

UNIT-II

Defining Classes, Creating objects, Methods

Classes and Objects:

To work with OOP languages, one should have knowledge of two things.

1. Class
2. Object

Class is an ADT (Abstract Data Type). Simply, it is a user defined data type. It is used to group the variables and functions that are required for an object. It is like a template of object or a blue print of object. We use the keyword “class” to create a class.

Syntax :

```
class<Class Name>
{
    Data members

    Member functions / methods
}
```

Object is a named instance of class. It is like a variable of a primitive data type. When we create an object to a class all the features of the class are abstracted to the object. We can make use of the features declared in class through object with a “.”DOT operator called PERIOD operator.

A method is a function defined within a class that does some process. We can also pass parameters to these methods as we did in C language.

Java is dynamic, i.e. each object in java must be allocated with memory with the keyword “new”. When we declare a reference for class, it cannot be used directly without instantiation (created with instance i.e. allocated with memory). When the reference is instantiated, then it becomes object.

A language to be an OOP, should have some standards and are called oops concepts.

They are:

Abstraction : Abstracting features of class into object in different ways.

Encapsulation: Selective hiding of class members in object

Polymorphism : One thing in many forms

Inheritance: Getting features of a class into another.

Different benefits of oops achieved by these concepts are better memory management, high data security, easy identification of methods that process data, reusability of code etc.

Q. Write about class-object and methods

OOPS Concepts – A language to be an OOP, should have some standards and are called oops concepts.

They are Abstraction, Encapsulation, Polymorphism, and Inheritance.

Abstraction: It is getting, bathing, grouping, summarising etc. When we create an instance of class, all the features of the class are abstracted (summarised) into that instance / object. This process of getting features of class into object is called abstraction.

The abstraction is done in two ways viz., data abstraction and functional abstraction.

Getting features of data members i.e. member variables of the class into object is called data abstraction. It is done individually and independently to each object.

Getting functions (method) features into the object is called functional abstraction. It is done once and shared by all the objects, thus providing better memory management.

When a method is called for an object, the object puts itself into the method definition and undergoes process, and that object that put itself into the method definition is called current object.

Encapsulation (Data hiding techniques) : Encapsulation is selective hiding of class members in object. The keywords private, protected and public are used as accessibility specifiers or modifiers used for encapsulation.

Private : the features are not accessible with object directly.

Protected : these features are accessible in current class and also in derived class.

Public : accessible in program anywhere with object.

If no access specifier is mentioned, then the java's default accessibility option is applied for all members. When default accessibility is applied, then they are accessible with object in all classes in current folder / package.

The other access controlling techniques are :

abstract : used to develop abstract classes and abstract methods.

An abstract method is one that has no definition, i.e. has no implementation and it is used to make sure the derived class provides its implementation.

Abstract classes can have abstract methods and also non-abstract methods. But if a class has atleast one abstract method then the class must be abstract class.

static : any feature, either a variable or a method declared as static within a class is allocated with memory even if object is not created for that class. So the java provides a facility of using them with class name also.

The static variables or methods are generally called class features and the non-static features are called instance features.

When a class has static variables / objects declared, they are generally initialized in a static block instead of constructor.

final : in C and C++, the keyword "const" is used for declaring constant variables. In java, the keyword "final" is used for declaring constants.

The final keyword can be used with variables, methods and also with classes.

The final variable can not be modified with its values

The final methods can not be overridden in derived classes
The final classes can not be used in inheritance as base classes.

volatile : the variables that need to change their values frequently for number of times during running of program can be declared with keyword “volatile”. It is something like register variables of C.

synchronized : used in multi threading programming

native : when the code is written in C or C++ languages and that code need to be used in a java program, then we use the keyword “native” to specify that the code is from external resource and need to be executed in native environment.

Polymorphism (Compile Time and Run Time Polymorphism), Method Overloading, Dynamic Binding

Polymorphism:

It is a concept of one thing existing in many forms. The polymorphism in java is classified into 2 types.

1. Runtime polymorphism : it is dynamic binding
2. Compile time polymorphism : The compile time polymorphism 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:

```

*****
#####

```

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 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.

Inheritance (Types of Inheritance, accessing super class members, Method Overriding, ‘super’ Keyword).

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 subclasses, member access rules,

The different hierarchies of inheritance are :

- 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 in java

Q. Explain Encapsulation or Explain access controls in java

Q. What is abstraction

Q. Explain Dynamic binding

Q. Explain Polymorphism

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.

Generally to make sure the overridden method is actually overridden correctly and not assumed to be overridden, we mention the annotation @Override at overridden method.

So the derived class actually looks as :

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

Q. What is method overriding?

'super' keyword :

```
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. In this case, disp() of base method cannot be called with derived object as local features have higher priority, when the disp() is called for derived object, only derived method definition executes.

In java, the keyword "super" is used to refer immediate base class features in derived class. To make a call of base class disp() with derived object, the derived disp() can be called with derived object and then base class disp() can be called in derived disp() definition with "super" keyword.

The Protected Specifier: protected is an access specifier, that controls the members can be accessed in current class or with in the methods of its derived classes.

Example:

```
class base{
    public void disp(){
```

```

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

```

** The super() can also be used to call a specific constructor of base class

Q. write about super keyword

Constructors: constructor is a special method of the class that has same name of the class name and will be invoked automatically whenever the class is instantiated, i.e. allocated with memory with new operator. The constructor can also take arguments. The class may have any number of constructors where each has different number of arguments or different data type argument implementing functional polymorphism. The constructor that has no arguments is called default constructor.

Each class in java must have a constructor. If no constructor is written in a class, then the JRE declares a default constructor in that class with empty definition. If a constructor is declared in a class with arguments and no default constructor written, then the JRE won't create a default constructor for that class. As these methods (constructors) are used to construct memory for the member references declared in a class, these are called constructors. And these can also be used for initializing the values of data members.

Though the constructors are methods, but not used for general process like normal methods, the constructors cannot return any value and these have no return type.

Q. Write about constructors

'this' keyword : The functions in C are called methods in java. A method is piece of code identified with a name. A method has two aspects, Method definition and method call. A method is defined within a class and can be called with object. The method name includes access specifier, access modifier, return type, method name and arguments.

Ex:

The first line that includes method name, arguments etc is called function signature in C and the same is called method header in java. The method body includes the process to be done. A method once defined, can be called with object any number of times. When a number of objects are created for a class, the methods are allocated with memory once and shared by all objects, and it is called functional abstraction, thus methods provide better memory management.

When a method is called by an object, the object puts itself into that method definition and gets processed. And the object that puts itself into method to undergo process is called “current object”. This current object is referred with keyword “this” within that method.

The “this” is a keyword used to refer current object and also we can use “this()” to pass arguments from one constructor to another constructor of the same class, if the class has more than one constructor.

The methods declared within a class are called instance methods of the class. The instance methods are allocated with memory once the objects are created. These can be called only with objects.

Q. Explain this keyword

Static fields and methods:

static : is a keyword used to declare class members. The class members are allocated with memory soon after program execution starts even if object is not created. So the static members can be called with class name without object. The Method header

```
public static void main(String args[]){
process;
}
```

static variables also get allocated with memory once. When a number of objects are created for the class, still the static variables will have memory allocated once and shared among all the objects. Static variables can be declared only within class. They cannot be declared within a method.

In java, methods can also be declared as static. If a method is declared as static, it can be called with class name without object. A static method of a class can also be called directly within another static method of same class.

Static Block : a class can have a static block for initializing static variables of the same class.

Ex:

```
class MyStaticTest{
    static int a;
    static{
        a=10;
    }
}
```

Q. write about instance methods and static methods of java

Q. what are the differences between instance and static methods of java

Abstract Classes and Interfaces – Defining Abstract Classes, Interfaces and their Implementation.

Abstract classes and 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 can not 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 xyz provides implementation for all the methods declared within 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.

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:
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.

Interfaces VS Abstract classes:

- Abstract classes are partially abstract and interfaces are pure abstract
- Abstract class can have normal variables also, but interface variables are by default static and final
- Abstract class can have abstract methods and also non-abstract methods, whereas the interface methods are by default abstract.
- Abstract classes can be extended by another class to provide implementation of abstract features and interfaces can be implemented to have implementation of its abstract features.
- Classes don't support multiple inheritance but interfaces support multiple inheritance.

Q. Explain about interfaces

Think about 5 marks questions on each aspect here.

Introduction to AWT – Component, Frame, Label, Button, Text Field, Choice, List.

awt : this package has classes for GUI development.

Frame : this class provides a base container onto which other components can be added.

classes hierarchy : Object -> Component -> Container -> Window -> Frame

constructors :

Frame()

Frame(String title)

methods :

setSize(int w, int h)

setLayout();

setVisible(boolean)

add(Component)

setLocation(int x, int y)

setResizable(boolean)

setTitle(String)

getTitle()

setUndecorated(boolean).. if set to true, the border and titlebar are removed

Label : this class is used for labeling other components.

CH : Object -> Component -> Label

const :

Label()

Label(String text)

methods :

setText(String)

getText()

Button : this class provides a push button.

CH : Object -> Component -> Button

const :

- Button()
- Button(String label)

methods :

- setLabel(string)
- getLabel()
- addActionListener()

TextField : this class provides a box in which user can type single lined text.

CH : Object -> Component -> TextComponent -> TextField

const :

- TextField()
- TextField(int size)
- TextField(String default text)
- TextField (Styring text, int size)

methods :

- setText(String)
- getText()
- setEchoChar(char)... used for password boxes
- setEditable(boolean)

TextArea : this class provides a box in which user can type multi lined text.

CH : Object -> Component -> TextComponent -> TextArea

const :

- TextArea()
- TextArea(int r, int c)
- TextArea(String def_txt, int r, int c)
- TextArea(String def_txt, int r, int c, int scrollbars)

the scrollbars are specified with built in static variables of the same class viz.,

SCROLLBARS_BOTH, SCROLLBARS_HORIZONTAL_ONLY,
SCROLLBARS_VERTICAL_ONLY, SCROLLBARS_NONE

methods :

- setText(String)
- getText()
- append(String)

Choice : this class provides a drop down list of ITEMS, amongst which user can select only one.

CH : Object -> Component -> Choice

const :

Choice()

methods :

add(String item)
getSelectedItem()
getSelectedIndex()
getItem(int index)
select(int index)
remove(int index)
removeAll()

List: this class provides a scrollable list of ITEMS.

CH : Object -> Component -> List

const :

List()

methods :

add(String item)
getSelectedItem()
getSelectedIndex()
getItem(int index)
select(int index)
remove(int index)
removeAll()

Checkbox : this class provides a square shaped box with a label, these can be used to provide multiple options to user amongst which user can select multiple.

CH : Object -> Component -> Checkbox

const :

Checkbox()

Checkbox(String label)

methods :

setLabel(String)
getLabel()
setState(boolean)
getState()

Q. There can be a 10 marks question on entire awt or any 2 to 3 classes.

Or

There can be short answer questions on any 1 or 2 classes.