
    }
    public int calculateArea() {
        return length * width;
    }
}
public class Main {
    public static void printArea(Rectangle rectangle) {
        int area = rectangle.calculateArea();
        System.out.println("Area of rectangle :" + area);
    }
    public static void main(String args[]) {
        Rectangle shape = new Rectangle(11,30);
        printArea(shape);
    }
}
```

#Output: Area of Rectangle : 330.

Q.5 Differentiate the followings:

i)

Constructor	Method
> A block of code that initializes a newly created objects	A collection of statements which return a value upon its execution.
> Doesn't have a return type	Must have a return type.
> It's name must be same as the name of class	It's name can be anything.
> It cannot be inherited by subclasses	It can be inherited by subclasses
> It is invoked when a object is created using the keyword 'new'.	It is invoked through method calls.

ii)

Abstract class	Interface
> It can have abstract and non-abstract method	It can only have abstract methods
> It doesn't support multiple inheritance	It supports multiple inheritance.
> It can be extended using keyword "extend"	It can be implemented using "implements".

> It can have class members like private, protected, etc.	It Here, members are public by default
> It can have final, non-final, static and non-static variables	It can have only static and final variables.

Q.6 Explain how you can pass and return objects using methods.

→ i) Passing objects as Method Arguments:

```
class Person {
    private String name;
    public Person (String name) {
        this.name = name;
    }
    public String getName () {
        return name;
    }
}
public class Main {
    public static void modify (Person person, String
                               newName) {
        person = new Person (newName);
    }
    public static void main (String args[]) {
        Person person1 = new Person ("Tony");
        System.out.println ("Before: " + person1.getName
        modify (person1, "Pepper");
        System.out.println ("After: " + person1.getName()
    }
}
```

Output: Before: Tony

ii) Returning Objects from Methods.

```
class Person {
    String name;
    public Person (String name) {
        this.name = name;
    }
}
public class Main {
    public static Person createPerson (String name) {
        Person person = new Person (name);
        return person;
    }
    public static void main (String args[]) {
        Person newPerson = createPerson ("Tony");
        System.out.println ("Name: " + newPerson.name);
    }
}
```

Q.7 What is nested class? Mention its types.

—ᵛ A Nested class is a class defined within another class and are also known as inner class. These nested classes have access to the members of their enclosing class and they can be used to encapsulate and group related functionality within parent class. They can improve code organisation, reduce code duplication. There are mainly two types of nested classes : Static and Non-Static

—ᵛ Static Nested class does not have access to the

members
instance-specific (i.e. non-static) of the enclosing class.

→ Non-Static Nested class have access to both static and instance-specific members of the enclosing class.

Syntax: class OuterClass {
    // Block of code
    class NestedClass {
        // Block of Code
    }
}

Q.8 Explain Java garbage collection mechanism.

→ Java Garbage collection mechanism is an automatic memory management system that helps manage memory allocation and dellocation for objects created in the program. It is designed to alleviate the burden of manual memory management, and to prevent memory leaks and segmentation faults, common issues in languages like C/C++.

First of all, when an object is created the Java memory management system allocates memory for the object on the heap (a region of memory). Then reference-counting mechanism keeps track of object references. Then the garbage collector scans through the entire heap and reclaims memory occupied by the unreferenced object which are considered as

garbages. This process is run automatically by Java's runtime system when some conditions are met like the heap is running low on memory. The purpose of this is to free up memory for new object allocations.

Despite automation, developers can still manually manage resources for non-memory resources.

Q.9 Describe Inheritance and its types with suitable example.

→ Inheritance is a concept which allows a class to inherit the properties and behaviours of another class. Inheritance is done in subclass or child class from super-class or parent-class. It promotes code reuse, extensibility and creates relationship between classes. There are three types of inheritance possible using class, which are:

i) Single Inheritance: When a class inherits from only one superclass. Every class can have only one direct superclass.

ii) Multilevel Inheritance: When a class inherits from another class, which in turn inherites from another class. It creates a chain of inheritance.

iii) Hierarchical Inheritance: When multiple subclasses

inherits from a single super-class.

Example:

```
class Animal {
    void eat() {
        System.out.println("Animal is Eating");
    }
}
// Single Inheritance
class Dog extends Animal {
    void bark() {
        System.out.println("Dog is Barking");
    }
}
// Multilevel Inheritance
class Labradar extends Dog {
    void play() {
        System.out.println("Labradar is Playing");
    }
}
// Hierarchical Inheritance
class Cat extends Animal {
    void meow() {
        System.out.println("Cat is meowing");
    }
}
public class Main {
    public static void main(String args[]) {
        Dog dog = new Dog();
        dog.eat();   # Animal is Eating
        dog.bark();  # Dog is Barking
```

```
        Labrador labrador = new Labrador();
        labrador.eat(); # Animal is Eating
        labrador.bark(); # Dog is Barking
        labrador.play(); # Labrador is Playing
        Cat cat = new Cat();
        cat.eat();  # Animal is Eating
        cat.meow();  # Cat is Meowing
    }
}
```

Q.10 What is multiple Inheritance? write a Java program to implement multiple inheritance

—↓ Multiple inheritance is a feature where a class can inherit attributes and methods from more than one superclass. It is not directly supported in Java for classes but through interfaces it can be done.

example:

```
interface Ingredient {
    void add(); }
interface Mixing {
    void mix(); }
class Beverage {
    String name;
    Beverage (String name) {
        this.name = name; }
    void serve() {
        System.out.println("Serving " + name);
    }
}
```

```
class Coffee extends Beverage implements Ingredient, Mixing {
    Coffee (String name) {
        super(name); }
    public void add() {
        System.out.println("Adding Coffee bean to "+name);
    }
    public void mix() {
        System.out.println("Mixing Coffee with hot water");
    }
}
public class Main {
    public static void main (String args[]) {
        Coffee coffee = new Coffee ("Espresso");
        coffee.add();
        coffee.mix();
        coffee.serve();
    }
}
```

Q.11 How can we protect sub classes to override the method of super class? Explain with example.

—> In Java, we can protect subclass to override the method of super-class with the help of "final" modifier. When you declare a method as 'final' in the superclass it becomes a non-overridable method and any attempt to override, it will give a compilation error.

Example:

```
class Super {
    final void finalMethod () {
```

```
        System.out.println("Final Method");
    }
}
class Subr extends Super {
    //void finalMethod() {
    //    Sout("Overriden Method"); }
}
public class Main {
    public static void main (String args[]) {
        Super obj1 = new Super();
        Sub obj2 = new Sub();
        obj1.finalMethod(); # Final Method
        obj2.finalMethod(); # Final Method.
    }
}
```

Q.12 What is polymorphism? what do you mean by run-time polymorphism? write a program to demonstrate run time polymorphism.

—x Polymorphism is a concept which allows objects to of different classes to be treated as objects of a common superclass. The word itself means "One name Many forms" in greek. There are two types of polymorphism:

i) Compile - Time Polymorphism:

Also known as method overloading or early binding. It involves multiple methods with same name in the same class, but with different parameter lists, and occurs at compile time.

Q.13 Explain Dynamic method dispatch with proper example
→ Same as Q.12

ii) Run-time Polymorphism:
Also known as method overriding or late binding. It involves a subclass providing a specific implementation for a method defined in its superclass.

example:
```
class Animal {
    void sound() {
        System.out.println("Animal makes a sound");
    }
}
class Dog extends Animal {
    void sound() {
        System.out.println("Dog Barks");
    }
}
class Cat extends Animal {
    void sound() {
        System.out.println("Cat Meows");
    }
}
public class Main {
    public static void main(String args[]) {
        Animal animal1 = new Dog();
        Animal animal2 = new Cat();
        animal1.sound(); # Dog Barks
        animal2.sound(); # Cat Meows
    }
}
```

Q.14 Discuss the following with example:

i) Super Keyword.

→ 'Super' is used to refer to immediate parent class of a child class. It is often used to access members of the superclass that are hidden or overriden by a member with same name in the subclass. It also invokes the superclass's constructor explicitly.

- Accessing Superclass Members.

eg:
```
class Super {
    int num = 30;
}
class Sub extends Super {
    void display() {
        System.out.println(super.num); }
}
```

- Calling Superclass Constructor.

eg:
```
class Super {
    int num;
    Super(int num) {
        this.num = num; }
}
class Sub extends Super {
    int num2;
    Sub(int num, int num2) {
        super(num);
        this.num2 = num2; }
}
```

ii) Final Keyword.

→ 'final' is used to declare a member as unchangeable or non-~~extensible~~ extendable. Its usage varies based on the context:

(i) Final Fields: It becomes a constant and cannot be modified after initialization.

(ii) Final Methods: It cannot be ~~overridden~~ overridden by subclasses

(iii) Final classes: It cannot be extended by other classes.

eg: final class Final {
    final int num = 100;
    Final void (int num) {
        System.out.println ("Final Method"); }
}

Q.15 Explain Abstract classes with example.

→ An Abstract class is a class that cannot be instantiated on its own and is typically used as a blueprint for other classes. They are useful when you want to define common behaviour and structure for a group of related classes while leaving methods to be implemented by subclasses.

example: abstract class Shape {
    abstract double area();
    void display() {
        System.out.println ("This is a Shape"); }
}

```
class Circle extends Shape {
    private double radius;
    public Circle (double radius) {
        this.radius = radius; }
    double area() {
        return Math.PI * radius * radius; }
}
public class Main {
    public static void main (String args[]) {
        Circle circle = new Circle (5.0);
        circle.display();
        System.out.println("Area :" + circle.area());
    }
}
```

Q.16 Define Interface and explain how it differs from the class.

—> An Interface is a blueprint for a class that defines a set of abstract methods that must be implemented by any class that implements the interface. It allows you to define a set of rules that concrete classes must follow.

—> Here are some features of interfaces and how they differ from class:

> In Java, a class ~~can~~ does not support multiple inheritance but through interfaces it can happen.

> To declare an interface, you use the 'interface'

keyword. In contrast, classes are declared using the 'class' keyword.

- Interfaces cannot have constructors because they cannot be instantiated directly and on the other hand concrete classes can have their constructors.
- To inherit interface, the keyword used is 'implements' and to inherit class, the keyword used is 'extends'.
- Variables and methods are declared as public in interface & in class they can declared using any access modifier.

Q.17 Can we re-assign a value to a variable of interface? Explain use of Interface with suitable example.

→ You cannot re-assign a value to a variable declared ~~within~~ an interface. Interface variables are implicitly 'public', 'static' and 'final', which means they are constants and cannot be modified once they are assigned a value.

example:
```
interface Constants {
    int Max_Value = 100;
}
public class Main {
    public static void main (String args[]) {
        // Constants.Max_Value = 200; # Error
        System.out.println ("Max Value :" +
        Constants.Max_Value);
    }
}
```

Q.18 What is a Package? What are the benefits of using packages? Write the steps in creating a package and using it in a java program.

→ A Package is a mechanism used to organize related classes and interfaces into a single namespace. It provides a way to group classes that have similar functionalities, making it easier to maintain.

→ Benefits of using Package:

i) Organization: Allows you to organize your code into logical groups, making it easier to locate and manage

ii) Reusability: Promotes code reusability by allowing you to create libraries of related classes.

iii) Access Control: You can control the access to classes and members using access modifiers

iv) Namespace Management: Prevent naming conflicts by providing a unique namespace for classes.

→ Steps to create a package and use it in program:

i) Package Declaration.

```
package packageName1.pack2.....;
```

ii) Define classes

> Define classes within the package.

iii) Compile the Code

> Using javac -d . *.java

java MainFile.java

iii) Import and use

↳ Import the package, using 'import' where you want to use classes from the package.

```
import com.example.mypackage.MyClass;
```

Q.19 Explain the Static and this keyword with the help of an example.

↳ Static Keyword : Used to declare members that belong to the class itself, rather than to instances of the class. It is shared among all instances of the class and can be accessed using the class name.

↳ This Keyword : A reference to the current instance of the class. It is used to differentiate between instance variables and parameters with the same name.

Example:

```
public class Main {
    private static int staticVariable = 0;
    private int instanceVariable;
    public Main(int instanceVariable) {
        this.instanceVariable = instanceVariable;
        staticVariable++; }
    public void display() {
        System.out.println("Static : " + staticVariable);
        System.out.println("Instance : " + this.instanceVariable); }
    public static void main(String args[]) {
        Main obj1 = new Main(5);
        Main obj2 = new Main(10);
        obj1.display();
        obj2.display(); } }
```

Repeat