**Java Package**

A package is Java's style of bundling classes together. **A package is a collection of related classes and interfaces**.

**Or**

A **package** is a single folder (directory) containing a group of related [classes](https://www.cis.upenn.edu/~matuszek/General/JavaSyntax/classes.html) and [interfaces](https://www.cis.upenn.edu/~matuszek/General/JavaSyntax/interfaces.html).

A package is equivalent to a header file of C-lang. Packages can be compressed into **JAR files** for fast traversal in a network or to download from Internet.

#### Package are categorized into two forms

* **Built-in Package:**  
  Existing Java package for example java.lang, java.util etc.
* **User-defined-package:**  
  Java package created by user to categorized classes and interface

**Package vs Directory**

The package of Java, at execution time, is converted into a **directory (folder)** by the operating system. The **java.util** is converted as **java\util**and **java.awt.event** is treated as **java\awt\event** by the OS. The**asterisk \*** is a wild character of Windows which means all the files (means all the classes).

**Default Package:**

Package statement is optional. If we define a class without package statement, then that class is said to be available in **“default package i.e.; current working directory.”**

## Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained. i.e. Package are used in Java, in-order to avoid name conflicts and to control access of class, interface and enumeration etc.   
  
2) Java package provides access protection.

3) Java package removes naming collision.

4) Using package it becomes easier to locate the related classes.

5) With a **single import** statement, all the classes and interfaces can be obtained into our program.

6) Unlike a header file, Java permits to import even a **single class** also.

7) Avoids **namespace problems**. Two classes of the same name cannot be put in the same package but can be placed in two different packages.

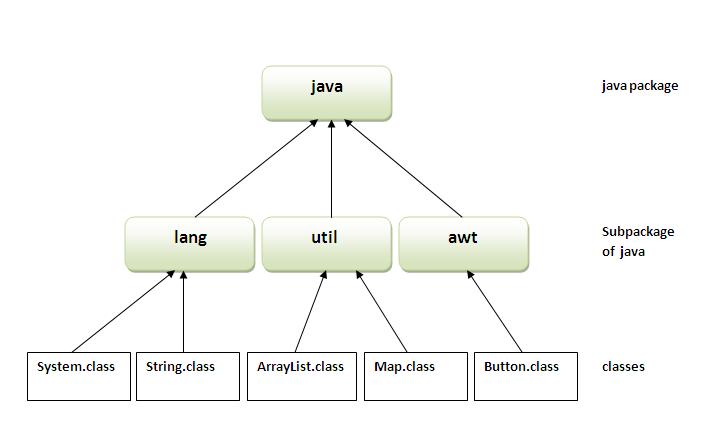
8) Access between the classes can be controlled. Using packages, **restrictions** can be imposed on the access of other package classes. Access Specifiers work on package boundaries (between the classes of other packages).

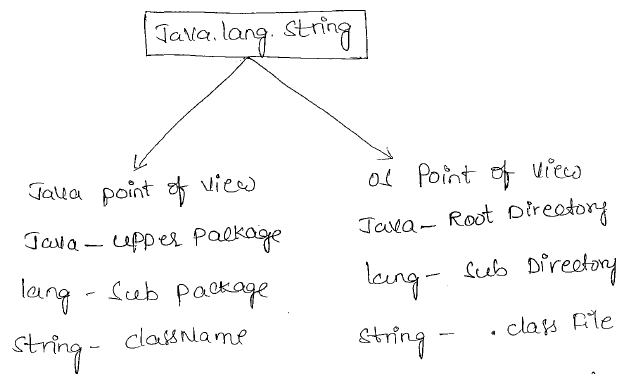
9) We can find all the related classes and interfaces in a single space. Searching and identification will be easier.

10) Packages and sub-packages are the easiest way to **organize** the classes.

11) If we develop Java applications with the concept of packages the we get the following advantages:

1. **Application development time is less.**
2. **Application memory is less.**
3. **Application execution time is less.**
4. **Application performance is enhanced.**
5. **Application redundancy of code is reduced.**
6. **Able to get slogan of Java (WORA).**
7. **We are able to differentiate/provide uniqueness among duplicate classes.**





Predefined packages are the packages which are given by sun micro systems or some other companies as a part of java.

**We have following 3 types of predefined packages.**

**1. Core packages**

Core packages are predefined packages given by sun micro systems which are begin with "java ".

**2. Extended packages**

Extended packages are also predefined packages given by sun micro systems which are begin with "javax ".

**3. Third party packages**

Third party packages are also predefined packages but which are given by some other companies as a part of java software.

**Eg:**

oracle.jdbc, com.mysql, ...

**Package Naming Conventions**

Like identifiers have conventions, the packages come with their own **naming conventions**. Like keywords and protocols, packages are also of lowercase letters. In a single project, a number of programmers may be involved assigned with different modules and tasks. To avoid namespace problems in storing their work, naming conventions are followed. While creating packages, they may follow company name, project name or personal name etc. to precede their package.

**Following are a few examples:**

1. **jyothi.solutions:**

Solutions is the package (folder) preceded by individual name, jyothi.

1. **forecastingtool.jyothi.solutions:**

Preceeded by the project name (forecastingtool) and individual name.

1. **mindspace.forecastingtool.jyothi.solutions:**

Preceeded by company name (mindspace), project name and individual name.

This package naming convention never clashes with the others work in a big project involving many programmers. The same convention is also followed in working on a domain also

**Note**:

Only alphabets, numbers and an underscore are allowed in naming the packages. By convention, names of package should start with lowercase although it is not a condition. Package name should start with a alphabets or underscore but not with a number.

*package javaConcept;****//Valid package name***

*package java\_Concept****;     //Valid package name***

*package java\_12;****//Valid package name***

*package 12\_java;/****/Invalid package name, shouldn’t start with a number.***

*package \_java12****//Valid package Name***

*package JAVA;****//Valid package name but not recommended.***

**Importing All/Single Class**

Packages have an advantage over header files of C-lang. A package allows importing a single class also instead of importing all. C-lang does not have this ease of getting one function from a header file.

*import java.net.\*; //* ***imports all the classes and interfaces****import java.awt.evnet.\*;* ***// imports all the classes and interfacesimport*** *java.net.Socket; //* ***imports only Socket class****import java.awt.event.WindowEvent;* ***// imports only WindowEvent class***

**Note:** While importing a single class, asterisk (\*) should not be used.

## Create User Defined Packages Java

Java is a friendly language and permits to create our own packages and use in programming. We know earlier, Java allows us to create our exceptions. Creating packages are indispensable in project development where number of developers are involved doing different modules and tasks. We know packages avoid **name collision**problems. Package naming conventions are very helpful to locate the applications developed by a single individual or team.

###### **Steps of creating User Defined Packages Java and using them**

1. Create a package with a .class file
2. Set the classpath from the directory from which you would like to access. It may be in a different drive and directory. Let us call it as a target directory.
3. Write a program and use the file from the package.

Let us create a package called **forest** and place a class called **Tiger** in it. Access the package from a different drive and directory.  
 **1st Step: Create a package (forest) and place Tiger.class in it.**

Let us assume **C:\snr** is the current directory where we would like to create the package.

**C:\snr > notepad Tiger.java**

|  |  |
| --- | --- |
|  | *Package forest;*  *import java.util.\*;*  *public class Tiger{*  *public void getDetails(String nickName, int weight)  {*  *System.out.println("Tiger nick name is " + nickName);*  *System.out.println("Tiger weight is " + weight);*  *}*  *}* |

**Order of Package Statement**

The above program coding wise is very simple but is important to know the steps of package creation.

***package forest;  
import java.util.\*;  
public class Tiger***

**Package** is a keyword of Java followed by the package name. Just writing the package statement followed by the name creates a new package; see how simple Java is to practice. For this reason, Java is known as a production language.

While create User Defined Packages Java, the order of statements is very important. The order must be like this, else, compilation error.

1. ***Package statement***
2. ***Import statement***
3. ***Class declaration***

If exists, the package statement must be first one in the program. If exists, the import statement must be the second one. Our class declaration is the third. Any order changes, it is a compilation error.

When the code is ready, the next job is compilation. We must compile with package notation. Package notation uses –d compiler option as follows.

**javac -d . Tiger.java**

The **–d compiler option** creates a new folder called forest and places the Tiger.class in it. The**dot (.)** is an operating system's environment variable that indicates the current directory. It is an instruction to the OS to create a directory called forest and place the Tiger.class in it.

**Using User Defined Packages Java**

After creating the package let us use it.

**2nd step: Set the classpath from the target directory.**

Let us assume D:\sumathi is the target directory. Let us access Tiger.class in forest package from here.

From the target directory set the classpath following way.

**set classpath=C:\snr;%classpath%;**

***classpath is another environment variable which gives the address of the forest directory to the OS. %classpath% informs the OS to append the already existing classpath to the current classpath that is right now set.***

***3rd Step:***

***Now finally, write a program from the target directory D:/sumathi and access the package.***

**notepad Animal.java**

**The above statement creates a file called Animal.java and write the code in it, say, as follows**

|  |
| --- |
| *import forest.Tiger;*  *public class Animal{*  *public static void main(String args[])  {*  *Tiger t1 = new Tiger ();*  *t1.getDetails("Everest", 50);*  *}*  *}* |

The compilation and execution is as usual as follows.

Javac Animal.java  
java Animal

***Importing the Package – Precautions***

The following statement raises compilation error.

*import forest.\*;*

To work with the above statement, remove the source file Tiger.java file from C:\snr directory; else use as import forest.Tiger.

**import forest.Tiger;**

In the above statement, only Tiger class from forest package is imported (ofcourse, in this case you have only class in the package).

**import forest.\*;**

The above statement imports all the classes present in the forest package. But if you write as above with \* mark, the program does not work. To work with, you have got Tiger.java in the C:\snr folder (from where you have compiled earlier). Now delete the Tiger.java file from c:\snr. Then \* will work.

#### Creating a package

Creating a package in java is quite easy. Simply include a package command followed by name of the package as the first statement in java source file.

***package mypack;***

***public class employee {***

***...statement;***

***}***

The above statement creates a package called **mypack**.

Java uses file system directory to store package.   
For example the .class for any classes you to define to be part of **mypack** package must be stored in a directory called mypack

#### Example of package creation

*package mypack*

*class Book{*

*String bookname;*

*String author;*

*Book(String b, String c) {*

*this.bookname = b;*

*this.author = c;*

*}*

*public void show(){*

*System.out.println(bookname+" "+ author);*

*}*

*}*

*class test{*

*public static void main(String[] args){*

*Book bk = new Book("java","Herbert");*

*bk.show();*

*}*

*}*

##### **To run this program :**

* Create a directory under your current working development directory(i.e. JDK directory), name it as **mypack**.
* compile the source file
* Put the class file into the directory you have created.
* Execute the program from development directory.

**NOTE :** Development directory is the directory where your JDK is install.

#### Uses of java package

Package is a way to organize files in java, it is used when a project consists of multiple modules. It also helps resolve naming conflicts. Package's access level also allows you to protect data from being used by the non-authorized classes.

## Simple example of java package

The **package keyword** is used to create a package in java.

1. **//save as Simple.java**
2. package mypack;
3. public class Simple{
4. public static void main(String args[]){
5. System.out.println("Welcome to package");
6. }
7. }

## How to compile java package

If you are not using any IDE, you need to follow the **syntax** given below:

**javac -d directory javafilename**

**Example:**

**javac -d . Simple.java**

The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

## How to run java package program

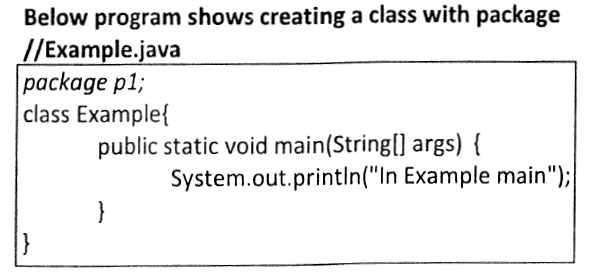
You need to use fully qualified name e.g. mypack.Simple etc to run the class.

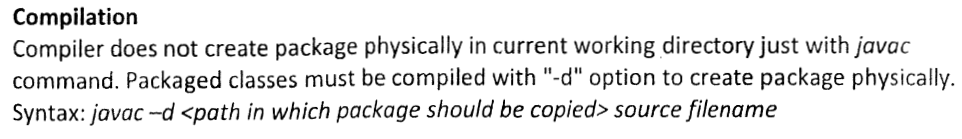
|  |
| --- |
| **To Compile:** javac -d . Simple.java |
| **To Run:** java mypack.Simple |

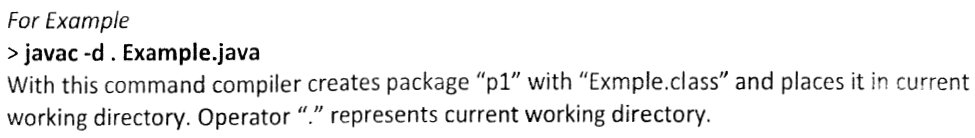
**Output:** Welcome to package

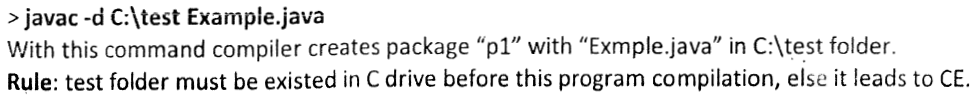
|  |
| --- |
| The -d is a switch that tells the compiler where to put the class file i.e. it represents destination. The . represents the current folder. |

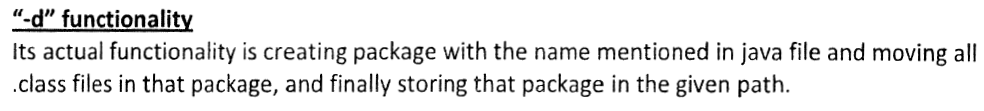
## Example:

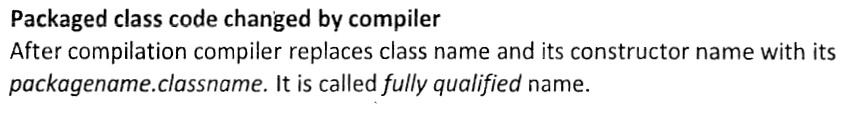


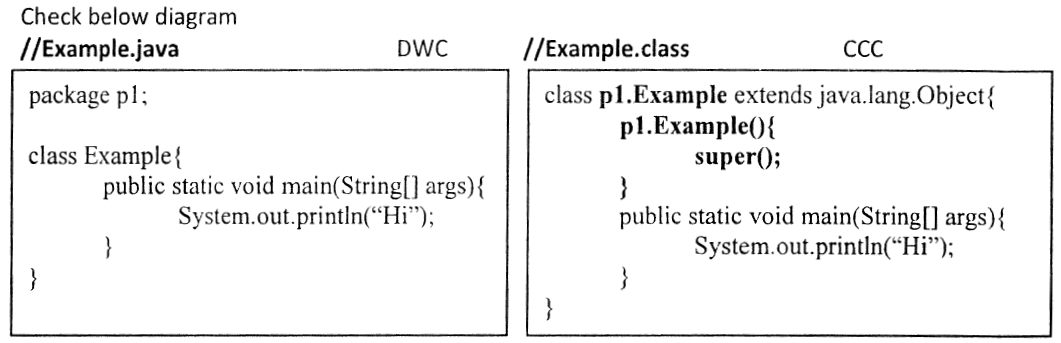


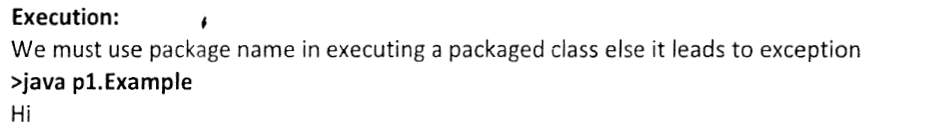


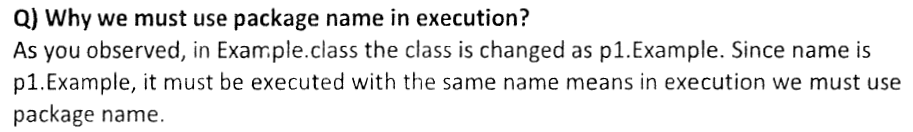


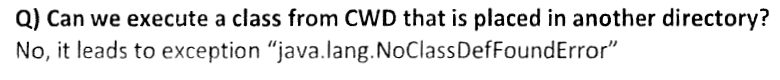


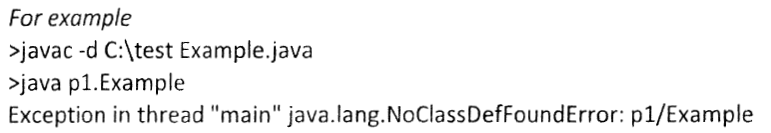


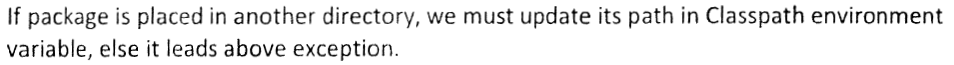












## How to access package from another package?

There are three ways to access the package from outside the package.

1. **import package.\*;**
2. **import package.classname;**
3. **fully qualified name.**

#### 1) Using packagename.\*

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.

## Example of package that import the packagename.\*

*//save by A.java*

*package pack;*

*public class A{*

*public void msg(){System.out.println("Hello");}*

*}*

*//save by B.java*

*package mypack;*

*import pack.\*;*

*class B{*

*public static void main(String args[]){*

*A obj = new A();*

*obj.msg();*

*}*

*}*

**Output:** Hello

#### 2) Using packagename.classname

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

## Example of package by import package.classname

***//save by A.java***

***package****pack;*

***public******class****A{*

***public******void****msg(){*

*System.out.println("Hello");*

*}*

*}*

***//save by B.java***

***package****mypack;*

***import****pack.A;*

***class****B{*

***public******static******void****main(String args[]){*

*A obj =****new****A();*

*obj.msg();*

*}*

*}*

**Output:** Hello

#### 3) Using 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.

## Example of package by import fully qualified name

***//save by A.java***

***package****pack;*

***public******class****A{*

***public******void****msg(){*

*System.out.println("Hello");*

*}*

*}*

***//save by B.java***

***package****mypack;*

***class****B{*

***public******static******void****main(String args[]){*

*pack.A obj =****new****pack.A();//using fully qualified name*

*obj.msg();*

*}*

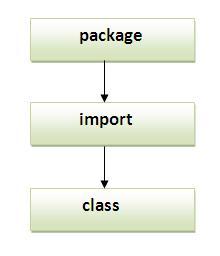
*}*

**Output:** Hello

#### Note: If you import a package, subpackages will not be imported.

If you import a package, all the classes and interface of that package will be imported excluding the classes and interfaces of the subpackages. Hence, you need to import the sub-package as well.

#### Note: Sequence of the program must be package then import then class.



## Subpackage in java

Package inside the package is called the **subpackage**. It should be created **to categorize the package further**.

Let's take an example, Sun Microsystem has definded a package named java that contains many classes like System, String, Reader, Writer, Socket etc. These classes represent a particular group e.g. Reader and Writer classes are for Input/Output operation, Socket and ServerSocket classes are for networking etc and so on. So, Sun has subcategorized the java package into subpackages such as lang, net, io etc. and put the Input/Output related classes in io package, Server and ServerSocket classes in net packages and so on.

#### The standard of defining package is domain.company.package e.g. com.javatpoint.bean or org.sssit.dao.

### 

### Example of Subpackage

*package com.javatpoint.core;*

*class Simple{*

*public static void main(String args[]){*

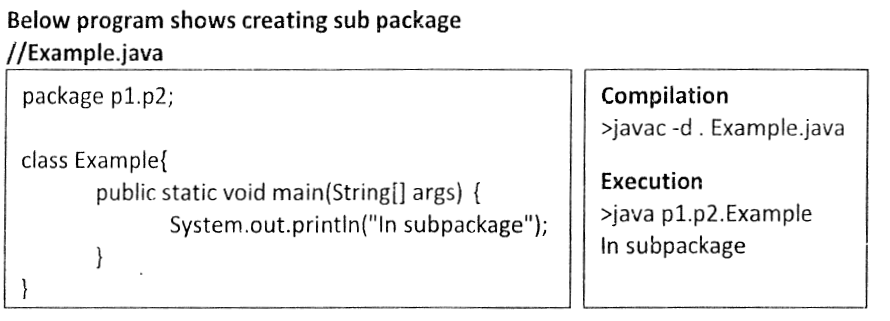
*System.out.println("Hello subpackage");*

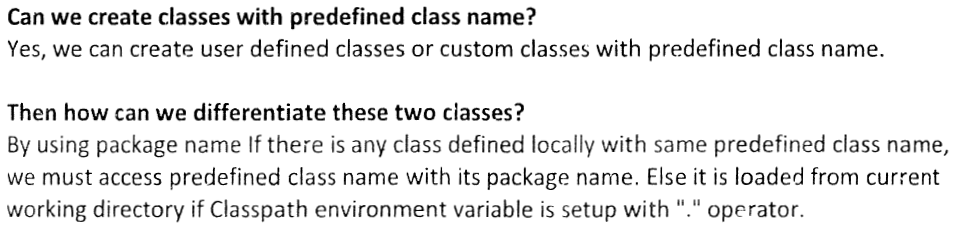
*}*

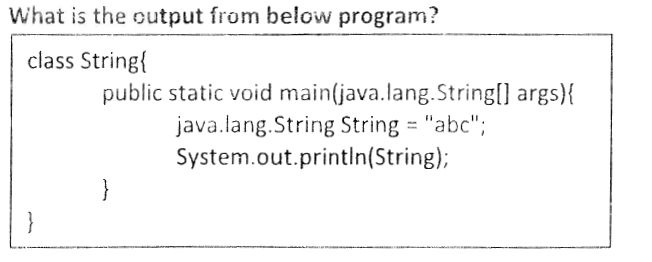
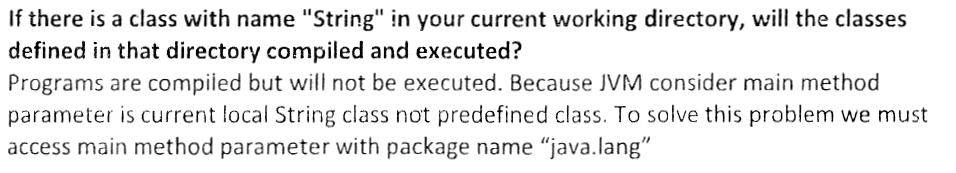
*}*

|  |
| --- |
| **To Compile:** javac -d . Simple.java |
| **To Run:** java com.javatpoint.core.Simple |

**Output:** Hello subpackage



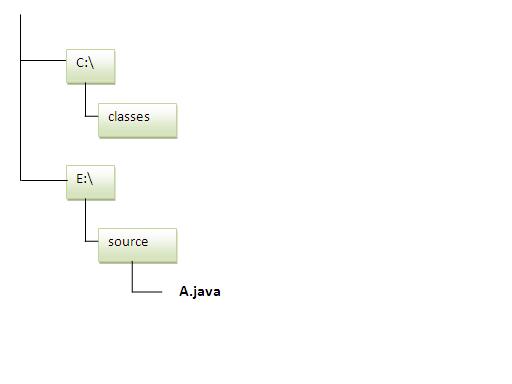




## How to send the class file to another directory or drive?

There is a scenario, I want to put the class file of A.java source file in classes folder of c: drive.

**Example:**



*//save as Simple.java*

***package****mypack;*

***public******class****Simple{*

***public******static******void****main(String args[]){*

*System.out.println("Welcome to package");*

*}*

*}*

### To Compile:

***javac -d c:\classes Simple.java***

### To Run:

|  |
| --- |
| To run this program from e:\source directory, you need to set classpath of the directory where the class file resides. |
| ***set classpath=c:\classes;.;*** |
| ***java mypack.Simple*** |

### Another way to run this program by -classpath switch of java:

The -classpath switch can be used with javac and java tool.

To run this program from e:\source directory, you can use -classpath switch of java that tells where to look for class file. For example:

***java -classpath c:\classes mypack.Simple***

**Output:**Welcome to package

### Ways to load the class files or jar files

|  |
| --- |
| There are two ways to load the class files temporary and permanent. |

* **Temporary**
  + By setting the classpath in the command prompt
  + By -classpath switch
* **Permanent**
  + By setting the classpath in the environment variables
  + By creating the jar file, that contains all the class files, and copying the jar file in the jre/lib/ext folder.

#### Rule: There can be only one public class in a java source file and it must be saved by the public class name.

*//save as C.java otherwise Compilte Time Error*

***class****A{}*

***class****B{}*

***public******class****C{}*

### How to put two public classes in a package?

|  |
| --- |
| If you want to put two public classes in a package, have two java source files containing one public class, but keep the package name same. For example: |

*//save as A.java*

***package****javatpoint;*

***public******class****A{}*

*//save as B.java*

***package****javatpoint;*

***public******class****B{}*

=======================================

# Package class

The package class provides methods to get information about the specification and implementation of a package. It provides methods such as getName(), getImplementationTitle(), getImplementationVendor(), getImplementationVersion() etc.

## Example of Package class

In this example, we are printing the details of java.lang package by invoking the methods of package class.

*class PackageInfo{*

*public static void main(String args[]){*

*Package p=Package.getPackage("java.lang");*

*System.out.println("package name: "+p.getName());*

*System.out.println("Specification Title: "+p.getSpecificationTitle());*

*System.out.println("Specification Vendor: "+p.getSpecificationVendor());*

*System.out.println("Specification Version: "+p.getSpecificationVersion());*

*System.out.println("Implementaion Title: "+p.getImplementationTitle());*

*System.out.println("Implementation Vendor: "+p.getImplementationVendor());*

*System.out.println("Implementation Version: "+p.getImplementationVersion());*

*System.out.println("Is sealed: "+p.isSealed());*

*}*

*}*

**Output:** package name: java.lang

Specification Title: Java Plateform API Specification

Specification Vendor: Sun Microsystems, Inc.

Specification Version: 1.6

Implemenation Title: Java Runtime Environment

Implemenation Vendor: Sun Microsystems, Inc.

Implemenation Version: 1.6.0\_30

IS sealed: false

**Resolving Namespace Problems**

By placing the same class in two different packages, which Java permits, namespace problems can be solved. **Namespace is the area of execution of a program in RAM**. The **Date** class exists in two packages – **java.util** and **java.sql**. Importing these two packages in a program gives**ambiguity**problem to the compiler. In the following program compiler gets ambiguity problem and is solved with fully-qualified name.

|  |  |
| --- | --- |
|  | ***import java.util.\*;***  ***import java.sql.\*;***  ***public class NSProblem{***  ***public static void main(String args[])     {***  ***//   Date d1 = new Date();      // raises compilation error***  ***java.util.Date d1 = new java.util.Date();***  ***// no error***  ***System.out.println("Today is: " + d1);***  ***}***  ***}*** |

###### **Java Class Libraries – : Predefined Packages Java API**

All the classes and interfaces that come with the installation of JDK are put together are known as **Java API (Application Programming Interface**). All the Java API packages are prefixed with **jav**a or **javax**. Following table gives some important packages, a few prominent classes and their functionality.

###### Following table gives frequently used Predefined Packages Java supports

Random, Date, GregorianCalendar and the DS like Stack,  
Vector, LinkedList, HashMap etc.

|  |  |  |
| --- | --- | --- |
| PACKAGE NAME | EXAMPLE CLASSES | FUNCTIONALITY (PURPOSE) |
| **java.lang** | System, String, Object, Thread, Exception etc. | These classes are indispensable for every Java program. For this reason, even if this package is not imported, JVM automatically imports. |
| **java.util** | These are called as utility (service) classes and are used very frequently in coding. |  |
| **java.io** | FileInputStream, FileOutputStream, FileReader, FileWriter, RandomAccessFile, BufferedReader, BufferedWriter etc. | These classes are used in all I/O operations including keyboard input. |
| **java.net** | URL, ServerSocket, Socket, DatagramPacket, DatagramSocket etc. | Useful for writing socket programming (LAN communication). |
| **java.applet** | AppletContext, Applet, AudioStub, AudioClip etc | Required for developing applets that participate on client-side in Internet (Web) programming. |
| **java.awt** | Button, Choice, TextField, Frame, List, Checkbox etc. | Essential for developing GUI applications. |
| **java.awt.event** | MouseListener, ActionListener, ActionEvent, WindowAdapter etc. | Without these classes, it is impossible to handle events generated by GUI components |
| **java.sql** | DriverManager, Statement, Connection, ResultSet etc | Required for database access in JDBC applications. |

**Table: Predefined Packages Java Java API**

**What is the difference between Java API package and User-defined packages?**

Java API comes with JDK installation. User defined, you are creating. Basically, both are packages and classes and methods.

### CLASSPATH - For Locating Classes

CLASSPATH is an environment variable (i.e., global variables of the operating system available to all the processes) needed for the Java compiler and runtime to locate the Java packages used in a Java program. (Why not call PACKAGEPATH?) This is similar to another environment variable PATH, which is used by the CMD shell to find the executable programs.

**CLASSPATH can be set in one of the following ways:**

1. CLASSPATH can be set permanently in the environment: In Windows, choose control panel ⇒ System ⇒ Advanced ⇒ Environment Variables ⇒ choose "System Variables" (for all the users) or "User Variables" (only the currently login user) ⇒ choose "Edit" (if CLASSPATH already exists) or "New" ⇒ Enter "CLASSPATH" as the variable name ⇒ Enter the required directories and JAR files (separated by semicolons) as the value (e.g., ".;c:\javaproject\classes;d:\tomcat\lib\servlet-api.jar"). Take note that you need to include the current working directory (denoted by '.') in theCLASSPATH.

To check the current setting of the CLASSPATH, issue the following command:

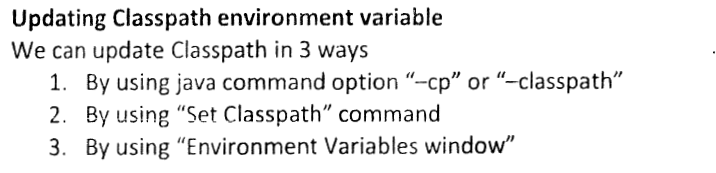
> SET CLASSPATH

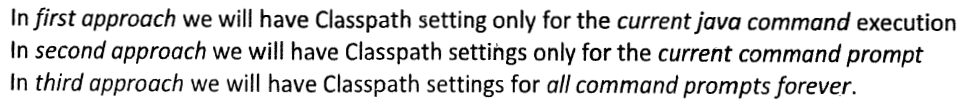
1. CLASSPATH can be set temporarily for that particular CMD shell session by issuing the following command:

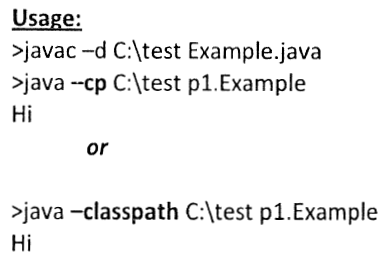
> SET CLASSPATH=.;c:\javaproject\classes;d:\tomcat\lib\servlet-api.jar

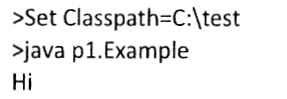
1. Instead of using the CLASSPATH environment variable, you can also use the command-line option -classpath or -cp of the javac and java commands, for example,

> java –classpath c:\javaproject\classes com.abc.project1.subproject2.MyClass3









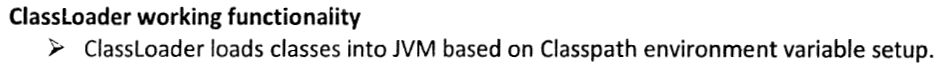
### How Classes are Found?

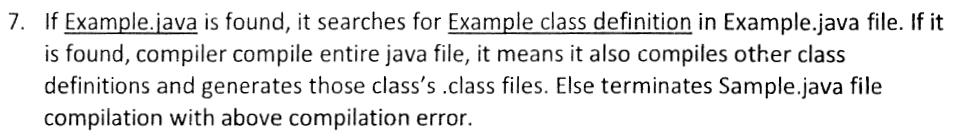
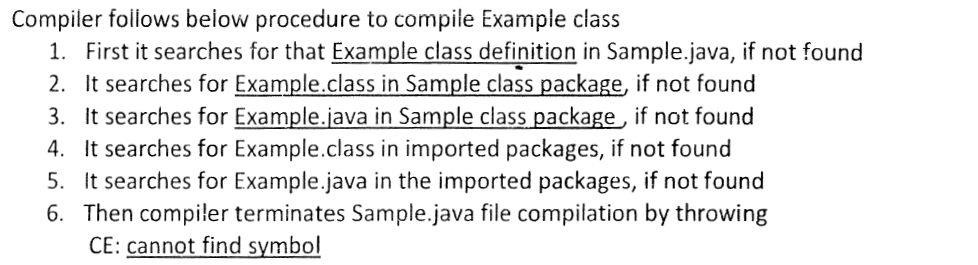
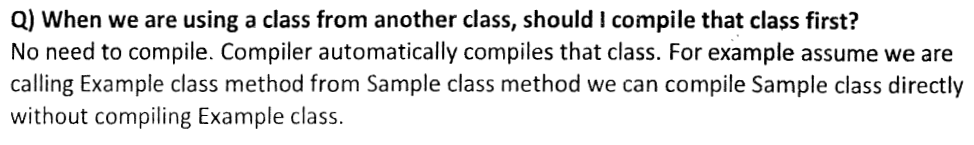
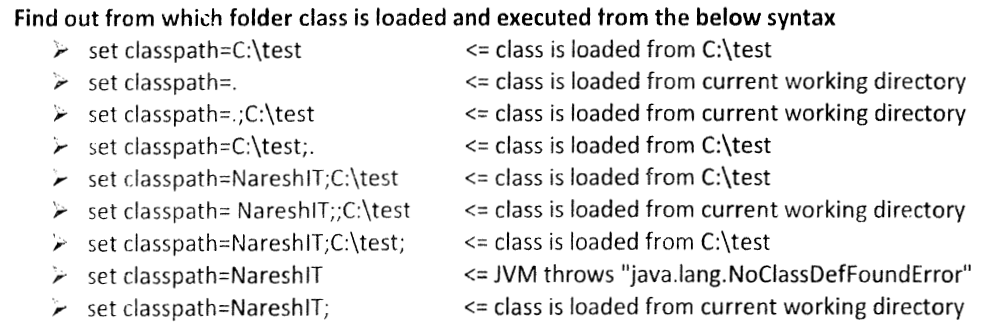
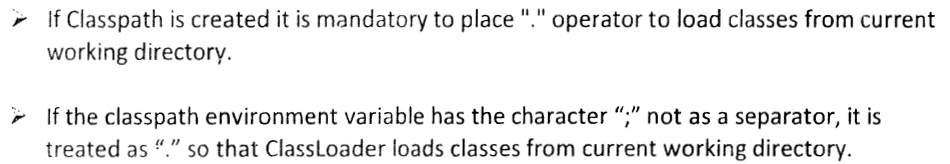
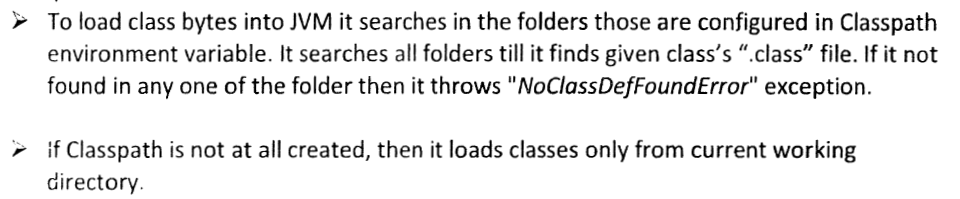
**The Java Virtual Machine (JVM) searches for and loads classes in this order:**

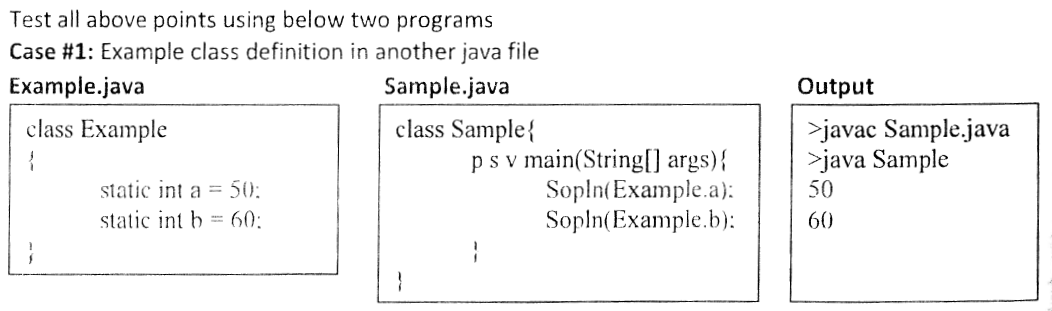
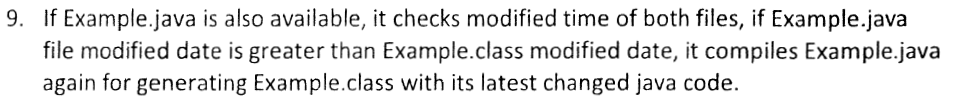
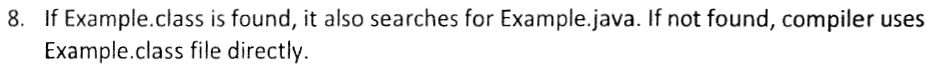
1. **Bootstrap Classes:** include rt.jar (runtime), and other classes specified in the sun.boot.class.path system property, which could include i18n.jar (internationalization), sunrsasign.jar,jsse.jar, jce.jar, charsets.jar, and jre/classes.
2. **Extension Classes via Java Extension mechanism:** classes bundled as JAR file and kept in the "$JAVA\_HOME/jre/lib/ext" directory.
3. **User Classes:** located via –classpath or –cp command-line option or CLASSPATH environment variable.

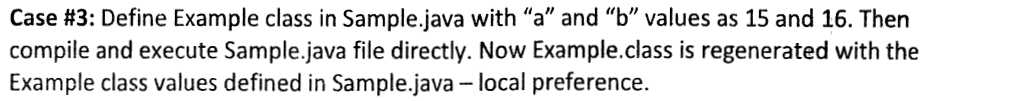
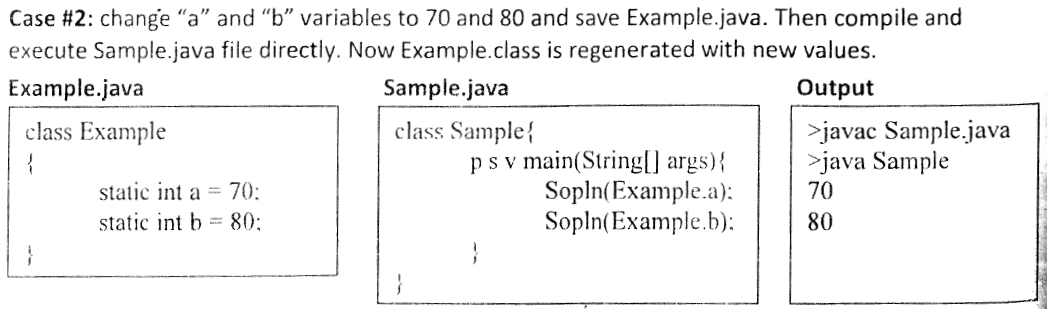
Normal users need not concern about bootstrap and extension classes. User classes are found though the so-called user class path - a list of directories and JAR files which contain class files. The directories and JAR files in the user class path are separated with a semi-colon ';' for Windows systems, or colon ':' for UNIX systems. The user class path is kept in the System Property java.class.path. The value is obtained from:

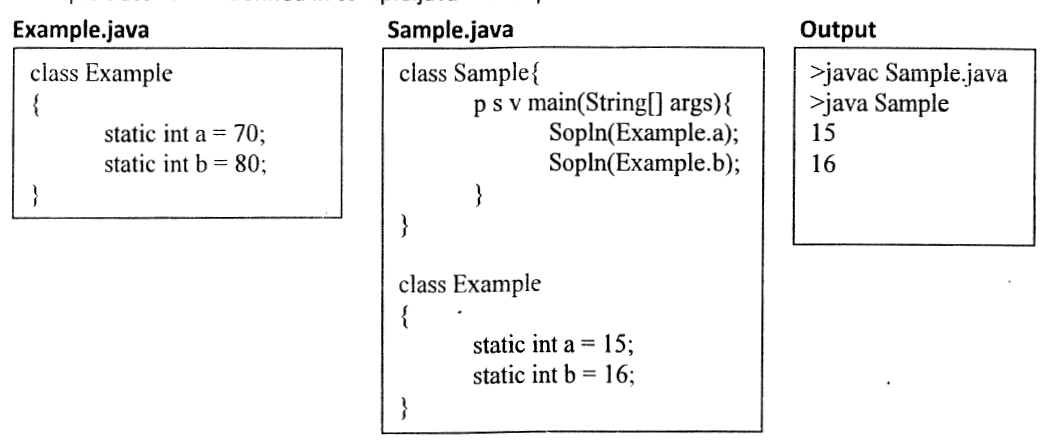
1. The default value '.' or current working directory.
2. The value of the CLASSPATH environment variable, which overrides the default value.
3. The value of –classpath or -cp command-line option, which overrides both the default value and the CLASSPATH value.
4. The JAR files in the -jar command line option, which overrides all other values.

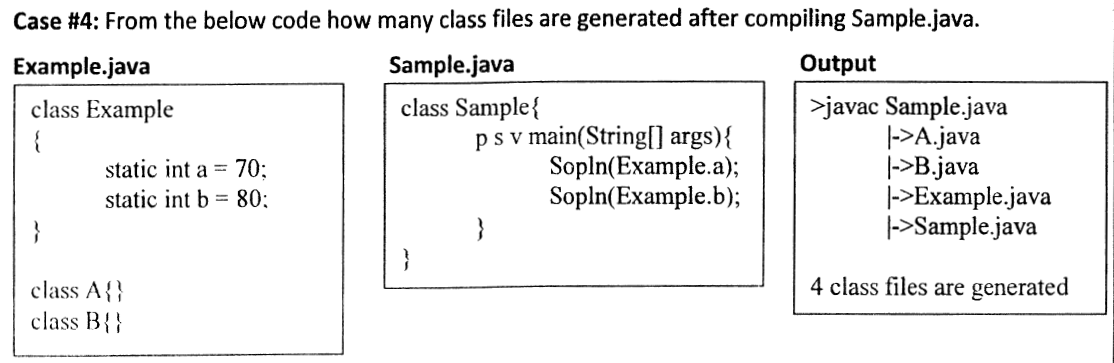
******

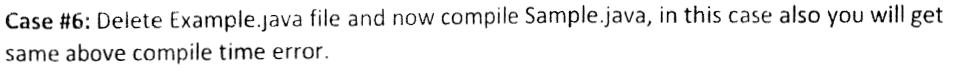
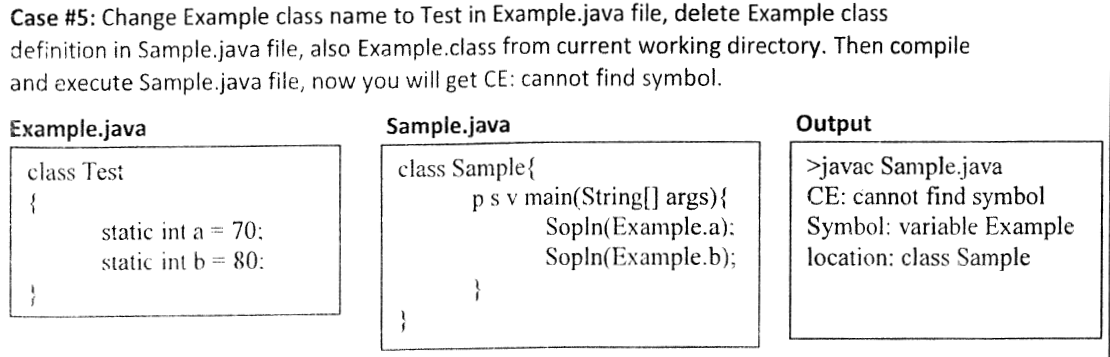
******

******

******

******

******

******

***SCJP:***

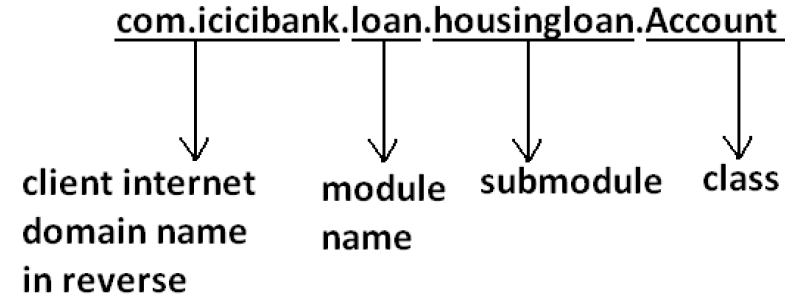
***Package statement:***

It is an encapsulation mechanism to group related classes and interfaces into a single module.

**The main objectives of packages are:**

* To resolve name confects.
* To improve modularity of the application.
* To provide security.
* There is one universally accepted naming conversion for packages that is to use internet domain name in reverse.

**Example:**

**

**How to compile package Program:**

**Example:**

**package com.durgajobs.itjobs;**

**class HydJobs{**

**public static void main(String args[]){**

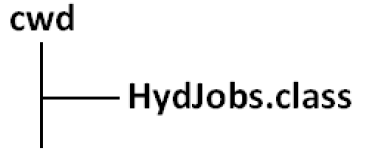
**System.out.println("package demo");**

**}**

**}**

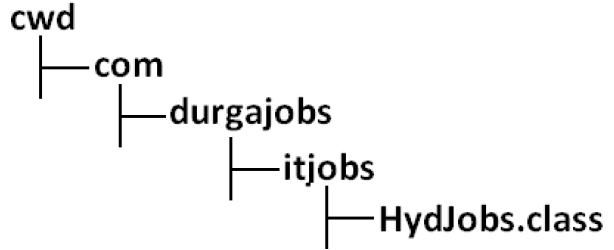
**Javac HydJobs.java generated class file will be placed in current working directory.**

**Diagram:**

**

* Javac -d . HydJobs.java
* -d means destination to place generated class files "." means current working
* directory.
* Generated class file will be placed into corresponding package structure.

**Diagram:**

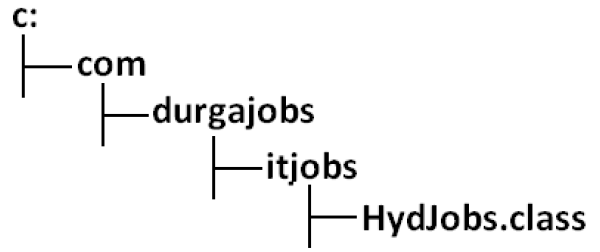
**

* If the specified package structure is not already available then this command itself will create the required package structure.
* As the destination we can use any valid directory.
* If the specified destination is not available then we will get compile time error.

**Example:**

**D:\Java>javac -d c: HydJobs.java**

**Diagram:**

**

If the specified destination is not available then we will get compile time error.

**Example:**

**javac -d z: HydJobs.java**

If Z: is not available then we will get compile time error.

**How to execute package Program:**

**java com.durgajobs.itjobs.HydJobs**

**At the time of execution compulsory we should provide fully qualified name.**

**Conclusion 1:**

**In any java Program there should be at most one package statement that is if we are taking more than one package statement we will get compile time error.**

**Example:**

*package pack1;*

*package pack2;*

*class A*

*{*

*}*

**Output:**

*Compile time error.*

*D:\Java>javac A.java*

*A.java:2: class, interface, or enum expected*

*package pack2;*

**Conclusion 2:**

In any java Program the 1st non comment statement should be package statement [if it is available] otherwise we will get compile time error.

**Example:**

*import java.util.\*;*

*package pack1;*

*class A*

*{*

*}*

**Output:**

*Compile time error.*

*D:\Java>javac A.java*

*A.java:2: class, interface, or enum expected*

*package pack1;*

**List of predefined J2SE package:**

J2SE contains 8 essential packages. They are :

* 1. Java.lang.\*:

It is one of the predefined package and it is always used for obtaining language facilities/ services.

**Some of the facilities are:**

i. Accepting command line arguments.

ii. Data Conversions.

iii. Obtaining garbage collection facilities.

iv. Developing the thread based applications.

v. Developing the exception based applications etc.,

By default this package is imported to each and every Java program and hence this package is called “Default Package.**”   
Note:**

In java, for every java program following 2 packages are available by default.

**1. java.lang**

**2. current working directory**

* 1. **Java.awt.\*; [Abstract Window Tool Kit]**

The purpose of this package is to design GUI applications. To design the GUI applications, we need GUI components which are available in the form of pre-defined classes.

**Example:** Label, Button, Checkbox, Text field etc.

Creating a GUI component is nothing but creating an object of an appropriate pre-defined class.

**Example:** Create a button with a name “Save”.

**Button b1=new Button(“Save”);**

* 1. **Java.awt.event.\*;**

Here event is the sub-package of awt package. Event package contains collection of classes, interfaces. They provide functionality/behavior to the GUI applications.

To develop the complete GUI application we must import java.awt.\* and java.awt.event.\*.

* 1. **Java.io.\*; [File Programming]**

The purpose of this package is to achieve the data persistency by using the concept of files

**Definition of Stream:**

The flow of data between main memory and secondary memory is called Stream.

* 1. **Java.applet.\*; [Applet Programming]**

The purpose of this package is to develop distributed applications where they can run in the context of browser and whose results are sharable across the universe.

To develop the distributed applications, in the initial versions of Java SUN has developed a concept called **applets**.

To fulfill the concept of applets, we have predefined class called **Applet** present in Java.applet.\*.

* 1. **Java.net.\*;[Network Programming]**

The purpose of this application is to develop networking applications or client server applications for sharing the data between multiple machines which are located in same network or in different network.

For developing client and server side applications there exists separate classes and interfaces which are present in java.net.\* package.

* 1. **Java.util.\*;[ Collection Framework]**

The purpose of this package is to develop high performance java based applications.

**Definition of Collection framework:**

It is one of the standardized mechanisms which allow grouping multiple values either of same type or different type or both the types in a single variable with a dynamic size. This single variable is called Collection framework variable.

* 1. **Java.text.\*; [Text Processing]**

This package is used for 3 purposes. They are  
i. Formatting the dates.  
ii. Formatting the times.  
iii. Numerical manipulations like truncating, rounding, ceiling, floaring etc.   
In other words, this package we use most of the times in report generation modules.

## *Why both java.awt and java.awt.event ?*

**Why java.awt.event is to be imported separately when already java.awt package is imported?**

It is a very general doubt uncleared in many minds. To get the clear answer, first you must know how the packages are maintained by Java with the colloboration of OS. **A package of Java is treated as a directory by the OS**. It is proved with **-d** option of **javac** utility. On the name of the package, a directory is created by the OS, which you can see practically.

Now come to our point.

* **java.awt is treated as /java/awt>**
* **java.awt.event is treated as /java/awt/event>**

###### **Why both java.awt java.awt.event ?**

When we look into "**/java/awt>**" with **dir** command, the OS shows the files and subdirectories of **awt**. It shows clearly that **event**subdirectory exists; but it is not opened. To see the files of event directory, we must go to "**/java/awt/event>**" and give the **dir** command.

That is, with **java.awt**, the event subpackage is imported but it classes are not opened and thereby not available to the program. To use the classes of **event** sub package, it is necessary to import explicitly **java.awt.event** package; else compiler raises error saying "cannot resolve symbol".

*What are the differences between packages and interfaces ?*

*Answer :* Interface concept always makes us to understand how to reuse the features within the program between classes to class, interfaces to interface and interfaces to class but not across the programs.

Packages concept allows to reuse the features both within and across the programs between class to class, interfaces to interface and interfaces to interface.

**Representing a package as a jar file:**

* Go to console, type
  + C:\> jar –cvf  
    **Options:**
  + **Example:**

***jar –cvf xy.jar myPack***

* Now let’s suppose copy xy.jar file and paste it in the same path as rt.jar i.e, C:\Program-files\Java\jdk\_1.6.0\jre\lib.
* **Unjaring the Jar file:**Just right click on jar file, open with winzip.