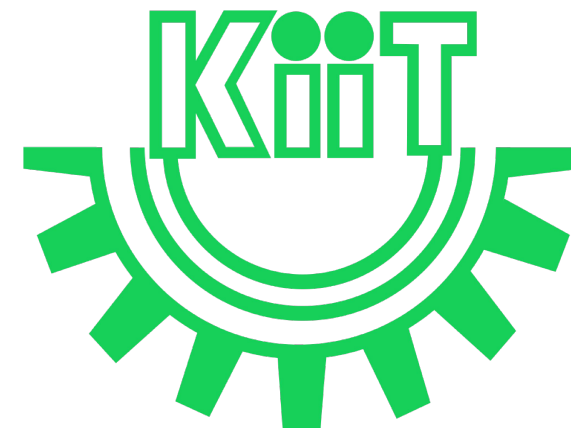


CS20004: Object Oriented Programming using Java

Lec-10



In this Discussion . . .


- Classes & Objects
 - Command line arguments
 - Recursion
- Static members
- final
- Inner class
- References



Command line arguments

- If arguments are passed at the time of running the java program, it is termed as command-line argument.
- The arguments passed from the console can be received in the java program and it can be used as an input.
- So, it provides a convenient way to check the behavior of the program for the different values. We can pass **N** (1,2,3 and so on) numbers of arguments from the command prompt.

```
class Cmdlineargs
{
    public static void main(String args[])
    {
        System.out.println("Your first argument is: "+args[0]);
    }
}
```



```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Objects and Classes$ javac Cmdlineargs.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Objects and Classes$ java Cmdlineargs Sourajit
Your first argument is: Sourajit
```

Command line arguments

- Passing multiple arguments in the console at the same time

```
class Cmdlineargs1
{
    public static void main(String args[])
    {
        for(int i=0;i<args.length;i++)
        {
            System.out.println(args[i]);
        }
    }
}
```

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Objects and Classes$ javac Cmdlineargs1.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Objects and Classes$ java Cmdlineargs1 abc 1 23
abc
1
23
```

Recursion

- Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method.

Syntax

```
data_type methodname()
{
    //code to be executed
    methodname();//calling same method
}
```

```
public class Recurexp
{
    static void recur()
    {
        System.out.println("hello");
        recur();
    }

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

Recursion

- Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method.

Syntax

```
data_type methodname()
{
    //code to be executed
    methodname();//calling same method
}
```

```
public class Recurexp
{
    static void recur()
    {
        System.out.println("hello");
        recur();
    }

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

```
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
Exception in thread "main" java.lang.StackOverflowError
```

Static Members

- Static keyword is used for memory management. It belongs to the class than instance of the class
- The static keyword can be utilized for a:
 - Variable (also known as a class variable)
 - Method (also known as a class method)
 - Block

Static Members : Java Static Variable

- It can be used to refer the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- It is a global variable shared by all instances of the class
- It gets memory only once at the time of class loading
- It cannot be used within a non-static method
- **Advantages** of static variable: It makes your program memory efficient (i.e., it saves memory).
- **Syntax:** static data_type variable_name = value; (Ex- static int a = 0;)

Static Members : Java Static Variable

```
//Program for demonstrating the usage of static variables
class Employee
{
    int empid;//instance variable
    String name;
    static String organization ="HCL Technologies";//static variable
    //constructor
    Employee(int r, String n)
    {
        empid = r;
        name = n;
    }
    //method to display the values
    void display ()
    {
        System.out.println(empid+" "+name+" "+organization);
    }
}
//Class for showing values of different objects
public class Staticvarsexp
{
    public static void main(String args[])
    {
        Employee s1 = new Employee(111,"Karan");
        Employee s2 = new Employee(222,"Aryan");
        //we can change the organization of all objects by the single line of code
        //Employee.organization="Microsoft Research";
        s1.display();
        s2.display(); }}

```

- For suppose there are 50000 employees in the organization. Now all instance data members will get memory each time when the object is created.
- All employees have their unique empid and name, so instance data member is good in such case.
- Here, "organization" refers to the common property of all objects. If we make it static, this field will get the memory only once.
- **Java static property is shared to all objects.**

Static Members : Java Static Variable

```
//Program for demonstrating the usage of static variables
class Employee
{
    int empid;//instance variable
    String name;
    static String organization ="HCL Technologies";//static variable
    //constructor
    Employee(int r, String n)
    {
        empid = r;
        name = n;
    }
    //method to display the values
    void display ()
    {
        System.out.println(empid+" "+name+" "+organization);
    }
}
//Class for showing values of different objects
public class Staticvarsexp
{
    public static void main(String args[])
    {
        Employee s1 = new Employee(111,"Karan");
        Employee s2 = new Employee(222,"Aryan");
        //we can change the organization of all objects by the single line of code
        //Employee.organization="Microsoft Research";
        s1.display();
        s2.display(); }}

```

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Static-and-final-variables$ javac Staticvarsexp.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Static-and-final-variables$ java Staticvarsexp
111 Karan HCL Technologies
222 Aryan HCL Technologies

```

Static Members : Java Static Variable

```
//Program for demonstrating the usage of static variables
class Employee
{
    int empid;//instance variable
    String name;
    static String organization ="HCL Technologies";//static variable
    //constructor
    Employee(int r, String n)
    {
        empid = r;
        name = n;
    }
    //method to display the values
    void display ()
    {
        System.out.println(empid+" "+name+" "+organization);
    }
}
//Class for showing values of different objects
public class Staticvarsexp1
{
    public static void main(String args[])
    {
        Employee s1 = new Employee(111,"Karan");
        Employee s2 = new Employee(222,"Aryan");
        Employee.organization="Microsoft Research";
        s1.display();
        s2.display(); }}

```

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Static-and-final-variables$ javac Staticvarsexp1.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Static-and-final-variables$ java Staticvarsexp1
111 Karan Microsoft Research
222 Aryan Microsoft Research

```

Static Members : Java Static Variable with example of counters

```
//Java Program to demonstrate the use of an instance variable
//which get memory each time when we create an object of the class.
class Counter
{
    int count=0;//will get memory each time when the instance is created

    Counter()
    {
        count++;//incrementing value
        System.out.println(count);
    }

    public static void main(String args[])
    {
        //Creating objects
        Counter c1=new Counter();
        Counter c2=new Counter();
        Counter c3=new Counter();
    }
}
```

Static Members : Java Static Variable with example of counters

```
//Java Program to demonstrate the use of an instance variable
//which get memory each time when we create an object of the class.
class Counter
{
    int count=0;//will get memory each time when the instance is created

    Counter()
    {
        count++;//incrementing value
        System.out.println(count);
    }

    public static void main(String args[])
    {
        //Creating objects
        Counter c1=new Counter();
        Counter c2=new Counter();
        Counter c3=new Counter();
    }
}
```

- In this example, we have created an instance variable named count which is incremented in the constructor.
- Since instance variable gets the memory at the time of object creation, each object will have the copy of the instance variable.
- If it is incremented, it won't reflect other objects. So each object will have the value 1 in the count variable.

Static Members : Java Static Variable with example of counters

```
class Counter
{
    int count=0;//will get memory each time when the
instance is created

    Counter()
    {
        count++;//incrementing value
        System.out.println(count);
    }

    public static void main(String args[])
    {
        //Creating objects
        Counter c1=new Counter();
        Counter c2=new Counter();
        Counter c3=new Counter();
    }
}
```

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Static-and-final-variables$ java Counter
1
1
1
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Static-and-final-variables$
```

Static Members : Java Static Variable with example of counters

- However, as static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

```
//Java Program to illustrate the use of static variable which
//is shared with all objects.
class Counter2
{
    static int count=0;//will get memory only once and
    retain its value

    Counter2()
    {
        count++;//incrementing the value of static
variable
        System.out.println(count);
    }
    public static void main(String args[])
    {
        //creating objects
        Counter2 c1=new Counter2();
        Counter2 c2=new Counter2();
        Counter2 c3=new Counter2();
    }
}
```

Static Members : Java Static Variable with example of counters

- However, as static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

```
//Java Program to illustrate the use of static variable which  
//is shared with all objects.
```

```
class Counter2
```

```
{
```

```
    static int count=0;//will get memory only once and  
    retain its value
```

```
    Counter2()
```

```
{
```

```
        count++;//incrementing the value of static
```

```
variable
```

```
        System.out.println(count);
```

```
}
```

```
public static void main(String args[])
```

```
{
```

```
    //creating objects
```

```
    Counter2 c1=new Counter2();
```

```
    Counter2 c2=new Counter2();
```

```
    Counter2 c3=new Counter2();
```

```
}
```

```
}
```

```
itp@itp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Static-and-final-variables$ javac Counter2.java
```

```
itp@itp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Static-and-final-variables$ java Counter2
```

```
1  
2  
3
```


Static Members : Java static method

- It belongs to the class rather than object of a class. It can be invoked without the need for creating an instance of a class.
- It can access static data member & can change its value.
- It can only call static methods & access static variables, i.e., in other words, the static method can not use non static data member or call non-static method directly.
- It cannot refer to this or super in any way, i.e., in other words, this and super cannot be used in static context.
- **Syntax:** `static return_type method_name() { ... }`

Static Members : Java static method

```
class Employee{
    int rollno;
    String name;
    static String organization = "HCL Technologies";
    //static method to change the value of static variable
    static void change(){
        organization = "Microsoft Research";
    }
    Employee(int r, String n){
        rollno = r;
        name = n;
    }
    //method to display values
    void display(){System.out.println(rollno+" "+name+"
"+organization);}
}
public class Staticmethexp
{
    public static void main(String args[]){
        Employee.change();//calling change method
        //creating objects
        Employee s1 = new Employee(111,"Tarun");
        Employee s2 = new Employee(222,"Pranjal");
        Employee s3 = new Employee(333,"Sambit");
        //calling display method
        s1.display();
        s2.display();
        s3.display(); }}
}
```

Static Members : Java static method

```
class Employee{
    int rollno;
    String name;
    static String organization = "HCL Technologies";
    //static method to change the value of static variable
    static void change(){
        organization = "Microsoft Research";
    }
    Employee(int r, String n){
        rollno = r;
        name = n;
    }
    //method to display values
    void display(){System.out.println(rollno+" "+name+"
"+organization);}
}
public class Staticmethexp
{
    public static void main(String args[]){
        Employee.change();//calling change method
        //creating objects
        Employee s1 = new Employee(111,"Tarun");
        Employee s2 = new Employee(222,"Pranjal");
        Employee s3 = new Employee(333,"Sambit");
        //calling display method
        s1.display();
        s2.display();
        s3.display(); }}

```

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Static-and-final-variables$ javac Staticmethexp.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Static-and-final-variables$ java Staticmethexp
111 Tarun Microsoft Research
222 Pranjal Microsoft Research
333 Sambit Microsoft Research

```

Static Members : Java static method

```
class Staticmethexp1
{
    int a=40;//non static
    public static void main(String args[])
    {
        System.out.println(a);
    }
}
```

Static Members : Java static method

```
class Staticmethexp1
{
    int a=40;//non static
    public static void main(String args[])
    {
        System.out.println(a);
    }
}
```

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Static-and-final-variables$ javac Staticmethexp1.java
Staticmethexp1.java:6: error: non-static variable a cannot be referenced from a static context
        System.out.println(a);
                           ^
1 error
```

Static Members : Java static method - Why is the Java main method static?

It is because the object is not required to call a static method. If it were a non-static method, JVM creates an object first then call main() method that will lead the problem of extra memory allocation.

Static Members : Java static block

- This is where the static variables are initialized
- It is executed before the main method at the time of classloading.
- It is executed exactly once, when the class is first loaded.
- **Syntax:** `static{.....}`

```
class Staticblockexp
{
    //static block
    static
    {
        System.out.println("static block is invoked");
    }
    public static void main(String args[])
    {
        System.out.println("Hello main");
    }
}
```

Static Members : Java static block

- This is where the static variables are initialized
- It is executed before the main method at the time of classloading.
- It is executed exactly once, when the class is first loaded.
- **Syntax:** `static{.....}`

```
class Staticblockexp
{
    //static block
    static
    {
        System.out.println("static block is invoked");
    }
    public static void main(String args[])
    {
        System.out.println("Hello main");
    }
}
```

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Static-and-final-variables$ javac Staticblockexp.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Static-and-final-variables$ java Staticblockexp
static block is invoked
Hello main
```


final keyword

- The final keyword in java is used to restrict the user. The java final keyword can be applied with:
 - variables
 - method
 - class

final keyword : Java final variable

- A final variable which has no value associated with it is called *blank final variable* or uninitialized final variable.
- It can be initialized in the constructor only.
- The blank final variable can also be static also, which can then only be initialized in the static block.

final keyword : Java final variable

- If we declare any variable as final, then we cannot change the value of final variable
(It will then becomes a constant).

```
class Foodfinal
{
    final String best_food="Pav bhaji";//final variable
    void update()
    {
        best_food="pizza";
    }
    public static void main(String args[])
    {
        Foodfinal obj=new Foodfinal();
        obj.update();
    }
}
```

final keyword : Java final variable

- If we declare any variable as final, then we cannot change the value of final variable (It will then becomes a constant).

```
class Foodfinal
{
    final String best_food="Pav bhaji";//final variable
    void update()
    {
        best_food="pizza";
    }
    public static void main(String args[])
    {
        Foodfinal obj=new Foodfinal();
        obj.update();
    }
}
```

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Static-and-final-variables$ javac Foodfinal.java
Foodfinal.java:6: error: cannot assign a value to final variable best_food
        best_food="pizza";
        ^
1 error
```

final keyword : Java final method

- If a method is made as final, then we cannot override it.

```
class Foodfinal1
{
    final void update()
    {
        System.out.println("pizza");
    }
}
class Foodfinal2 extends Foodfinal1
{
    void update()
    {
        System.out.println("chole bhature");
    }
    public static void main(String args[])
    {
        Foodfinal2 ff= new Foodfinal2();
        ff.update();
    }
}
```

final keyword : Java final method

- If a method is made as final, then we cannot override it.

```
class Foodfinal1
{
    final void update()
    {
        System.out.println("pizza");
    }
}
class Foodfinal2 extends Foodfinal1
{
    void update()
    {
        System.out.println("chole bhature");
    }
    public static void main(String args[])
    {
        Foodfinal2 ff= new Foodfinal2();
        ff.update();
    }
}
```

```
tipt@tipt-HP-Notebook:~/Desktop/Web-Technology/class/Java-Static-and-final-variables$ javac Foodfinal2.java
Foodfinal2.java:10: error: update() in Foodfinal2 cannot override update() in Foodfinal1
    void update()
      ^
    overridden method is final
1 error
```

final keyword : Java final class

- If any class is declared as final, then we cannot extend it.

```
final class Foodfinal3
{
}

class Foodfinal4 extends Foodfinal3
{
    void update()
    {
        System.out.println("running safely with 100kmph");
    }

    public static void main(String args[])
    {
        Foodfinal4 ff= new Foodfinal4();
        Foodfinal4.update();
    }
}
```

```
tiit@tiit-HP-Notebook:~/Desktop/Web-Technology/class/Java-Static-and-final-variables$ javac Foodfinal4.java
Foodfinal4.java:6: error: cannot inherit from final Foodfinal3
class Foodfinal4 extends Foodfinal3
    ^
Foodfinal4.java:16: error: non-static method update() cannot be referenced from a static context
        Foodfinal4.update();
            ^
2 errors
```

final keyword : Java final class

- Can we declare a constructor final?
 - Ans - No, because constructor is never inherited.
- Final parameter: If we declare any parameter as final, we cannot change its value.

Inner class

- Java inner class is a type of nested class is a class that is declared inside the class or interface.
- They are used to logically group classes and interfaces in one place for improving readability & maintainability.
- Additionally, it can access all the members of the outer class, including private data members and methods.
- To access the inner class, we need to create an object of the outer class first, and then create an object of the inner class.

Syntax

```
class Outer_class_name
{
    //code
    class Inner_class_name
    {
        //code
    }
}
```

Advantages of Inner class

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

Java Inner Class: Need?

- Sometimes users need to program a class in such a way so that no other class can access it. Therefore, it would be better if you include it within other classes.
- If all the class objects are a part of the outer object then it is easier to nest that class inside the outer class. That way all the outer class can access all the objects of the inner class.

Nested Class Types

- An inner class is a part of a nested class. Non-static nested classes are known as inner classes.
- For inner classes, instance have access to instance members of containing class, while **static**

Nested classes instances **do not** have access to instance members of containing class.

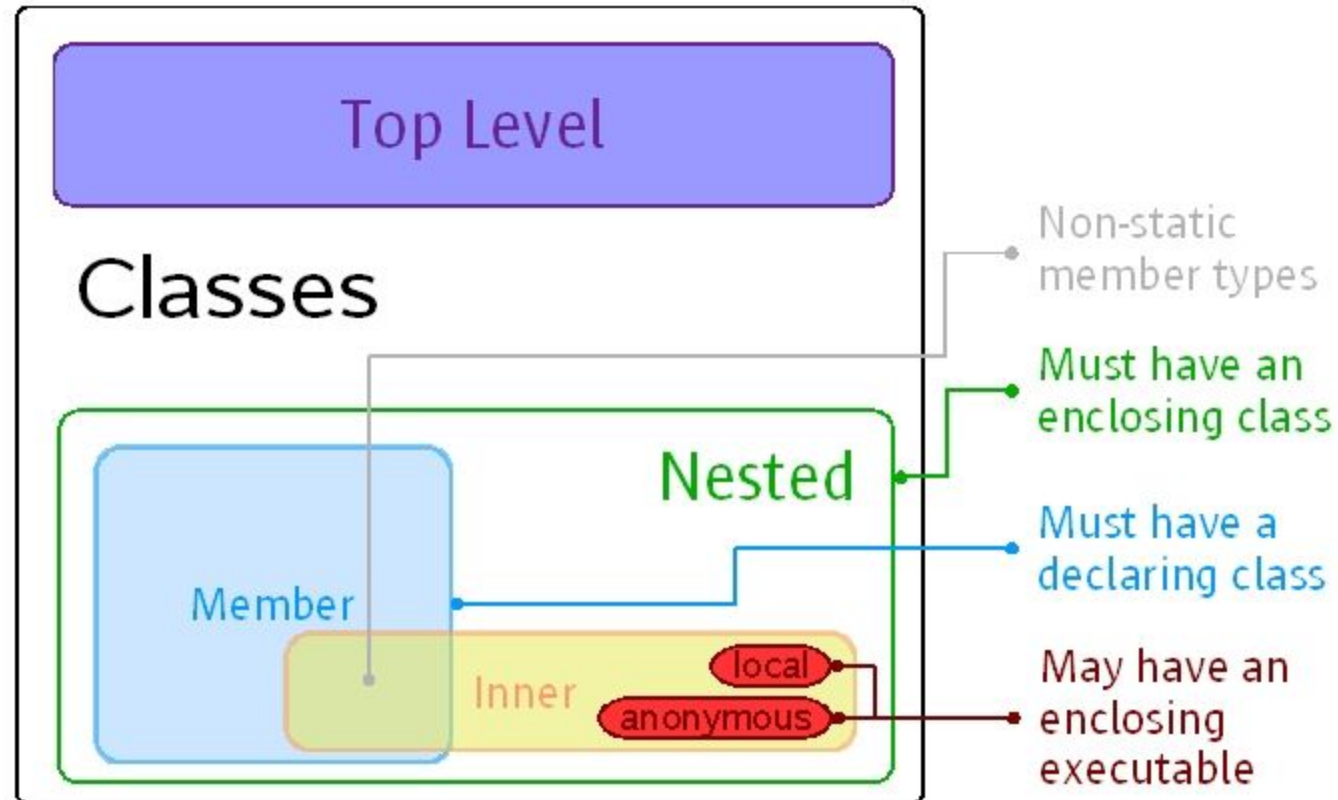
- Types of Nested Classes: **2 types**.
 - **Non-static nested class** (Inner Class)
 - **Member inner class**
 - **Anonymous inner class**
 - **Local inner class**
 - **Static nested class**

Nested Class Types

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 created within the method.
Static Nested Class	A static class created within the class.
Nested Interface	An interface created within class or interface.

Taxonomy of Classes in java

Taxonomy of Classes in the Java Programming Language



Non-static Nested Classes: Instantiating Member Inner Class

- 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:

```
Outer_Class_Reference.new Member_Inner_Class_Constructor();
```

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

Non-static Nested Classes: 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.

```
class Outermem
{
    private int data=30;
    class Inner
    {
        private int x = 100;

    }
    public static void main(String args[])
    {
        Outermem obj=new Outermem();
        Outermem.Inner in=obj.new Inner();
        System.out.println("Sum of private variables from
different classes are: "+(in.x + obj.data));
    }
}
```

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Objects and Classes$ j
avac Outermem.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Objects and Classes$ j
ava Outermem
Sum of private variables from different classes are: 130
```


Non-static Nested Classes: Private Member Inner Class

- If we don't want any outside objects to access the inner class, we need to declare the inner class as private.
- If we try to access a private inner class from an outside class, an error occurs:

```
class OuterClass
{
    int x = 10;
    private class InnerClass
    {
        int y = 5;
    }
}
public class Firstdemo
{
    public static void main(String[] args)
    {
        OuterClass myOuter = new OuterClass();
        OuterClass.InnerClass myInner = myOuter.new
InnerClass();
        System.out.println(myInner.y + myOuter.x);
    }
}
```

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Objects and Classes$ j
avac Firstdemo.java
Firstdemo.java:14: error: OuterClass.InnerClass has private access in OuterClass
        OuterClass.InnerClass myInner = myOuter.new InnerClass();
                ^
Firstdemo.java:14: error: OuterClass.InnerClass has private access in OuterClass
        OuterClass.InnerClass myInner = myOuter.new InnerClass();
                ^
2 errors
```

Non-static Nested Classes: Anonymous Inner Class

- **Unnamed classes whose instances are created in expressions and statements**
- 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.

Non-static Nested Classes: Anonymous Inner Class

- An anonymous class is a local class without a name that can be used to define and instantiate a class at the same time.
- An anonymous inner class is one that has no name and produces only one object.
- In simple words, a class that has no name is known as 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:
 - Class (may be abstract or concrete).
 - Interface

Non-static Nested Classes: Anonymous Inner Class (Syntax)

```
class ABC {  
  
    // Definition of anonymous inner class  
    Object obj = new  
    SuperType(constructorArguments)    {  
        // anonymous class body  
    }  
}
```

Here, SuperType can be a class or interface type that the anonymous class extends or implements, and constructorArguments are the arguments to the constructor of the superclass.

Non-static Nested Classes: Anonymous Inner Class

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

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Objects and Classes$ javac TestAnonymousInner.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Objects and Classes$ java TestAnonymousInner
Pineapple
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Objects and Classes$
```

Non-static Nested Classes: Local Inner Class

- A class which is created inside a method, is called as the local inner class in java.
- Local Inner Classes are the inner classes that are defined inside a block.
 - Generally, this block is a method body. Sometimes this block can be a for loop, or an if clause.
- Local Inner classes are not a member of any enclosing classes.
 - They belong to the block they are defined within, due to which local inner classes cannot have any access modifiers associated with them.

Non-static Nested Classes: Local Inner Class

- The local inner classes can be marked as final or abstract. These classes have access to the fields of the class enclosing it.
- If you want to invoke the methods of the local inner class, you must instantiate this class inside the method.

Non-static Nested Classes: Local Inner Class



```
public class Localinnerexp
{
    private int data=30;//instance variable
    void display() //method definition
    {
        class Localexp //Local inner class
        {
            void msg()
            {
                System.out.println(data);
            }
        }
        Localexp l=new Localexp();
        l.msg();
    }
    public static void main(String args[])
    {
        Localinnerexp obj=new Localinnerexp();
        obj.display();
    }
}
```

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Objects and Classes$ javac Localinnerexp.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Objects and Classes$ java Localinnerexp
30
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Objects and Classes$
```


References

1. https://www.w3schools.com/java/java_inner_classes.asp
2. <https://www.javatpoint.com/java-inner-class>
3. <https://www.topjavatutorial.com/java/inner-class-in-java/>
4. <https://blogs.oracle.com/darcy/nested,-inner,-member,-and-top-level-classes>
5. <https://www.scaler.com/topics/anonymous-class-in-java/>
- 6.