JAVA CHAPTER 4 :-LAMBDA EXPRESSION

PROGRAM_1:- Anonymous_class

```
package Chapter_4;
class A{
    public void show(){
        System.out.println("this is class A");
    }
}
public class Anonymous_class {
    public static void main(String[] args) {
        A obj = new A(){
            public void show(){
                System.out.println("ths is class B");
            }
        };
        obj.show();
    }
}
```

PROGRAM 2:- Functional_Interfaces_default_method

```
package Chapter_4;
 interface A {
    default void abc(){
        System.out.println("this is A class");
    }
}
class demo implements A{
    public void show(){
        System.out.println("This is demo class");
    }
}
public class Functional_Interfaces_default_method {
    public static void main(String[] args) {
        demo d= new demo();
        d.abc();
        d.show();
    }
}
```

PROGRAM 3:- Lambda

```
interface Abc{
    void show();
}
class Lambda{
    public static void main(String args[]){
        Abc obj =() ->System.out.println("-> this is lambda 
expresion");
        obj.show();
    }
}
```

PROGRAM 4:- NESTED_CLASS

```
package Chapter_4;

class OuterClass

{

    static int x = 10;

    int y = 20;

    private int outer= 30;

    class InnerClass

    {

        void display()

        {

            System.out.println("x = " + x);

            System.out.println("y = " + y);

            System.out.println("outer= " + outer);


        }

    }

}

public class Nested_class

{

    public static void main(String[] args)

    {

        OuterClass outerObject = new OuterClass();

        OuterClass.InnerClass innerObject = outerObject.new
InnerClass();
```

```
        innerObject.display();

    }
}
```

PROGRAM 5 :- Passing_Argument_IN_LAMBDA

```
interface Test3 {
    void print(Integer p1, Integer p2);
}


class Passing_Argument {
    static void fun(Test3 t, Integer p1, Integer p2)
    {
        t.print(p1, p2);
    }
    public static void main(String[] args)
    {
         fun((p1, p2)-> System.out.println(p1 + " " + p2),10, 20);
    }
}
```

POINT :-1

```
//4.1 Nested Classes and Inner Classes
class point_1{
    public static void main(String[] args) {
        /*
           what is nested class :-1. A class which is declared or
contain inside another class called nested class
                                  2.A nested class is also a member of
its enclosing class.
                                  3.The scope of a nested class is
bounded by the scope of its enclosing class.
                                    Thus in below example, class
NestedClass does not exist independently of class OuterClass.
                                  4.Nested classes are divided into
two categories:
                                        1|static nested class : Nested
classes that are declared static are called static nested classes.
                                        2|inner class : An inner class
is a non-static nested class.
                                  5.syntax=
                                                class OuterClass{
                                                    class NestedClass
                                                    {
                                                       //code
                                                    }
                                                }
         */
        /*
```

Program:-

```
class OuterClass
{
    static int x = 10;
    int y = 20;
    private int outer= 30;
    class InnerClass
    {
        void display()
        {
            System.out.println("x = " + x);
            System.out.println("y = " + y);
            System.out.println("outer= " + outer);

        }
    }
}
public class Nested_class
{
    public static void main(String[] args)
    {
        OuterClass outerObject = new OuterClass();
        OuterClass.InnerClass innerObject = outerObject.new
InnerClass();

        innerObject.display();
    }
}
          */
    }


}
POINT :- 2
package Chapter_4;
//4.2 Anonymous Inner Classes
class A{
    public static void main(String[] args) {
        /*
        what is Anonymous-
          • Anonymous class is nothing but a class without any name.
          • They are used to override a class method or interface.
          • Anonymous classes in Java help us to write more concise and
readable code
          */
        /*
         Program:-
               package Chapter_4;
                class A{
                    public void show(){
                        System.out.println("this is class A");
                    }
                }
                public class Anonymous_class {
                    public static void main(String[] args) {
                        A obj = new A(){
                            public void show(){
                                System.out.println("ths is class B");
                            }
                        };
                        obj.show();
                    }
                }
            */
        }
    }
```

POINT:- 3

4.3 Default Methods and Functional Interfaces

- Default methods enable us to add new functionality to existing interfaces.
- This feature was introduced in java 8 to ensure backward compatibility while updating an interface.
- A class implementing the interface need not implement the default methods.
- Interfaces can also include private methods for default methods to use.
- You can easily override a default method like any other method of an interface

POINT:- 4

4.4 Introduction to Lambda Expressions

1|Lambda Expressions[

»Lambda expression is a new and important feature of Java which was included in Java SE 8.

»It provides a clear and concise way to represent one method interface using an expression.

»It is very useful in collection library. It helps to iterate, filter and extract data from collection.

»It saves a lot of code. In case of lambda expression, we don't need to define the method again for

providing the implementation. Here, we just write the implementation code.

»Java lambda expression is treated as a function, so compiler does not create .class file.

]

2|Why use Lambda Expression[

»To provide the implementation of Functional interface.

»Less coding.

]

3|Syntax-[

```
    »syntax of lambda expression =  (argument-list) -> {
                                         //body
                                    }


    »No Parameter Syntax =      () -> {
```

```
                                  //Body of no parameter lambda
                                }
    »One Parameter Syntax =      (parameter1) -> {
                                      //Body of single parameter
lambda
                                }
    »Two Parameter Syntax =     (parameter1,parameter2) -> {
                                    //Body of multiple parameter
lambda
                                }
]
```