Java Inner, Outer Classes

Java Inner Classes (Nested Classes)

Java inner class or nested class is a class that is declared inside the class or interface. (like nested if statements)

The use inner classes to logically group classes and interfaces in one place to be more readable and maintainable.

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

Syntax of Inner class

```
class Java_Outer_class
{
    //code  // method memebers
    class Java_Inner_class
    {
            //code
    }
}
```

```
Class a
{
    Method 1;  user scan x and y
    Method2;

    Class b
    {
            Me 3    x and y process
            Me4
    }
}
```

Advantage of Java inner classes

There are three advantages of inner classes in Java. They are as follows:

1. Nested classes represent a particular type of relationship that is **it can access all the members (data members and methods) of the outer class,** including private.
2. Nested classes are used **to develop more readable and maintainable code** because it logically group classes and interfaces in one place only.
3. **Code Optimization**: It requires less code to write.

Difference between nested class and inner class in Java

An inner class is a part of a nested class. Non-static nested classes are known as inner classes.

Types of Nested classes

There are two types of nested classes: non-static and static nested classes. The non-static nested classes are also known as inner classes.

- Non-static nested class (inner class)
    1. Member inner class
    2. Anonymous inner class
    3. Local inner class
- Static nested class

Type	Description
Member Inner Class	A class created within class and outside method.
Anonymous Inner Class	A class created for implementing an interface or extending class. The java compiler decides its name.
Local Inner Class	A class was created within the method.
Static Nested Class	A static class was created within the class.

Nested Interface	An interface created within class or interface.

Java Member Inner class

A non-static class that is created inside a class but outside

a method is called **member inner class**.

It is also known as a **regular inner class**.

It can be declared with access modifiers like public,

default, private, and protected.

Syntax:

```
class Outer{
 //code
 class Inner{
  //code
 }  }
```

Java Member Inner Class Example

In this example, we are creating a show() method

in the member inner class that is accessing the

private data member of the outer class.

Creating a object of inner class

Outclass.innerclass objname = new outerclass.innerclass();

```
outer.inner obj = new outer.inner();
```

Compile by Outerclass.java

```
public class outer
{
private static String s= "Java";
private static int x= 10;
//Static and nested class
static class inner
{
//non-static method of the nested class
public void show()
{
System.out.println(s);
System.out.println(X);
 }
}
public static void main(String args[])
{
outer.inner obj = new outer.inner();
//invoking the method of the nested class
obj.show();
```

}

}

Instantiate Member Inner class in Java

An object or instance of a member's inner class always exists within an object of its outer class. The new operator is used to create the object of member inner class with slightly different syntax.

The general form of syntax to create an object of the member inner class is as follows:

Syntax:

OuterClassReference.**new** MemberInnerClassConstructor();

Example:

obj.**new** Inner();

Here, OuterClassReference is the reference of the outer class followed by a dot which is followed by the new operator.

Java Inner Classes

In Java, it is also possible to nest classes (a class within a class). The purpose of nested classes is to group classes that belong together, which makes your code more readable and maintainable.

To access the inner class, create an object of the outer class, and then create an object of the inner class:

Compile : outerinner.java

```
class OuterClass1
{
  int x = 10;

class InnerClass1
{
    int y = 5;
```

```
  }
}
public class outerinner
{
  public static void main(String[] args)
{
    OuterClass1 myOuterclass = new OuterClass1();

    OuterClass1.InnerClass1 myInnerclass = myOuter.new InnerClass1();
    System.out.println(myOuterclass.x);
    System.out.println(myInnerclass.y);
    System.out.println(myInnerclass.y + myOuterclass.x);
  }
}
```

Output :

10

5

15

Java Anonymous inner class

Java anonymous inner class is an inner class

without a name and for which only a single object

is created. An anonymous inner class can be useful

when making an instance of an object with certain

"extras" such as overloading methods of a

class or interface, without having to actually

subclass a class.

An anonymous inner class in Java. It should be used if you have to override a method of class or interface. Java Anonymous inner class can be created in two ways:

1. Class (may be abstract or concrete).

2. Interface

Syntax:

```
// Test can be interface,abstract/concrete class
Test t = new Test()
{
  // data members and methods
  public void test_method()
  {
    .....
    .....
  }
};
```

Java anonymous inner class example using class

```
abstract class AnonymousInner {
  public abstract void mymethod();
}

public class Outer_class {

  public static void main(String args[]) {
    AnonymousInner inner = new AnonymousInner() {
      public void mymethod() {
        System.out.println("This is an example of anonymous inner class");
      }
    };
    inner.mymethod();
  }
}
```

Ex: TestAnonymousInner.java

```
abstract class Person
{
  abstract void eat();
}

class TestAnonymousInner
{
 public static void main(String args[])
{
  Person p=new Person()
  {
  void eat()
{
System.out.println("We are eat Fruits");
}
  };
  p.eat();
 }
}
```

Output: We are eat Fruits

Internal working of given code

```
Person p=new Person()
{
void eat()
{
System.out.println("We are eat Fruits");
}
}
};
```

1. A class is created, but its name is decided by the compiler, which extends the Person class and provides the implementation of the eat() method.
2. An object of the Anonymous class is created that is referred to by 'p,' a reference variable of Person type.

Internal class generated by the compiler

```
import java.io.PrintStream;

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

Abstract Classes and Methods

Data **abstraction** is the process of hiding certain details and showing only essential information to the user.

Abstraction can be achieved with either **abstract classes** or **interfaces** (which you will learn more about in the next chapter).

The abstract keyword is a non-access modifier, used for classes and methods:

- **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
- **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

An abstract class can have both abstract and regular methods:

Interface in Java

An **interface in Java** is a blueprint of a class. It has static constants 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.(it declared with empty)

Multiple inheritance : interface :

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

Why use Java interface?

There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.
- It can be used to achieve loose coupling. (independent classes)

declare an interface?

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default. A class that implements an interface must implement all the methods declared in the interface.

Syntax:

```
interface <interface_name>{

   // declare constant fields
   // declare methods that abstract
   // by default.
}
```

Java anonymous inner class example using interface

```
interface Eatable
{
 void show();
}
```

```
class interfaceAnnonymousInner
{
 public static void main(String args[])
{
 Eatable e=new Eatable()
{
  public void show()
{
System.out.println("Welcome to D Y Patil ");
}
 };
 e.show();
 }
}
```

Output: We eat fruits

Internal working of given code

It performs two main tasks behind this code:

```
Eatable p=new Eatable()
{
void eat()
{
System.out.println("nice fruits");}
};
```

1. A class is created, but its name is decided by the compiler, which implements the Eatable interface and provides the implementation of the eat() method.
2. An object of the Anonymous class is created that is referred to by 'p', a reference variable of the Eatable type.

```
interface student
{
 void show();
}

class interface
{
 public static void main(String args[])
{
 Student s= new student();
```

```
{
  public void show()
{
System.out.println("Welcome to D Y Patil");
}
 };
s.show();
 }
}
```