Prepared By Abhiyan Dev Rimal
https://github.com/abhirimal/be-software-notes

Chapter = 2
OOP in Java

1) What is constructor. Explain with example.

Ans: A constructor is a block of codes similar to the method. It is called when the instance of the class is created. A constructor initializes an object immediately upon creation.
A constructor has the same name as that of the class and doesnot have any return type.

// Syntax

```
public class Test {
        Test() {
                // constructor body
                }
        }
```

Here, Test() is the constructor and it has the same name as that of the class and doesn't have a return type.

Scanned with CamScanner

PAGE :
DATE : / /

Example

```
public class Example {
    int x;
    Example() {                    // Constructor
        x = 5;
    }
public static void main (String[] args) {
        Example obj = new Example();
        System.out.println(obj);
    }
}
```

output : 5

PAGE :
DATE : / /

3) Parameterized constructor

→ Constructors can also take parameters, which is used to initialize attributes.

Example:

```
public class Example {
    int x;
    Example(int y) {
        x=y;
    }
    public static void main(String[] args){
        Example obj= new Example(5);
        System.out.println(obj.x);
    }
}
```

Output
5

Ans

Q)

→ The use of this operator is mandatory to solve the problem of instance variable hiding.

Output = 10

If this keyword was not used then output would be 0.

Scanned with CamScanner

Instance variable is for object.
This keyword refers to object.
So 'this' refers to instance variable

PAGE :
DATE : / /

Q)

113) This key word (Instance variable hiding)

In java we can have a local variable and instance variables of same name. However, ~~the~~ when the local variable has the same name as an instance variable, the local variable hides the instance variable. In this case ~~we use~~ "this" ~~keyword to refer to the we with~~ console shows error/ gives wrong output.

To solve this problem we use "this" keyboard to refer to the current class instance variable.

Example

```
class Test {
    int i;
    void setValues(int i) {
        this.i = i;          // this refers to instance
                                variable
    }
    void show() {
        System.out.println(i);
    }
}
class XYZ {
    public static void main(String[] args) {
        Test t = new Test();
        t.setValues(10);
        t.show();
    }
}
```

Scanned with CamScanner

PAGE :
DATE : / /

This keyword can be used in various contexts as given below:-

1) to refer instance variable of current class.
2) To invoke or initiate current class constructor.
3) can be passed as an argument in the method call.
4) can be passed as an argument in the constructor call.
5) can be used to return the current class instance.

4) Finalize keyword

Ans: ~~Java automatically deallocates~~
Java performs deallocation automatically with the help of garbage collection. Garbage collection free ups the memory that is not in use.

Finalize is a method which is used to perform clean up processing just before object is garbage collected.

This method is called by garbage collector before an object is completely destroyed. As sometimes, an object might have to complete some important task like closing an open connection, freeing up a resource, etc. before it gets destroyed. If these tasks are not done it might decrease the efficiency of the program. Thus, the garbage collector calls it for the objects that aren't referenced anymore and have been marked

Scanned with CamScanner

PAGE :
DATE : / /

for garbage collection.

//

```
public class A {
    public void finalize()
    System.out.println("object is destroyed");

    public static void main(String[] args) {

        A a = new A();
        test = null;
        System.gc();
    }
}
```

Scanned with CamScanner

PAGE :
DATE : / /

5) Access modifier

⇒ There are four types of access modifier. They are:

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 default modifier is only within the package. It cannot be accessed from outside the ~~class~~ package. If you do not specify any access level it will be default.

3) protected : The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

4) Public : The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Scanned with CamScanner

PAGE :
DATE : / /

6) ~~Polymorphism In Java~~ Inheritance in Java

→ Inheritance can be defined as the process where one class acquires the properties of another class. The class which inherits the properties of other is known as subclass/derived class/child class and the class whose properties are inherited is known as superclass/baseclass/parent class.

extend keyword is used to inherit the properties of a class

Syntax:-

```
class super {
    ...
    ...
}
class Sub extends super {
    ...
    ...
}
```

Inheritance helps in reusability. It represents is-a relationship which is known as parent-class relationship.

→ it is used for method overloading (so runtime polymorphism can be achieved)

→ for code reusability.

Scanned with CamScanner

PAGE :
DATE : / /

~~class~~

//

```
class Employee {
    ~~int id;~~
    ~~int~~
    String name="Ram";
    float salary = 40000;
}

class Programmer extends Employee {
    int bonus = 10000;
    public static void main(String[] args) {
    Programmer p = new Programmer();
    System.out.println("Name:" + p.name);
    System.out.println("Salary:" + p.salary);
    System.out.println("Bonus:" + p.bonus);
    }
}
```

Output

Name: Ram

Salary: 40000

Bonus: 10000

Scanned with CamScanner

PAGE :
DATE : / /

7) Polymorphism in Java

-> The word polymorphism means ~~array~~ having many form. Polymorphism is the ability of a message to be displayed in more than one form.

1) Compile Time Polymorphism
2) Runtime Polymorphism

2) Runtime Polymorphism
-> It is also known as Dynamic Method Dispatch.
-> It is a process in which a call to an overridden method is resolved at runtime rather than compile time.
-> In this process, an overridden method is called through the reference variable of a super class. The determination of the method to be called is based on the object being referred to by the reference variable.

Scanned with CamScanner

PAGE :
DATE : / /

Eg:- Overriding

```
class Animal {
    public void animalSound() {
        System.out.println("The animal makes a
                sound");
    }
}

class Pig extends Animal(){
    public void animalSound() {
        System.out.println("The pig says:
                wee wee");
    }
}

class Dog extends Animal(){
    public void animalSound() {
        System.out.println("The dog says:
                bow bow");
    }
}

class Main {
    public static void main(String[] args) {
        Animal myAnimal = new Animal();
        Animal myPig = new Pig();        <-
        Animal myDog = new Dog();        <-
    myAnimal.animalSound();
    myPig.animalSound();
    myDog.animalSound();
} }
```

Scanned with CamScanner

PAGE :
DATE : / /

8.) Super keyword

→ The super keyword in Java is a reference variable which is used to refer immediate parent class object.

Wherever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of super keyword

1. Super can be used to refer immediate parent class instance variable.
2. Super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

The ~~super~~ keyword 'super' come into the picture with the concept of inheritance.

Scanned with CamScanner

PAGE :
DATE : / /

```
// Example of super keyword

class Animal {
    String color = "white";
  }
class Dog extends Animal {
    String color = "black";
  void printColor() {
    System.out.println(color);
    System.out.println(super.color);
  }
}

class TestSuper {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.printColor();
    }
}
```

prints color of Dog class

print color of Animal class

Scanned with CamScanner

PAGE :
DATE : / /

9) ~~Interface~~ Abstract keyword

→ Abstraction is a process of hiding the implementation details and showing only functionality to the user.

There are two ways to achieve abstraction in java. They are
1. Abstract class
2. Interface

~~b) Abstract class & method~~

→ Abstract keyword is a non-access modifier, used for classes and methods.

a) Abstract class
→ It is a restricted class that cannot be used to create objects (to access it it must be inherited from another class).

b) Abstract method
→ It can only be used in abstract class, and it does not have a body. The body is provided by the sub class (inherited from).

Scanned with CamScanner

PAGE :
DATE : / /

Example:

```
abstract class Animal {
    public abstract void animalSound();
    public void sleep () {
        System.out.println("zzz");
    }
}
```

from the example above, it is not possible to create an object of the Animal class.

Animal is an abstract class that contains only one abstraction method ~~sleep~~ animalSound.

Scanned with CamScanner

PAGE :
DATE : / /

```
//Example 2
abstract class Animal {
    public abstract void animalSound();
    public void sleep() {
        System.out.println("zzz");
    }
}

class Pig extends Animal {
    public void animalSound() {
        System.out.println("The pig says: wee wee");
    }
}

class Main {
    public static void main(String[] args) {
        Pig myPig = new Pig();
        myPig.animalSound();
        myPig.sleep();
    }
}
```

Animal is an abstract class that contains only one abstraction method animalSound. Its implementation ~~method~~ is provided by the another Animal class.

Scanned with CamScanner

PAGE :
DATE : / /

~~The abstract keyword is used~~

Q) Abstract keyword

→ The abstract keyword is used to achieve abstraction in Java. It is a non-access modifier which is used to ~~make~~ create abstract class and method but it can't be used on a variable.
The role of an abstract class is to contain abstract methods. However, it may ~~ss~~ also contain non-abstract methods. The method which is declared with abstract keyword and doesn't have any implementation is known as an abstract method.

Scanned with CamScanner

PAGE :
DATE : / /

Example:-

```
abstract class A {
    abstract void m1();
    void m2()
      {
        System.out.println("This is a concrete
                            method.");
      }
    }

class B extends A {
    void m1() {
        System.out.println("B's implementation
                            of m2");
      }
    }

public class Main
    {
        public static void main (String args[]){
            B b = new B();
            b.m1();
            b.m2();
        }
    }
```

Output: B's implementation of m2.
This is a concrete method.

Scanned with CamScanner

PAGE :
DATE : / /

10) Interface

→ An interface is a blueprint of class. It has static constants and abstract method. The interface is a mechanism to achieve abstraction. Interfaces specify what a class must do and not how.

We use interface keyword to declare an ~~implem~~ interface. To implement interface we implements keyword.

Importance of interface

1. It is used to achieve total abstraction.
2. Since Java does not support multiple inheritance in case of class, but by using interface it can achieve multiple inheritance.
3. It can be used to achieve loose coupling.

Example:

```
interface printable {
    void print();
    }

class draw implements printable {
    public void print() {
        System.out.println("Hello");
    }

    public static void main(String[] args) {
        draw obj = new draw();
        obj.print();
    }
}
```

Scanned with CamScanner

PAGE :
DATE : / /

To access the interface methods, the interface must be "implemented" by another class with the implements keyword (instead of extends).

```
// Interface
interface Animal {
    public void animalSound();
    public void sleep();
}

class pig implements Animal {
    public void animalSound() {
        System.out.println("The pig says: wee wee");
    }
    public void sleep() {
        System.out.println("zzz");
    }
}

class Main {
    public static void main(String[] args) {
        Pig mypig = new Pig();
        mypig.animalSound();
        mypig.sleep();
    }
}
```

Output:

The pig says: wee wee
zzz

Scanned with CamScanner

PAGE :
DATE : / /

11)

Abstract class	Interface
An abstract class have both abstract and non-abstract method.	An interface can have only abstract method
It doesn't support multiple inheritance	It supports multiple inheritance
It can have final, non-final, static and non-static variables.	It has only static and final variables.
~~The~~ An abstract keyword is used to declare an abstract class.	An interface keyword is used to declare interface
It can extend another Java class and implement multiple Java interfaces.	It can extend another Java interfaces only.
~~It can to contains that~~ It can have class members like private, protected, etc.	Members of Java interface are public by default.
An abstract class is extended by using the extends keyword.	An interface is implemented by using the implements keyword.

Scanned with CamScanner

PAGE :
DATE : / /

Example	Example
public abstract class Shape {	public interface Drawable {
public abstract void draw ();	void draw();
}	}

Example of abstract class

```
public ~~static~~ abstract class Bank {
        abstract void money();
    }
    public class SavingAccount extends Bank {
        void money() {
            System.out.println("Saving amount ~~Account~~ Rs 10000");
        }
    }
    public class CheckingAccount extends Bank {
        void money() {
            System.out.println("Checking amount : Rs.15,000");
        }
    }
    public class BankApp {
        public static void main (String[] args) {
            SavingAccount sa = new SavingAccount();
            sa.money();
            CheckingAccount ca = new CheckingAccount();
            ca.money();
        }
    }
```

Scanned with CamScanner

PAGE :
DATE : / /

Examples of Interface

```
public interface Car {
    void start();
    void stop();
}
public class Tesla implements Car {
    public void start() {
        System.out.println("Start car");
    }
    public void stop() {
        System.out.println("Stop car");
    }
}

public class TeslaTestDriving {
    public static void main (String[] args){
        Tesla tesla = new Tesla();
        Tesla.stop();
        Tesla.start();
    }
}
```

Scanned with CamScanner

PAGE :
DATE : / /

Chapter-2
Interface

1) Why is multiple inheritance not supported in Java? Provide a simple code example to support your answer.

Ans Java doesnot support multiple inheritance in order to prevent ambiguity. Multiple inheritance is a feature of object oriented concept, where a class can inherit properties of more than one parent class. The problem occurs when there exists methods with same signature in both the parent classes and child class. On calling the method, the compiler cannot determine which class method to be called and even on calling which call methods gets the priority.

Example Code

```
//first Parent class
    class Parent1
  {
      void fun()
        {
            System.out.println("Parent1");
        }
  }
```

Scanned with CamScanner

PAGE :
DATE : / /

```
//Second Parent class

class Parent2 {
    {
        void fun()
        {
            System.out.println("Parent2");
        }
    }

//Error: Test is Inheriting from multiple class

class Test extends Parent1, Parent2
    {
        public static void main (String[] args)
        {
            Test t=new Test();
            t.fun();
        }
    }
```

Output: Compiler Error

from the code, we see that, on calling the method fun() using Test object will cause complications such as weather to call Parent1's fun() or Parent2's fun() method.

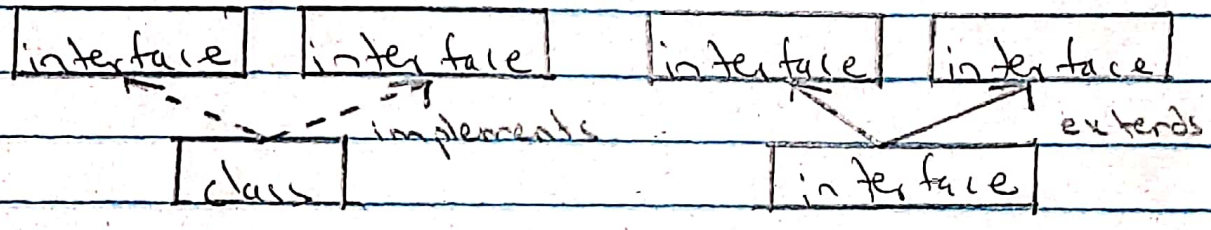
Scanned with CamScanner

PAGE :
DATE : / /

2) Explain the use of interface to achieve multiple inheritance.

Ans: An interface in java is a blueprint of a class. It has static constant and abstract methods. The interface in Java is a mechanism to achieve abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.



Multiple Inheritance in Java

Java does not support multiple inheritance in order to prevent ambiguity. Consider a case where class B extends class A and class C. Both class A and C have the same method display(). Now, java compiler cannot decide, which display method it should inherit. To prevent such situation, multiple inheritances is not allowed in java.

Scanned with CamScanner

PAGE :
DATE : / /

// multiple Inheritance using Interface

```
interface Printable {
    void print();
}
interface Showable {
    void show();
}
class Example implements Printable, Showable {
    public void print() {
        System.out.println("Hello World");
    }
    public void show() {
        System.out.println("Hello Nepal");
    }

    public static void main(String[] args) {
        Example obj = new Example();
        obj.print();
        obj.show();
    }
}
```

Output: Hello World
Hello Nepal

Scanned with CamScanner

PAGE :
DATE : / /

12) Package

Ans: A java package is a group of similar types of classes, interfaces and sub-packages. Package in java can be categorized in two form, built-in package and user-defined package. There are many ~~user defined~~ built-in packages such as java.lang, javax.swing, io, etc.

Advantages of java package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2) It provides access protection.
3) It ~~provide~~ removes naming collision.

The package keyword is used to create a package in java.

A.java

```
package pack;
public class A {
    public void msg() {
        System.out.println("Hello");
    }
}.
```

P.T.O →

PAGE :
DATE : / /

// this program will import previous package

```
package mypack;
import pack.*;

class B {
    public static void main(String[] args){
        A obj = new A();
        obj.msg();
    }
}
```

Output: Hello

Scanned with CamScanner

PAGE :
DATE : / /

13) Reflection API

= Reflection is an API which is used for examining or modifying the run time behavior of a class methods interfaces at runtime.

-> The required classes for reflection are provided under java.lang.reflect package.

-> Reflection gives us information about the class to which an object belongs ~~to~~ and also the methods of that class which can be executed by using the object.

-> Through reflection we can invoke methods at runtime irrespective of the access specifier used with them.

Reflection can be used to get information about -

1. Class

-> The getClass() method is used to get the name of the class to which an object belongs.

2. Constructors

-> The getConstructors() method is used to get the public constructors of the class to which an object belongs.

3. Methods

-> The getMethods() method is used to get the public methods of the class to which an object belongs.

Scanned with CamScanner

PAGE :
DATE : / /

Advantages of using Reflection:

1) Extensibility features
-> An application may make use of external user-defined classes by creating instances of extensibility objects using their fully qualified names.

2) Debugging & testing tools
-> Debuggers use the property of reflection to examine private members on classes.

~~Disad~~ Drawbacks :-
1) Performance Overhead
-> They ~~are~~ have slower performance.

2) Exposure of Internals
-> Reflective code breaks abstraction and therefore may change behavior with upgrades of the platform.

Scanned with CamScanner