UNIT-III
Polymorphism

Polymorphism is one thing in many forms. It is classified into two types.

1. Compile time polymorphism
2. 2. Runtime polymorphism

The polymorphism that can be identified when the code is compiled is called compile time polymorphism and it is further classified into two types viz., operational polymorphism and functional polymorphism. The operational polymorphism is implemented by operator overloading, by assigning a new functionality to the operator. But as it changed basic meaning of operator, it is not supported in java. But java has pre-defined operational polymorphism implemented for + for string concatenation.

Polymorphism using Methods: is a concept of defining more than one method with same name with in a class. It can be done by functional overloading.

It can be done in following ways:
o By passing different number of arguments
o By passing different data type arguments
o By passing arguments in different sequence

Ex:

```
class Pattern {
// method without parameter
        public void display() {
                for (int i = 0; i < 10; i++) {
                        System.out.print("*");
                }
        }

// method with single parameter
        public void display(char symbol) {
                for (int i = 0; i < 10; i++) {
                        System.out.print(symbol);
                }
        }
}

class Main {
        public static void main(String[] args) {
                Pattern d1 = new Pattern();
                // call method without any argument
                d1.display();
                System.out.println("\n");
                // call method with a single argument
                d1.display('#');
        }
}
```

Output:
****
##########

Q. Write about overloading methods

Runtime polymorphism-Dynamic binding:

Dynamic binding is also called polymorphism with Variables, with class object variables (objects)we can implement polymorphism during runtime of program. Dynamic binding is a concept of assigning derived class instance to its base class reference during running of program. In this case, the base reference behaves as derived instance whose memory is assigned to it.

Ex:

```
class ProgrammingLanguage {
	public void display() {
		System.out.println("I am Programming Language.");
	}
}

class Java extends ProgrammingLanguage {
	@Override
	public void display() {
		System.out.println("I am Object-Oriented Programming Language.");
	}
}

class Main {
	public static void main(String[] args) {
		// declare an object variable
		ProgrammingLanguage pl;
		// create object of ProgrammingLanguage
		pl = new ProgrammingLanguage();
		pl.display();
		// create object of Java class
		pl = new Java();
		pl.display();
	}
}
```

The compile time polymorphism is further classified into two types viz., operational

Inheritance

Inheritance is getting features of one class into another. The class giving features is called base class or super class and the class getting features is called a derived class. When we create object of base class the object abstracts all base class features and when we create

object of derived class, it gets both base and derived features. In java, the keyword “extends” is used to implement inheritance.

The inheritance can be understood in different ways like,

- Getting features of one class into another
- Creating a new class as of existing class
- Extending features of existing class with the help of a new class

Example:

```
class base{
}
class derived extends base{
}
```

Inheritance hierarchies, super and sub classes, member access rules:

- Single inheritance: one derived class extending features of one base class.
- Hierarchical inheritance: multiple derived classes extending features of one base class.
- Multi-level inheritance: a class derived from other class which is derived from another.
- Multiple inheritance: one class extending features from multiple classes.
- Hybrid inheritance: mixture of any two or more of above types.

The main advantage of inheritance is reusability of code. When designing the base and derived classes, the generalization and specialization technique is used. The most general / common features are declared in base class and most specific features are declared in derived class.

The base class that gives features is called super class / parent class in java and the derived classes are called child classes or sub classes.

The java doesn’t support multiple inheritance of classes, as it is away from practical life approach.

When we create base object, we can access only base class features with that object and when we create derived class instance, we can access both base and derived class features.

When we create instance of base class, only base constructor gets called and when we instantiate derived class, first base constructor gets called and then derived constructor, as unless base memory existing, it cannot be extended. Once inheritance is implemented, as the derived class gets all the features of base class, we can implement nesting of methods in derived class. i.e. we can call base class methods directly in derived class method definitions without object.

Q. Explain Inheritance concepts in java

Method overriding:

It is a concept of defining a method in derived class that already has a definition in base class. i.e. both base and derived classes will have methods with same name. In this case, when the method is called with base object, base method definition executes and when called with derived object, derived method definition executes.

When derived class gives definition to a method that already exists in base class, then it is said that the derived class is overriding the base class method. With this technique of overriding, standard method names are maintained in the entire program. i.e. the same method when called for base object performs a different process and for derived object performs a different process.

Example:

```
class base{
        public void disp(){
                System.out.println("base");
        }
}

class derived extends base{
        public void disp(){
                System.out.println("derived");
        }
}
```

In above example, it is said that the class derived is overriding the disp() method of the class base. If the disp() is called with base object base's method disp() executes and when called with derived object, derived's disp() is executed.

Q. What is method overriding?

---

abstract classes- abstract methods:

Abstraction is a quality of dealing with ideas rather than events. It is a process of hiding the implementation details and showing only functionality to the user, i.e. it lets user know what to be done instead of how to be done. In java, the low level abstraction is done when the class is instantiated. The object lets user to know what to be done by hiding how to be done. It is something like, the bike rider knows that the bike starts without knowing how it happens within the bike. The high level abstraction in java is done by abstract classes and interfaces. With the help of abstract classes one can identify what to be done in the program rather than how it is implemented. The abstract classes are concrete by design. A class is made as abstract class by declaring it with the keyword "abstract". Abstract class may have abstract methods or non-abstract methods i.e. normal methods with definitions.

Abstract method is declared with keyword "abstract" and it cannot have body (definition). It can have only declaration. Abstract classes may have or may not have abstract methods. But if a class has at least one abstract method, then the class must be declared as abstract class. Abstract classes cannot be instantiated, i.e. allocated with memory directly. So they can be used as base classes in inheritance.

If an abstract class is extended, then the derived class must override all the abstract methods of the abstract base class. i.e. the derived class provides the implementation of the methods of base class.

An abstract class can have constructors, static members and also final methods. The final methods force the derived class NOT to change the body of the method. From a programmer point of view, the abstract classes are used to make sure standard method names are used in all derived class.

The abstract class provides the information about what to be done and the derived classes provide its implementation, i.e. how it is to be done

---

Q. Write about abstract classes in java

Interfaces:

An interface in java is like a pure abstract class. The interface is defined like a class but with keyword “interface” instead of “class”. The interface methods are by default abstract. So the interfaces cannot have non-abstract methods. The interface variables are by default static and final. So the values assigned to them are not modifiable.

These are concrete by design. These provide information of what to be done in derived classes. The derived classes will have implementation specifying how it is to be done. The interface will have method declarations only. These cannot have definitions / processes to be done. The implementing sub class of interface will have to provide the definition / how the process is to be implemented.

Syntax:

```
interface <interface_name> {
variables;
method declarations;
}
ex:
interface abc{
int x=10;
public void disp();
}
```

In above example the variable x declared with in the interface abc is static and final. It can be accessed with interface name like abc.x and its value can not be modified. The method disp() doesn’t have definition. It is abstract by default. It tells what to be done in derived class. If a class implements the above interface “abc”, then that class must override the method disp() and provide the definition in that class specifying how the process is to be done.

The interfaces cannot be instantiated as these are abstract. These can be used as bases in inheritance. The interfaces can be used in inheritance with the keyword “implements” instead of extends.

Ex:
class xyz implements abc{
}
Now the class class xyz provides implementation for all the methods declared with in interface abc.
In java an interface can “extends” other interface. But a class implements the interface.
interface xyz extends abc{
}
The interface variables can be accessed either with its derived class object or with interface name as these are static and final.

Multiple inheritance using interfaces:

Java doesn’t support multiple inheritance of classes as it is impractical. But Java supports implementing multiple interfaces, thus providing multiple inheritance concept fulfilled with interfaces.
Ex:
interface I1{
}
interface I2{
}
class abc implements I1, I2{
}

Where both I1 and I2 are interfaces and now the class abc must override all the methods of both I1 and I2 interfaces.

We can also implement dynamic binding with interface references. i.e. we can access derived class’s overridden features with interface reference by assigning derived class instance to interface reference.

Q. Explain about interfaces and its specifications.
Q. Explain about multiple inheritance of interfaces.

Encapsulation:

Access specifiers and modifiers :

Encapsulation is one of the fundamental principles of Object-Oriented Programming. It is the process of binding data and methods together into a single unit called a class. More importantly, encapsulation is **selective hiding of class members in an object**. This means that not all members of a class are made accessible to the outside world. Only the required members are exposed, while the internal implementation details are hidden. This concept is also known as data hiding. By restricting direct access to the internal data of a class, encapsulation protects the integrity of the data and ensures controlled access through defined methods. It improves security, maintainability, and flexibility of the program.

Encapsulation in Java is implemented using access specifiers and modifiers. Access specifiers control the visibility of class members such as variables, methods, constructors, and classes. Java provides four access specifiers: private, default, protected, and public. The

private access specifier allows members to be accessed only within the same class and is mainly used for data hiding. The default access specifier allows access only within the same package. The protected access specifier allows access within the same package and also in subclasses outside the package. The public access specifier allows members to be accessed from anywhere in the program. By properly choosing access specifiers, a programmer can control how and where class members are accessed.

Apart from access specifiers, Java also provides other modifiers that define the behavior of class members. **These control behavior, not visibility.** The final modifier is used to prevent modification; a final variable cannot change its value, a final method cannot be overridden, and a final class cannot be inherited. The static modifier indicates that a member belongs to the class rather than to an object and is shared among all instances. The abstract modifier is used for classes and methods to indicate incomplete implementation, requiring subclasses to provide implementation. Other modifiers like synchronized are used to control execution behavior in multithreaded environments.

Thus, encapsulation combined with access specifiers and modifiers ensures controlled access, security, and proper structure in Java programs. It plays a vital role in achieving modularity and robustness in object-oriented software development.

Q. Explain about access specifiers & modifiers.
Q. Write about excapsulation
