Java Concepts: Part – 04

In the previous parts of this article series we had covered some basics topics related to Java programming, which will give you an idea about Java programming, the topic which we had covered earlier are overview of Java, Java programming elements and Input/output handling in Java.

In this part we will see some concepts related to encapsulation in Java programming. Encapsulation is a broad and very important concept of Object-oriented programming (OOPs), and that’s why I will recommend you to learn it and explore more about it.

1. Java’s two most innovative features are:

* Packages
* Interface

1. What is a package ?

* A package is a container for the classes that are used to keep the class name space compartmentalized.
* Example: You can contain all classes related to all sorting programs in your own package.

1. Why packages?

* It allows flexibility to give same but to many classes, that is to avoid name space collision.
* The packages in Java provide mechanism for partitioning the class name space more manageable chunks.
* In fact, a package is both a naming and a visibility control mechanism.
* It supports reusability and maintainability.

1. Advantages of Packages :

* Packages provide code reusability, because a package contains group of classes.
* It helps in resolving naming collisions when multiple packages have classes with same name.
* Packages also provide the hiding of class facility. Thus, other program cannot use the classes from hidden packages.
* Access limitation can be applied with the help of packages.
* Nesting of a package, that is one package can be defined in another package in a hierarchy fashion.

1. Built-in packages in Java :

* Packages are nothing more than the way we organize files into different directories according to their functionality, usability as well as category they should belong to. A Java package is a Java programming language mechanism for organising classes into name space.
* In Java, already many predefined packages are available, those are to help programmers to develop their software in an easy way.
* Example :- “java.swing” is a package in Java providing all the basic support in developing GUI programs.

1. Packages in Java :

* Code reusability is the main philosophy of Object-oriented programming.
* To power this advantage, Java has a number of “packages called API bundle with the JDK.”
* Packages are “collection of classes and interface” to facilitate a number of readymade solutions.
* A great “knowledge of packages help as a Java developer” to master in Java solutions.
* In addition to the API, user maintains their own packages.

1. Using API packages :

* A package is a collection of classes and each class is a collection of member and methods.
* Any class as well as any member and method in a package are accessible from a Java program.
* This can be achieved in Java by “import” statement.
* There are two ways of using “import” statement.

1. With fully quantifies class name :- When it is required to access a particular class

Example : “java.lang.String”

1. With default (.\*) quantification : When it is required to access a number of classes.

Example : “java.lang”

* However, instead of importing whole package or a class it is possible to refer a class in order to instance an object.

Example : “java.util.Date toDay = new java.util.Date();

System.out.println(toDay);”

The same thing but with import statement can be done as follows

Example : “import java.util.Date;

Date toDay = new Date()”

1. Package naming conventions :

* Packages are usually defined using a hierarchical naming pattern, with levels in the hierarchy separated by periods(.).
* Although package lower in the naming hierarchy are often referred to a “sub package” of the corresponding packages higher in the hierarchy, there is no semantic relationship between packages.

1. Organizational package naming conventions :

* Package names should be “all lowercase characters” whenever possible.
* Frequently, a package name begins with the top level domain name of the organization and then the organization’s domain and then any sub domains listed in the reverse order.
* The organization can then choose a specific name for their package.

1. User defined packages : In addition to this, following steps must be taken into consideration.

* Use package statement at the beginning of the package file.
* Define the class that is to be put in the package and declare it as public.
* Create a sub directory under the working directory with the same name as the package name.
* Store the file with same name as the className.java in the subdirectory created.
* Store the compiled version (i.e. class) file into the same sub-directory.

1. Package design guidelines :

* Only closely related classes should belong to the same package.
* Classes that change together should belong to the same package.
* Classes that are not reused together should not belong to the same package.

1. Package declaration is file based :

* All classes in the same source file belong to the same package.
* Each source file may contain an optional package declaration in the following form.

“package <packageName>;”

* Let us consider the source file “ElevatorFrame.java”, for example.

Package elevator;

Public class ElevatorFrame{  
 public double x;

// ………..

}

* The package declaration at the top of the source file declares that the “ElevatorFrame” class belongs to the package named “elevator.”
* When the package declaration is absent from a file, all the classes contained in the file belong to unnamed package.
* A class in a named package can be referred in two ways.

1. More on user defined packages :

* Create a directory named P.
* Store the class A in the file A.java in it.
* Compile A.java and place it in the directory P.
* Store the class B in the field B.java in it.
* Compile B.java and place it in the directory P.
* Import P.\*;
  + Will import all classes in the packages

Note :- We cannot put two or more public classes together in a .java file, otherwise there will be an ambiguity in naming the .java file

1. Finding packages and CLASSPATH : Example

* Consider the following package specification:
  + Package MyPack;
* For a program to find MyPack, one of three things must be true.
  + The program can be executed from a directory immediately above MyPack.
  + The CLASSPATH must be set to include the path to MyPack
  + The –classpath options must specify the path to MyPack when the program is run via Java.

1. Importing a package :

* This is the general form of the import statement.
  + “import pkg1 [.pkg2].(<classname> | \*);”
* Here, pkg1 is the name of a top-level package.

Pkg2 is the name of a sub-ordinary package inside the outer separated by a dot(.).

* There is no practical limit on the depth of a package hierarchy, expect that imposed by the file system.

Finally, you specify either an explicit class or a star (\*), which indicates that Java compiler import the entire package.

Example : “import java.util.Date;”

“import java.io.\*;”

1. Using package :

* Class in a named package can be referred to in two different ways. Import a class or all the classes in the designed package using “import packageName.<className>;”

“import packageName.\*;”

Example :- The “ElevatorPanel” class in package elevator can simply be referred to as elevator when either of the following import classes occurs at the top of source file

“import elevator.ElevatorPanel;”

“import elevator.\*;”

1. Access protection :

* Because of the interplay between classes and packages, Java addresses four categories of visibility for class members.
  + Sub classes in the same package.
  + Non-sub classes in the same package.
  + Sub classes in different package.
  + Classes that are neither in the same package nor sub classes.

The three access modifiers, “private”, “public”, “protected”, provide a variety of ways to produce the many levels of access required by these categories.

Note : Default is not used because it has scope only in the same program file/package in which it is defined.

1. Few important facts about “Abstract” class :

* Any class with “an abstract method is automatically abstract” itself, and must be declared such.
* A class may be declared abstract “even if it has no abstract” method. This prevents it from being instantiated.
* A sub class of an abstract class is instantiated if it overridden each of the abstract method of its super class and provide an implementation (i.e. a method body of all of them).
* If sub class of an abstract class “does not implement all the abstract”, it inherits that “sub class is itself abstract”.

1. Multiple inheritance in Java :

* However, this is not possible as Java does not support multiple inheritances.
* Java’s solution to this problem is called interface.

1. Multiple inheritance and interface :

* Java does not support multiple inheritance.
* Java supports an alternative approach to this OPP feature known as interface.
* What is an interface ?
  + An interface is basically “a kind of class.” Like classes, an interface contains “members” and “methods”, unlike classes, in interface, “all members are final and all methods are abstract”

1. Interface Concepts :

* An interface defines a protocol of behaviour that can be “implement by any class” anywhere in the class hierarchy.
* An interface “defines” a set of methods but “does not implement” them. A class that implements the interface agree to implement all the methods defined in the interface, thereby agreeing to certain behaviour.
* An interface is a named collection of methods definitions (without implementations). Interface reverse behaviour for classes that implements them.
* Methods declared in an interface “are always public and abstract”, Java compiler will not complain if you omit both keywords. “Static methods cannot be declared” in the interface – these methods are never abstract and do not express behaviours of objects.

1. Basic concept of inheritance :
   * Using the keyword interface, one can define an abstract class.
   * Interface are syntactically “similar to classes”, but they lack “instance variables”, and “their methods are defined without any body.”

Example :-

Interface callMe{

Void call(int p);

}

1. Properties of interface :

* Interface must be declared with the keyword “interface.”.
* All interface methods are “implicitly public and abstract.” In other words, you do not need to actually type the public or abstract modifiers in the method declaration, but method is still always public and abstract.
* All variables, defined in an interface is “public”, “static” and “final”. In other words, interface can declare only constants, “no instance variable.”
* Interface methods must not be static.
* Because interface methods are abstract, they cannot be marked final.
* An interface can extends one or more other interfaces.
* An interface cannot implement another interface or class.
* Interface types can be used polymorphically.

1. Extending interface :
   * Interface can inherit other interface.
   * Interface can also multiply inherit.

This was a detailed theoretical explanation of the OPPs concept “Encapsulation”, in which we had seen different sub topics which are equally important and have a great impact on Java programming. I recommend you that if you do not understand any part of this article then just go to Google or YouTube and search about it, make sure that at the end of the day you learn something.

If you have any doubt, question or query related to the above article or in general as well, then feel free to contact me.