# Access Control

MAY 22, 2015

**Modifiers** fall into *two* categories:

- **Access modifiers**: `public, protected, private and default (package-level access)`.
- **Non-access modifiers**: `transient, synchronized, native, strictfp, final, abstract and static`.

## Access Modifiers

Two **types of access** are there:

- Whether method code in one class can access a member of another class
- Whether a subclass can inherit a member of its superclass

A **default member** may be accessed only if the class accessing the member belongs to the same package, whereas a **protected member** can be accessed by a subclass in the same package *(through dot operator and inheritance)* and even if it is in a different package *(through inheritance* *only).*

You cannot access a protected member using the dot (.) operator in the subclass **if the subclass is in a different package** from the parent class.

The following code snippet makes it clear:

```java
package certification;
public class Parent {
    protected int x = 9; // protected access

    protected int getX() {
        return x;
    }
}

package other; // different package
import certification.Parent;
class Child extends Parent {
    public void testIt() {
        System.out.println("x is " + x); // No problem; Child
                                          // inherits x
        Parent p = new Parent(); // Can we access x using
                                 // p reference?
        System.out.println("X in parent is " + p.x); // Compiler
                                                      // error
        System.out.println("X in parent is " + p.getX()); // Compiler
                                                           // error
    }
}
```

*NOTE: If `Child` class would have been in the same package as `Parent` then there would be no compiler error.*

**The below table gives the picture of all access modifiers:**

| Visibility | Public | Protected | Default | Private |
|---|---|---|---|---|
| From the same class | Yes | Yes | Yes | Yes |
| From any class in the same package | Yes | Yes | Yes | No |
| From a subclass in the same package | Yes | Yes | Yes | No |
| From a subclass outside the same package | Yes | Yes, through inheritance | No | No |
| From any non-subclass class outside the package | Yes | No | No | No |

## Non-Access Modifiers

**Final** is the only modifier which can be applied to local variables. It can also be used in method arguments like:

```
    // final in method arguments, can't be altered inside the method
    public Record getRecord(int fileNumber, final int recordNumber) {}
```

**Final** when applied to a method prevents it to be overridden and when applied to a class makes it un-inheritable i.e, it can never be subclassed (no class can extend it).

An **abstract method** is a method that's been declared (as abstract) but not implemented. In other words, the method contains no functional code. And the class which contains at least one of such methods is an **abstract class** and has to be declared abstract. An abstract class can **never be instantiated**.

A method can never, ever, ever be marked as **both abstract and final, or both abstract and private or both abstract and static**.

A class having even a single abstract method has to be declared **abstract** or if it extends an abstract class then it must implement all abstract methods of the superclass otherwise you have make it abstract as well.

### Comparison of modifiers on variables vs. methods:

| Local Variables | Non-local Variables | Methods |
|---|---|---|
| final | final | final |
| | public | public |
| | protected | protected |
| | private | private |
| | static | static |
| | transient | abstract |
| | volatile | synchronized |
| | | strictfp |
| | | native |

**Declarations**

« Previous

**Object oriented design**

Next »

0 Comments Java Notes

1 Login

Recommend 1 Share Sort by Best

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.

ALSO ON JAVA NOTES

**Nested Classes**

2 comments • 3 years ago

Amit Satpathy — Ram, it's really great to find your short and precise writeups on tech. Keep up the passion.

**Overloading | Java Concepts**

1 comment • 3 years ago

Anagh Hegde — public class MyTest { public static void main(String[] args) { MyTest test = new MyTest(); int i = 9; test.TestOverLoad(i); } …