**Modifier:**

Before the definition of instance variable, class or method modifiers can be placed to convey additional stipulation about the definition.

1. access control modifiers: defining the level of access or visibility

Ability to limit access among classes supports a key principle of object-orientation known as encapsulation.

-Public: all classes may access the defined aspect.

A class, method, constructor, interface etc declared public can be accessed from any other class.

However if the public class we are trying to access is in a different package, then the public class still need to be imported.

The main() method of an application has to be public. Otherwise, it could not be called by a Java interpreter (such as java) to run the class.

--public class: each public class should be defined in a separate file. Clasname.java

All other classes are allowed to construct new instances of this class.

--public method: allows to any other class to make a call

-- public instance variable: dot notation is used to directly access the variable in other classes that

has a reference to an object of this class.

-Protected:

The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected.

--protected class: access to the defined aspect is only granted to the following groups of other

Classes:

---classes that are designated as subclass of the class through inheritance.

---classes that belong to the same package of the given class.-

-Private: designates that access to a defined member of the class be granted only to code within the

class. Neither subclass nor other classes have access to such members.

Private access modifier is the most restrictive access level. Class and interfaces cannot be private.

Example: we define an instance variable of a class has private access level we are allowed to

Read or modify its value from the methods within that class in this case we provide some

Public methods to grant outside classes the behavior that depend on the current private

Instance variable.

Java default access modifier: package\_ptotected.

**Access Control and Inheritance:**

The following rules for inherited methods are enforced:

* Methods declared public in a superclass also must be public in all subclasses.
* Methods declared protected in a superclass must either be protected or public in subclasses; they cannot be private.
* Methods declared without access control (no modifier was used) can be declared more private in subclasses.
* Methods declared private are not inherited at all, so there is no rule for them.

**Non access modifiers:**

Static: to declare any variable or a method

--when a variable is static its value is associated to the class as whole rather than with individual instance.

--static variables can be used to maintain the total number of instances created.

-- static variables exist even there is no instance of the class.

--when a method is static is associated with the class not with a particular instance. It mean the method is not invoked on a particular instance of the class it’s typically invoked using the class name.

Example: java.lang package contains Math class that provides many static methods.

When you wanna invoke one of the provided methods you don’t need to create an object of the class. Math.sqrt(2)

--static methods are useful for providing behavior related to a class that need not rely on the state of the any particular instance.

Abstract: The *abstract* modifier for creating abstract classes and methods.

--Abstract method: signature is provided but without an implementation of the method body

--Any subclass of a class with abstract method is expected to provide concrete implementation

For each abstract method.

--Any class with one or more abstract methods must also be formally declared as abstract.

--Java doesn’t allow any subclass of an abstract class to be constructed.

Note:

Declaring an abstract class only means that you don't allow it to be instantiated on its own.

|  |  |
| --- | --- |
|  | Declaring a method abstract means that subclasses have to provide an implementation for that method. |

The two are separate concepts, but obviously you can't have an abstract method in a non-abstract class.

**Final:** The *final* modifier for finalizing the implementations of classes, methods, and variables

--if a base type is final is constant.

--if a reference variable is final it refers to the same object (even if the internal state of the object changes)

- if a member variable of a class defined as final it should be static as well it would be wasteful to have every instance store the identical value.

-final class: class cannot be subclassed.

-final method: cannot be overridden by subclass.