8. INNER CLASSES

1. Nested classes:

we can define class within another class or interface. It holds the property of encapsulation.

2. Uses of Nesting:

- When we want to protect the class from outside world we create class as nested.
- We use nested classes to logically group classes and interfaces in one place so that it can be more readable and maintainable.

Additionally, it can access all the members of outer class including private data members and methods.

```
class Java_Outer_class
{
        // other body of outer class
        class Java_Nested_class
                {
                        // body of nested class
                }
}
```

Outer class can not access nested class property directly. It must use object of nested class.

3. Types of Nested Classes:

There are two types of nested classes.
1. static nested classes
2. nonstatic nested classes: also know as inner classes

3.1. Static Nested Classes:

This type of nested class is defined as static.
It can not access outer class non-static members directly. It must use object of outer class. Static nested class can access only static members of outer class. Becuase of that thses classes are seldom used.

Example:

```
class Outer1
{
        int i=20;
        public void fun() {
                System.out.println("In outer class method");
        }
        static class Inner1
        {
                int j=30;
                public void show() {
                        System.out.println("In inner class method");
                        Outer1 o=new Outer1(); //  if you want to make object of outer class here
                        o.fun();
                        System.out.println("Outer class member:"+o.i);
                }

        }//end inner class

        public void test() {
                System.out.println("In outer class test method");
                Inner1 in=new Inner1(); // if you want to make object of inner class here
                in.show();
                System.out.println("Inner class member:"+in.j);
        }
}

public class StaticNestedDemo {
        public static void main(String[] args) {

        Outer1 o=new Outer1();
        o.test();
        Outer1.Inner1 in=new Outer1.Inner1(); // if you want to make object of inner class here
        in.show();
        }
}
```

3.2. Nonstatic Nested Classes / Inner classes:
Nonstatic nested classes are also known as Inner classes. These can access outer class both static and non static members directly without using object. These classes are frequently used.

4. Types of Inner classes:

1. member inner classes
2. local inner classes
3. annonymous inner classes

1. Member Inner classes:
The class directly inside another class, but not inside any block is a member of that class.
Syntax:

```
class Outer
{
        /* members of outer class */
        class Inner
        {
                // members of inner class
        }
}
```

Example:

```
class Outer
{
        int i=20;

        public void fun() {
                System.out.println("In outer class method");
        }

        class Inner {
                int j=30;
                public void show() {
                        System.out.println("In inner class method");
                        fun();
                        System.out.println("Outer class member:"+i);
                }
        }//end inner class

        public void test() {
                System.out.println("In outer class test method");
                Inner in=new Inner(); // if you want to make object of inner class here
                in.show();
                System.out.println("Inner class member:"+in.j);
        }
}
public class InnerDemo {

        public static void main(String[] args) {
                Outer o=new Outer();

                o.test();
                Outer.Inner in=o.new Inner();//if you want to create object of inner class here
                in.show();
        }
}
```

2. Local Inner classes:
Inner classes (nonstatic) can be defined inside the method or block of Outer class. Such classes are known as Local Inner classes. The access is local. Local inner classes can not be accessed from outside of its scope. Local inner classes can not be defined static. Local inner classes can not be declared with any access modifier.

Example:

```
class Outer2
{
        int i=20;
        public void fun() {
                System.out.println("In outer class method");
        }
        public void test() {
                System.out.println("In outer class test method");

                class  Inner2
                {
                        int j=30;
                        public void show() {
                                System.out.println("In inner class method");
                                fun();
                                System.out.println("Outer class member:"+i);
                        }//end show
                }//end inner class

                Inner2 in=new Inner2(); // object should be created inside the scope only.
                in.show();
                System.out.println("Inner class member:"+in.j);
        }  //end test
}//end outer class

public class LocalInnerDemo {

        public static void main(String aregs[]) {
                                Outer o=new Outer();
                                o.test();
                                Outer2.Inner2 in=o.new Outer2.Inner2();     //this  will  be  wrong  as
local classes can not be accessed outside
```

```
          }
}
```

3. Annonymous Inner classes:

Annonymous inner classes are like local classes except that they do not have a name. A class is created but its name is decided by the compiler. It is generally used if you have to override method of class or interface.

The anonymous class expression consists of the following:

- The new operator
- The name of an interface to implement or a class to extend.
- Parentheses that contain the arguments to a constructor, just like a normal class instance creation expression. **Note**: When you implement an interface, there is no constructor, so you use an empty pair of parentheses.
- A body, which is a class declaration body. More specifically, in the body, method declarations are allowed but statements are not.

Use of annonymous inner classes:

- to use them once
- to minimize the code – as we can define body, create object and call methods at a time
- to override the methods when you can not extend the class

Suppose the scenario where two abstract classes Person and Animal both's abstract methods should be overriden in class Vampire. But as java does not support multiple inheritance, only one class can be extended. Thus in following example Animal class is extended and its method is overriden, but Person class need to be extended in local by annonymous inner class as we use it one time.

Example:

```
abstract class Animal {
          abstract void jump();
}

abstract class Person {
          abstract void eat();
}

class Vampire extends Animal
{
          /* jump method is overriden directly in class */
          public void jump() {
                              System.out.println("nice jump");
                    }
```

```
        public static void main(String args[]) {
        /* eat method need to be overriden in local annonymous class */
                (new Person()  {
                        void eat() {
                                System.out.println("nice fruits");
                                }
                        }).eat();
        }//end main
}
```

Internal local class generated by Compiler:
The following local class will be generated by compiler inside main method.

```
class TestAnonymousInner$1 extends Person
{
        TestAnonymousInner$1(){
                        //initialization
                }
        void eat() {
                System.out.println("nice fruits");
        }
}
```

The compiler itself gives name to this local class. This annonymous class will become subclass of Person class where eat method will get overriden
Anonymous classes enable you to make your code more concise. They enable you to declare and instantiate a class at the same time.

ASSIGNMENTS

1. mplement the program of Inner classes. Define Outer class as Organization and Inner class Department. Define the states and behaviours of both the classes.
2. Show the use of local inner class, and annonymous inner class in above example.