

Top-level and Inner Classes

www.techfaq360.com

Top-Level Class

- * Top-level classes can only have public, abstract, and final modifiers, and it is also possible to not define any class modifiers at all. This is called default/package accessibility.
- * Besides that, private, protected, and static modifiers cannot be used when declaring top-level classes.
- * It is an error to declare the same modifier twice; the compiler will throw a Repeated modifier exception.
- * More than one top-level class can be defined in a Java source file, but there can be at most one public top-level class declaration. The file name must match the name of the public class.
- * The file name can be anything, provided there is not a public class definition. If there is more than one class declaration in a source file, each will be compiled into separate .class files.
- * An abstract class is a class which may have incomplete implementations for its methods, some of the methods can have implementation and some others can be empty method declarations.
- * If an abstract class is going to be subclassed, all of its incomplete method implementations must be implemented in the subclass, or the subclass must be declared as abstract too; otherwise the compiler will complain.
- * A class that is declared with a final modifier cannot be subclassed. If you want your class not to be able to subclassed, use the final modifier.
- * A class, defined with an abstract modifier cannot be instantiated, because they have incomplete implementations. This type of class needs to be subclassed and have implementation for its incomplete methods before instantiation.
- * Multiple inheritance is not allowed in Java. A class can be declared to implement multiple interfaces but can only extend at most one class.
- * An interface is the same as an abstract class, but you cannot define any method implementations at all.
- * Abstract classes allow having some method implementations. Method implementations cannot be defined when an interface is concerned.
- * Interfaces are defined with the interface keyword.
- * An interface doesn't need abstract and class keywords. Member variables are implicitly defined as public static final.
- * An abstract class can have a constructor(s), but the constructor(s) must be implemented.
- * An Interface must not have a constructor at all.

Nested top-level classes and interfaces (Static nested classes/interfaces)

- * Nested top-level classes are declared within an enclosing class.
- * Nested top-level classes are always defined with a static keyword.
- * A nested top-level class/interface is defined as a static member of its enclosing class/interface. For this reason, it can be accessed or instantiated without any instance of an enclosing class. Since it is declared static, it can only access static members of enclosing class. Do not confuse nested top-level classes with top-level classes -- they are not the same.

* Nested top-level classes can not have the same name as the enclosing class; doing so will cause an exception. (same holds true for inner classes also. Gives compilation error)

* Since nested top-level classes are declared static, they act like any other static features of a class. The following example illustrates the instantiation of a nested top-level class.

```
Outer.Inner theInner= new Outer.Inner();
```

* There is no such thing as an inner interface. Because, interfaces are always implicitly static. They are always top-level, not inner. Inner classes cannot have any static modifiers at all.

```
1. class Outer {
2. //class Cinner {interface IInner {}; } //Error. Inner class can't have static members
3. static class CSinner {interface IInner {}; }; //OK
4. }
5.
6. interface IOuter {
7. class ICinner {interface IInner {};} //OK
8. static class ICSinner {interface IInner {};} //OK
9. interface IInner {interface Inner{ };} //OK
10. }
```

Inner Classes

* Declaration of a class within another class is called an inner class. It is basically a name for nested classes.

* A top-level class can be accessed directly without an instance of an enclosing class, but an inner class requires an instance of an outer class. We have used the static modifier when defining a top-level class. If a static modifier is used, there is no requirement for an enclosing class, like a top-level class. For that reason, you can think every type of inner class as a non-static class.

* The compiler will throw an error if you want to declare static initializers, because inner classes do not allow static initializer code blocks.

* Inner classes can have any access modifier, like public, protected, and private, or no access modifier, but it cannot have any static members.

* An inner class can extend or implement any class or interface, there are no restrictions or obligations.

* There are three types of inner classes.

Member Classes (Non-static inner classes)

Local Classes

Anonymous Classes

"An inner class is a nested class that is not explicitly or implicitly declared static." ('8.1.2) [1].

Member Classes (Non-static Inner Classes)

* A member class is declared as a member of an enclosing class. A member class is declared like a top-level inner class except for a static modifier. Member classes do not have static modifiers.

* Member classes require an instance of an enclosing class and every instance of a member class

holds an implicit reference to its enclosing class. The following example illustrates the instantiation of a member class.

```
Outer.Inner theInner = new Outer().new Inner();
```

```
//or
```

```
Outer theOuter = new Outer();
```

```
Outer.Inner theInner = theOuter.new Inner();
```

- * Member classes cannot have any static member unless it is a compile-time constant. (i.e. static final variables)

"Inner classes may not declare static members, unless they are compile-time constant fields .".

- * A member class can access all the fields and methods (even private) of an enclosing class.

- * A member class can be declared as public, protected, private, abstract, final, or static.

- * Note that when you declare a member class with a static modifier it will not be an inner class anymore. It will be a top-level nested class.

Local Classes

- * A local class is declared inside a method, a constructor, a static initializer, or an instance initializer.

- * A local class cannot have a static modifier. A local class is analogous to a local variable in some ways.

- * You cannot use public, protected, private, or static modifiers in a local class analogous to a local variable.

- * A local class can access all the fields and methods of the enclosing class but can only access final modifiers declared inside a method.

- * A local class cannot be accessed from outside the method.

- * Local classes cannot be declared as static, but they can be defined in a static context (for example, in a static method).

Anonymous Classes

- * An anonymous class is one declared without a name. This is a convenient solution for most situations.

- * You can create an object on the fly this way. For example, event listeners can be created by the use of anonymous inner classes.

- * Anonymous classes do not allow the use of extends and implements clauses and access modifiers.

- * An anonymous class doesn't have constructors, and it is not allowed to define a constructor. Since anonymous classes don't have names, there is no way to define a constructor.

- * Anonymous classes can either implement an interface or extend a class but cannot do both. The general form is:

```
//interface implementation
```

```
new InterfaceName() {}
```

```
//extends a class
```

```
new ClassName() {}
```

* The interface name is the name of the interface that is going to be implemented, and the class name is the name of the class that is going to be extended.

* They are not the name of our newly created class. It is the name of the class that we intend to implement or extend.