Section 11: Naming Conventions and Packages. static and final keywords

15 July 2021 19:23

Section 11: Naming Conventions and Packages. static and final keywords

Naming Conventions :-

1) Packages : lower case, unique, not start with number, can starts with .com. Example - com.rj
2) Class Names - CamelCase, Nouns, Starts with Capital Letter (Examples - ArrayList, LinkedList)
3) Interface Names - CamelCase (Example - List, Comparable, Serializable)
4) Methods Names - mixedCase, Often verbs, (Example - size(), getName(), addPlayer())
5) Constants : ALL UPPER_CASE, separate words using underscore _, declared using the final keyword. (Examples - Static final int MAX_INT)
6) Variable names - mixedCase, Meaningful and Indicative, starts with lower case letter, do not use underscore _. (Examples - I, league, BoxLength)
7) Type Parameters - Single Character, capital letters (seen in generic section). Examples - E (Element), K (Key), T (Type), V (Value)

Packages :-

Makes easy to know where to find classes and interfaces that can provide the functions provided by the package.

We can create our own code package (saved as jar) and import it in another project and add the library after that we can use all the functions in our new project. It's like creating a library and then using it in other project directly.

Creating Your Package (Library) :-

Select Package (com.rj.whatever) -> File (Project Structure) -> Artifacts (Add - click on +) -> Select Jar then From modules with dependencies -> Click OK -> Check default location and click OK.

Click Build (Build Artifacts) -> Click on Build in Action menu

Jar file is successfully created just check in folder in directory out -> artifacts -> ProjectName_jar -> jar file is here.

Using Created Package (Library) in other Projects :-

File (Project Structure) -> Libraries (Add - click on +) -> Select Java (And select the Jar file) -> Click OK -> Click OK -> Click OK

Now it's added in this project, can be cross checked in External Libraries.

Scope :-
Actually refers to the visibility of a class, member or variable.

Calling base class method of same name exist also in inner class from inner class : MainClass.this.functionName();

Access Modifiers :-

At Top Level :

only classes, interfaces and enums can exist at the top level, everything else must be included within one of these.

Public : the object is visible to all classes everywhere, whether they are in the same package or have imported the package containing the public classes.

Package-private: the object is only available within its own package (and its visible to every class within the same package). Package-private is specified by not specifying, i.e. it is the default if you do not specify public. There is not a "package-private" keyword.

Member Level :

Public :
Package-private:
Private : the object is only visible within the class it is declared. It is not visible anywhere else (including in subclasses of its class).
Protected : the object is visible anywhere in its own package (like package-private) but also in subclasses even if they are in another package.

All methods are public in interfaces because all interface methods are automatically public, so lack of an access modifier here does not imply package private.

But it's not possible to have anything except public methods in an interface.

The lack of an access modifier means the default of package private except with interface methods and variables, which are always public.

Static :

Static methods can directly be accessed using ClassName.staticMethodName(); wihout using object name.
Static methods and fields belong to the class, not to instances of the class, and as a result, can be called by referencing the class name rather than a class instance.
That's main has to be Static.

Non static methods and fields cannot be referenced from a static context. All has to be static that's why in Main when we call any other method or field that also has to static. (in their own classs only)

So as a result, Java can't allow a static method to access non-static fields or methods because they don't exist when the static Methods called but we can call the static methods from non-static once with no problems.

But static methods can access not-static fields and methods from another class because it creats instances of a class in order to do so.

Final :

Used generally to define constant values. Not exactly constants because they can be modified but only once, and any modification must be performed before class constructor finished.

Marking a class final, we can prevent our class from being subclassed.

Adding final in super class methods, prevent sub classes to override that methods.
We cannot override final methods.

All static initialization block called first in the order they define, after that constructor is called.