Unit 4:

Inheritance and Packaging

Inheritance :

- Inheritance is one of the Basic Concepts of OOPs in which one object acquires the properties and behaviors of the parent object. It's creating a parent-child relationship between two classes.
- It helps to reuse the code and establish a relationship between different classes.

As we can see in the image :
A child inherits the properties from his father.

Similarly, in Java, there are two classes:

1. Parent class (Super or Base class)
2. Child class (Subclass or Derived class)

A class which inherits methods and values from the superclass known as Child class.

whereas

A class from which the child class inherits all the methods and values known as Parent Class.

extends Keyword :

- The extends keyword in Java indicates that the child class inherits or acquires all the properties of the parent class.
- This keyword basically establishes a relationship of an inheritance among classes.
- If a class extends another class, then we say that it has acquired all the properties and behavior of the parent class.
- We use the extends keyword in Java between two class names that we want to connect in the Inheritance relationship.

Note :

- ***It is not possible to extend multiple classes in Java because there is no support for multiple inheritances in Java. And therefore we cannot write multiple class names after the extended keyword.***

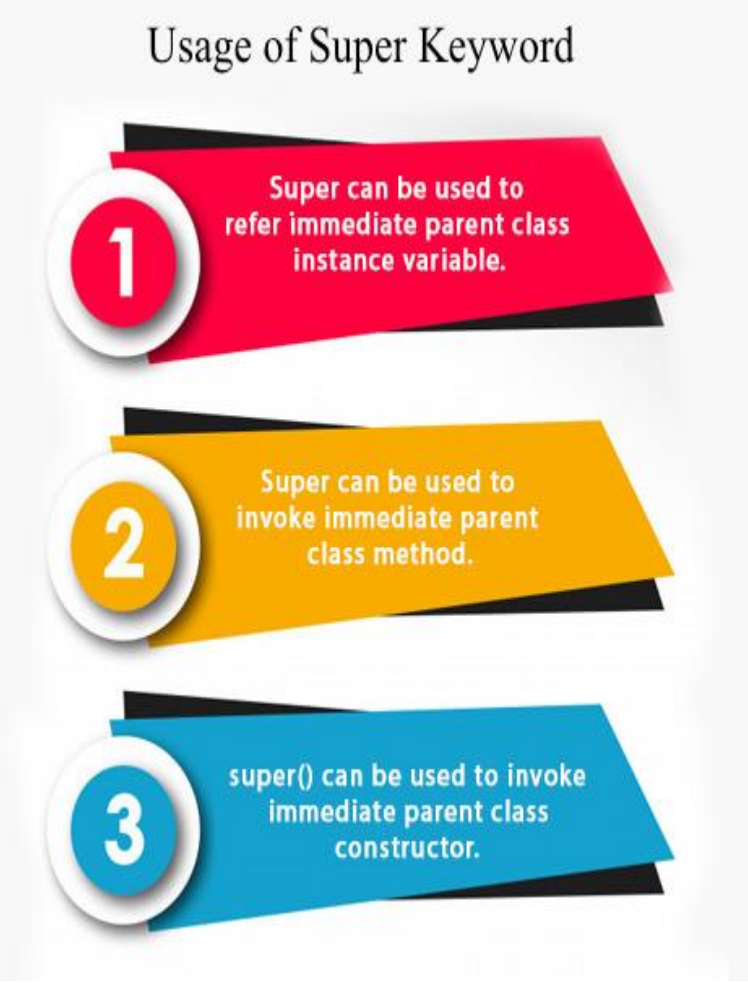
Syntax:

class ParentClass{ ...}

class ChildClass **extends** ParentClass { ... }

Super Keyword:

- The **super** keyword in java is a reference variable that is used to refer parent class objects.
- It is majorly used in the following contexts:



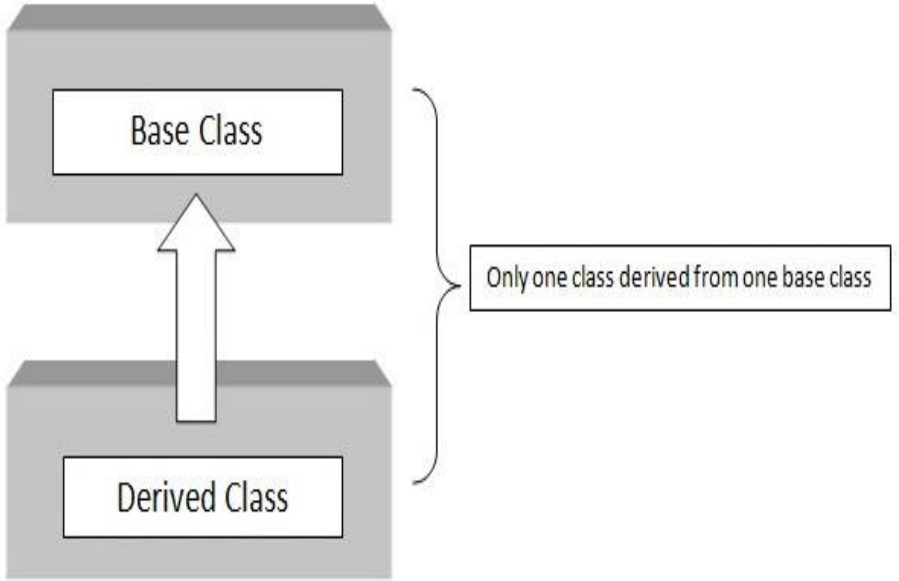
super	super()
The super keyword in Java is a reference variable that is used to refer parent class objects.	The super() in Java is a reference variable that is used to refer parent class constructors.
super can be used to call parent class' variables and methods.	super() can be used to call parent class' constructors only.
The variables and methods to be called through super keyword can be done at any time,	Call to super() must be first statement in Derived(Student) Class constructor.
If one does not explicitly invoke a superclass variables or methods, by using super keyword, then nothing happens	If a constructor does not explicitly invoke a superclass constructor by using super(), the Java compiler automatically inserts a call to the no-argument constructor of the superclass.

Different types of Inheritance in Java :

1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance
4. Multiple Inheritance(Only through Interfaces)
5. Hybrid Inheritance

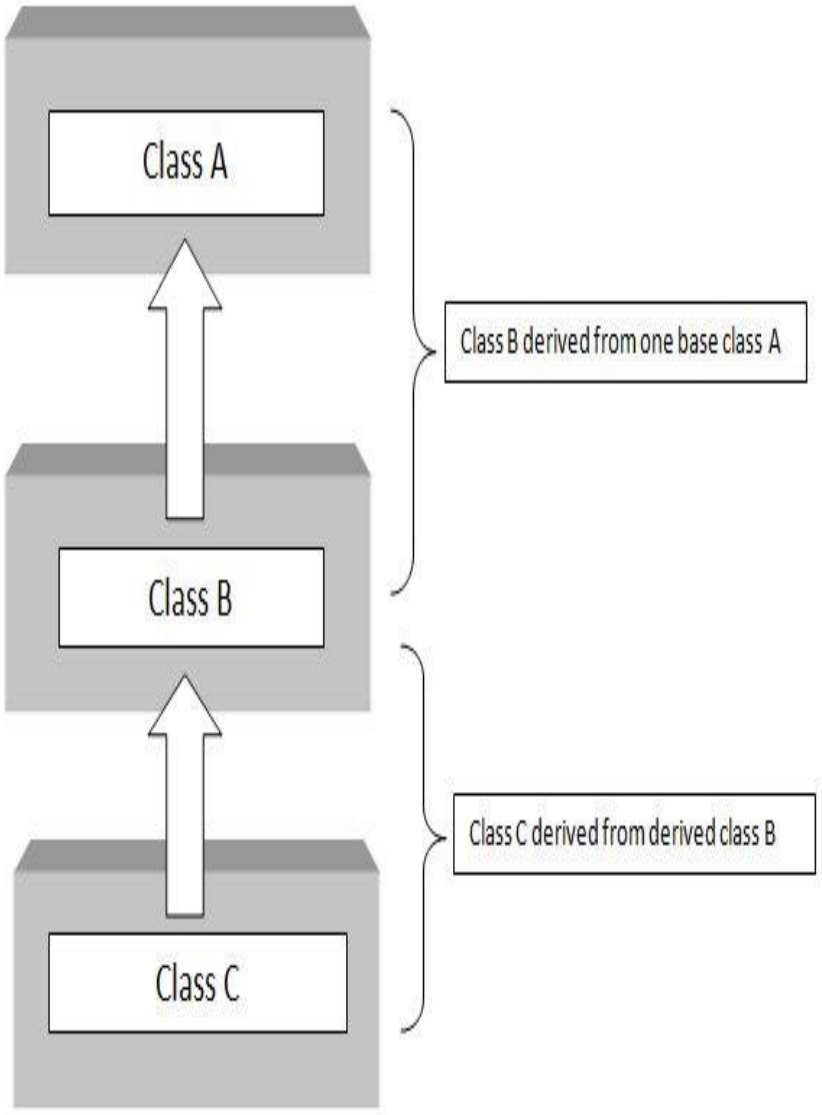
Single Inheritance :

- In single inheritance, a single child class inherits data and methods from its parent class.
- In this case, a child class can access all the methods and the variables of the parent class.



Multilevel Inheritance :

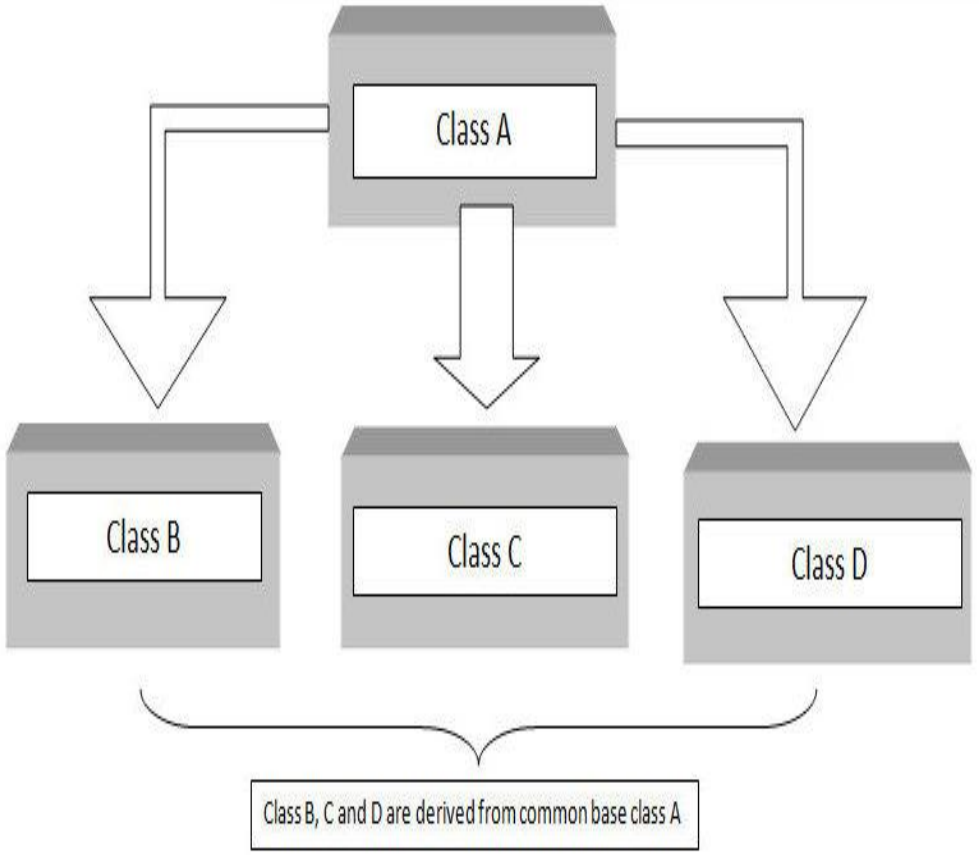
- The multi-level inheritance includes the involvement of at least two or more than two classes.
- One class inherits the features from a parent class and the newly created sub-class becomes the base class for another new class.



Hierarchical Inheritance:

- If more than one class is inherited from the base class, it's known as hierarchical inheritance.

- In hierarchical inheritance, all features that are common in child classes are included in the base class.



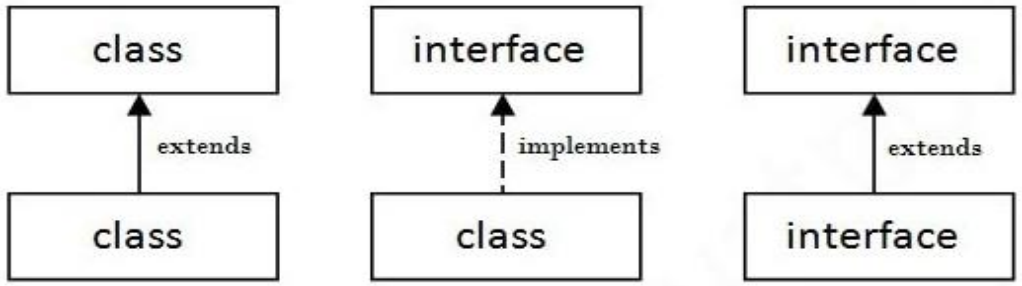
Java Interface :

- Interface can be defined as a container that stores the signatures of the methods to be implemented in the code segment.

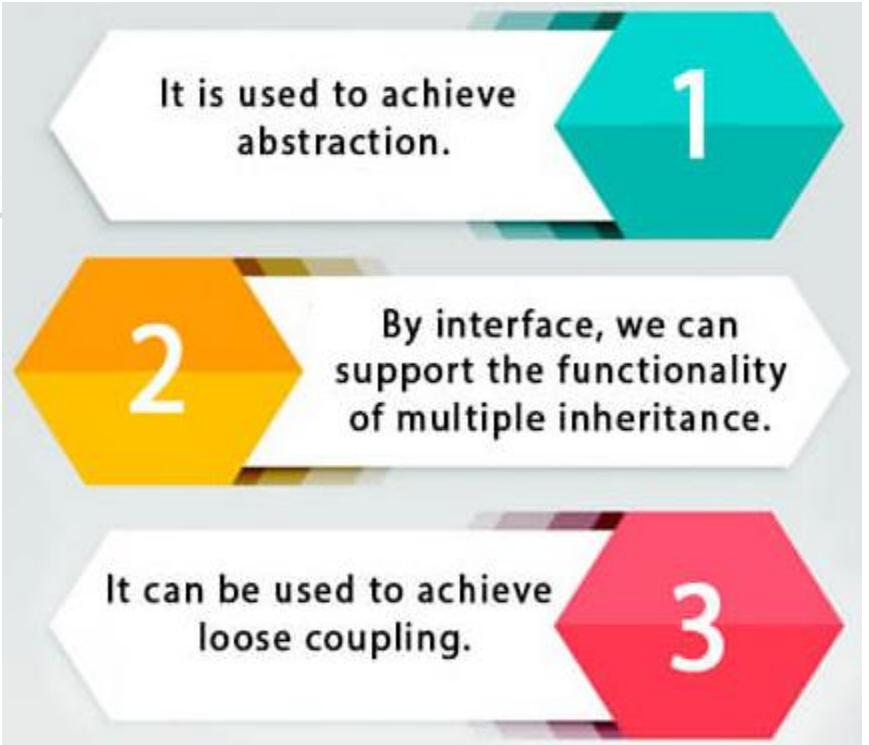
or,

- An **Interface in Java** programming language is defined as an abstract type used to specify the behavior of a class.
- A Java interface contains static constants and abstract methods.
- All methods in the interface are implicitly public and abstract.
- In Java, interfaces are declared using the interface keyword.
- Java Interface also **represents the IS-A relationship**.

- As shown in the figure given below :
  - a class extends another class
  - an interface extends another interface,
  - a **class implements an interface**.



Reasons to use interface



Note : ***The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members.***

Class	Interface
The keyword used to create a class is “class”	The keyword used to create an interface is “interface”
A class can be instantiated i.e, objects of a class can be created.	An Interface cannot be instantiated i.e, objects cannot be created.
Classes does not support multiple inheritance.	Interface supports multiple inheritance.
It can be inherit another class.	It cannot inherit a class.
It can be inherited by another class using the keyword ‘extends’.	It can be inherited by a class by using the keyword ‘implements’ and it can be inherited by an interface using the keyword ‘extends’.
It can contain constructors.	It cannot contain constructors.
It cannot contain abstract methods.	It contains abstract methods only.
Variables and methods in a class can be declared using any access specifier(public, private, default, protected)	All variables and methods in a interface are declared as public.
Variables in a class can be static, final or neither.	All variables are static and final.

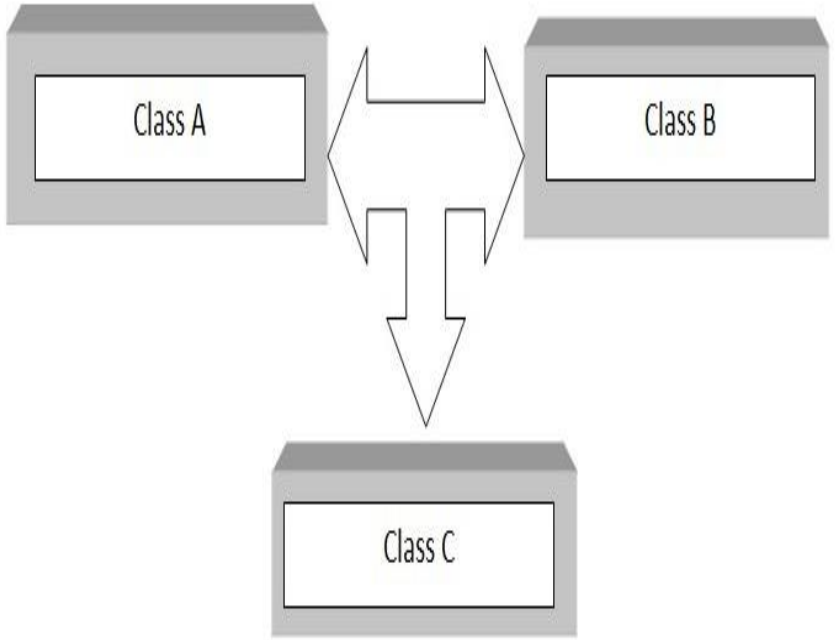
Multiple Inheritance :

- When one class extends more than one classes then this is called **multiple inheritance.** For example: class C extends class A and B then this type of inheritance is known as **multiple inheritance.**

- Multiple inheritance is not supported by Java because of ambiguity problem. This means that a class cannot extend more than one class.

```
public class C extends A, B{}
```

- To achieve multiple inheritance in Java, we must use the interface.



Hybrid Inheritance :

- Hybrid inheritance in Java is a combination of two or more types of inheritances.
- The purpose of using hybrid inheritance in Java is to modularize the codebase into well-defined classes and provide code reusability.

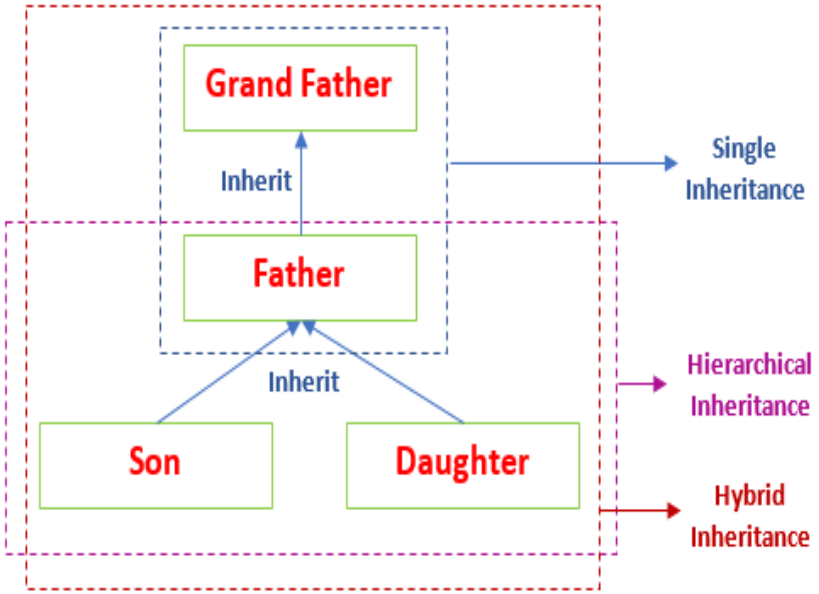


Fig 1.

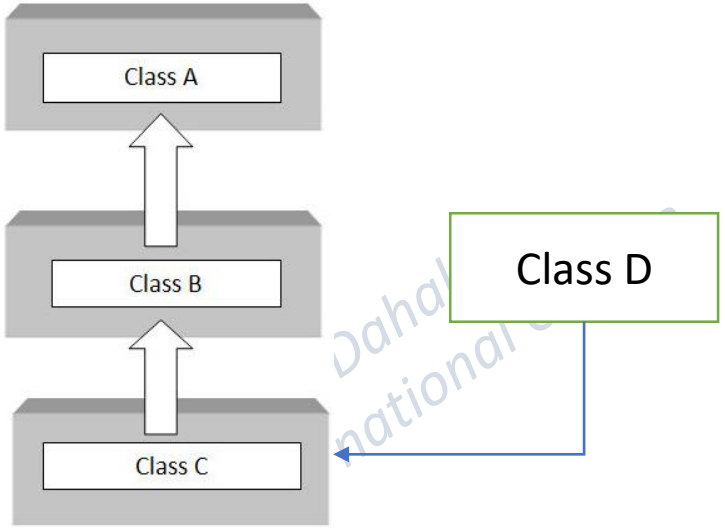


Fig 2.

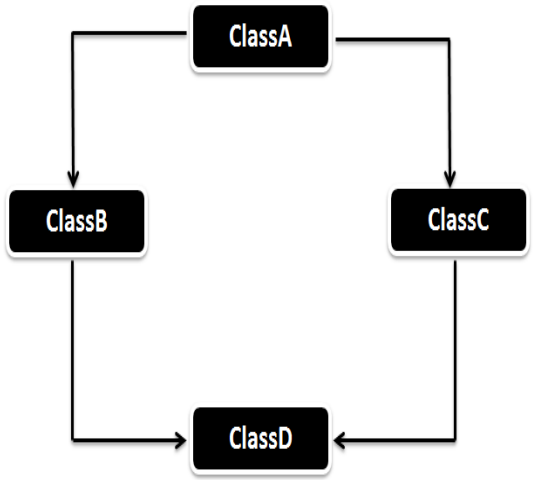


Fig 3.

Final Class in Java :

- In java, the final keyword can be used with variable, method, or class.
- A class that is declared with the final keyword is known as the final class.
- A final class can't be inherited by subclasses.

Syntax:

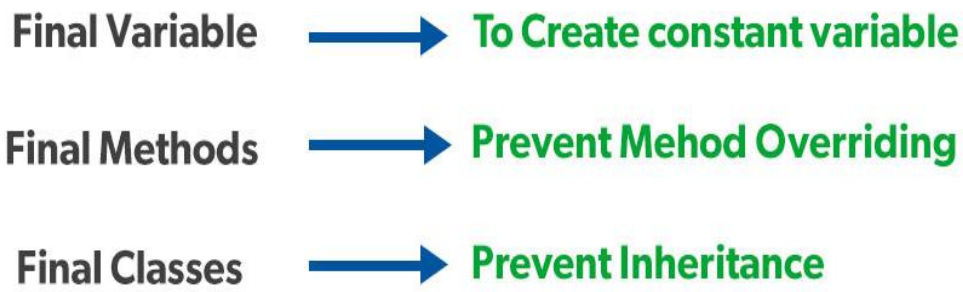
```
accessModifier final class className
{
// Body of class
}
```

When to use a final class in java?

1. A final class is introduced in JDK to prevent the inheritance of class. Suppose I have a class that having some personal or secured information and you want that class should not be extended by any class. Then you should use the final keyword with class.

2. To create an immutable class then the final keyword is mandatory. It means an immutable class is always a final class.

3. The following are different contexts where final is used



Java Packages :

- A package as the name suggests is a pack(group) of classes, interfaces and other packages.
- In java we use packages to organize our classes and interfaces.

Types of packages in Java :

1. Built-in Packages (packages from the Java API) :

- Built-in packages or predefined packages are those that come along as a part of JDK (Java Development Kit) to simplify the task of Java programmer.
- They consist of a huge number of predefined classes and interfaces that are a part of Java API's.
- Some of the commonly used built-in packages are :

- **java.lang,**
- **java.io,**
- **java.util,**
- **java.applet, etc.**

2. User-defined Packages (create your own packages) :

- User-defined packages are those which are developed by users in order to group related classes, interfaces, and sub-packages.

How to Create a package?

Creating a package is a simple task as follows :

- Choose the name of the package
- Include the package command as the first line of code in your Java Source File.
- The Source file contains the classes, interfaces, etc you want to include in the package
- Compile to create the Java packages

Following statement creates a package named texas***:***

package texas;

Note: *If you omit the package statement, the class names are put into the default package, which has no name. Though the default package is fine for short programs, it is inadequate for real applications.*

import Keyword :

- The import keyword is used to import a package, class or interface.

How to access package from another package?
There are three ways to access the package from outside the package.

1. import package.*;'
- If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.
- The **import** keyword is used to make the classes and interface of another package accessible to the current package.

2. import package.classname;
- If you import package.classname then only declared class of this package will be accessible.

3. fully qualified name :
- If you use fully qualified name, then only declared class of this package will be accessible. Now there is no need to import.
- But you need to use fully qualified name every time when you are accessing the class or interface.
- It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Packages are used for:

- Preventing naming conflicts.
  For example, there can be two classes with name Employee in two packages,
  college.staff.cse.Employee and
  college.staff.ee.Employee

- Making searching/locating and usage of classes, interfaces, enumerations and annotations easier.

- Providing controlled access: protected and default have package level access control. A protected member is accessible by classes in the same package and its subclasses. A default member (without any access specifier) is accessible by classes in the same package only.

- Packages can be considered as data encapsulation (or data-hiding).