CHARUSAT
CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

CE251: PROGRAMMING IN JAVA
July – November 2020-21

Java Packages: Putting Classes Together

Prof. Rima Patel
Assistant Professor

Devang Patel Institute of Advance Technology and Research

Introduction

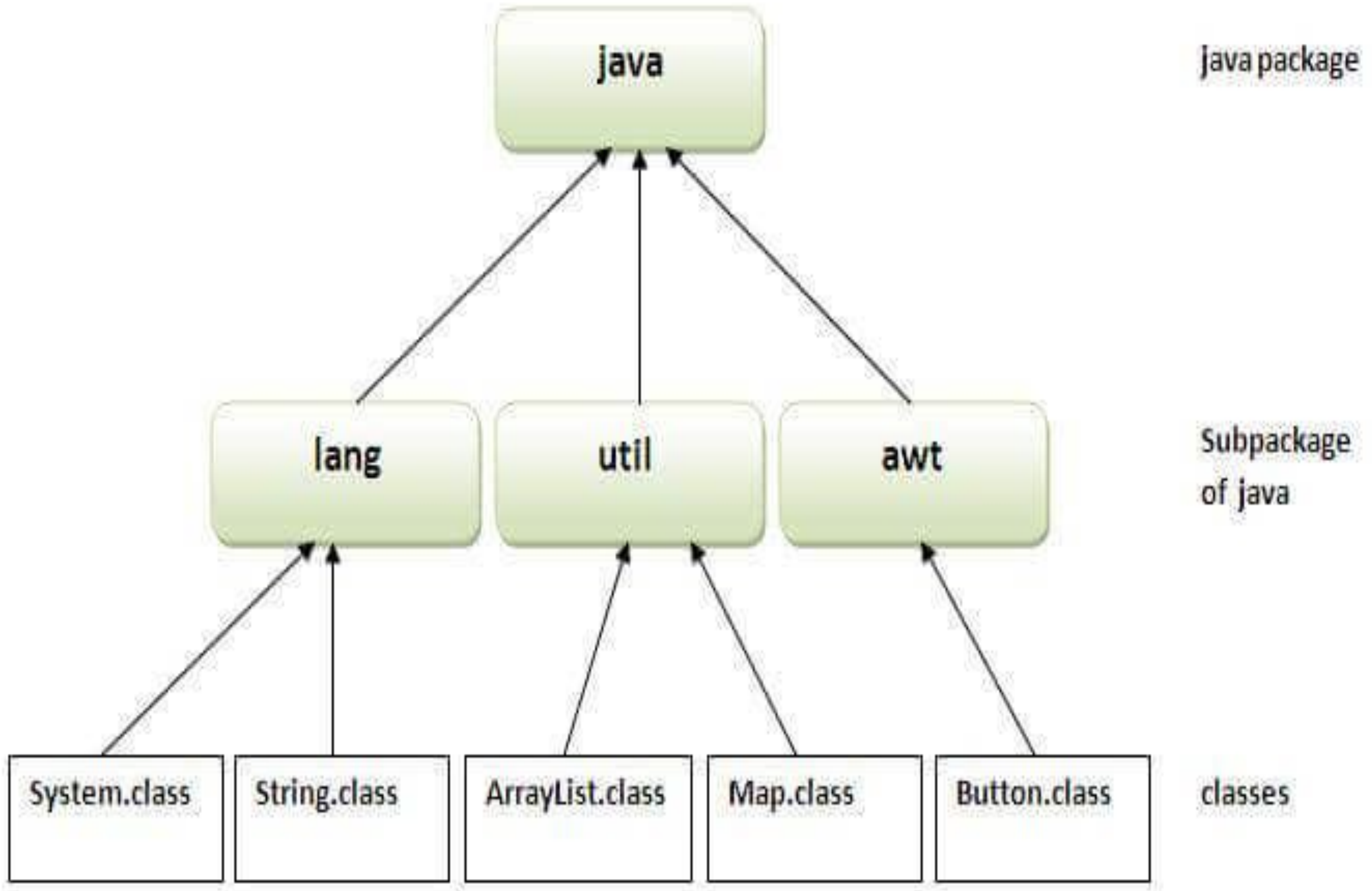
- The main feature of OOP is its ability to support the reuse of code:
  - Extending the classes (via inheritance)
  - Extending interfaces
- The features in basic form limited to reusing the classes within a program.

What if we need to use classes from other programs without physically copying them into the program under development ?

In Java, this is achieved by using what is known as “packages”, a concept similar to “class libraries” in other languages.

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.


java
java package
lang
util
awt
Subpackage of java
System.class
String.class
ArrayList.class
Map.class
Button.class
classes

- Advantage of Java Package
- 1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- 2) Java package provides access protection.
- 3) Java package removes naming collision.

What packages are contains?

Packages

Packages are Java’s way of grouping a number of related classes and/or interfaces together into a single unit. That means, packages act as “containers” for classes.

What are the benefits of organising classes into packages ?

- The classes contained in the packages of other programs/applications can be reused.
- In packages classes can be unique compared with classes in other packages. That two classes in two different packages can have the same name. If there is a naming clash, then classes can be accessed with their fully qualified name.
- Classes in packages can be hidden if we don't want other packages to access them.
- Packages also provide a way for separating "design" from coding.

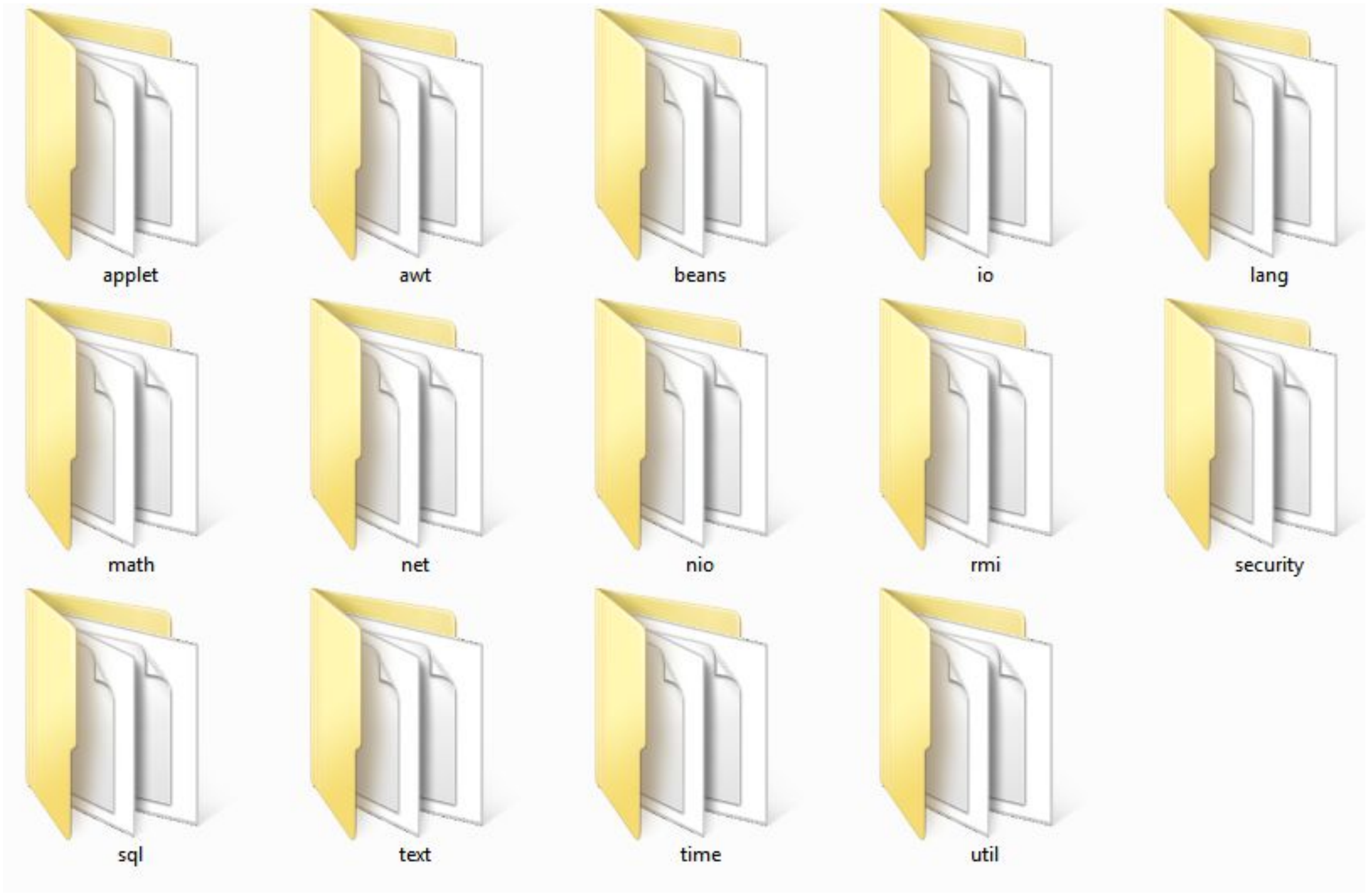
What is the real time use of packages?

- Every project divided into different set of modules
- Every modules is nothing but a package statement

Java Predefine/foundation Packages

- Java provides a large number of classes grouped into different packages based on their functionality.
- The 14 predefined Java packages are:
  - java.lang
    - Contains classes for primitive types, strings, math functions, threads, and exception
  - java.util
    - Contains classes such as vectors, hash tables, date etc.
  - java.io
    - Stream classes for I/O
  - java.awt
    - Classes for implementing GUI – windows, buttons, menus etc.
  - java.net
    - Classes for networking
  - java.applet
    - Classes for creating and implementing applets

Where you can find all those packages?


applet
awt
beans
io
lang
math
net
nio
rmi
security
sql
text
time
util

Example

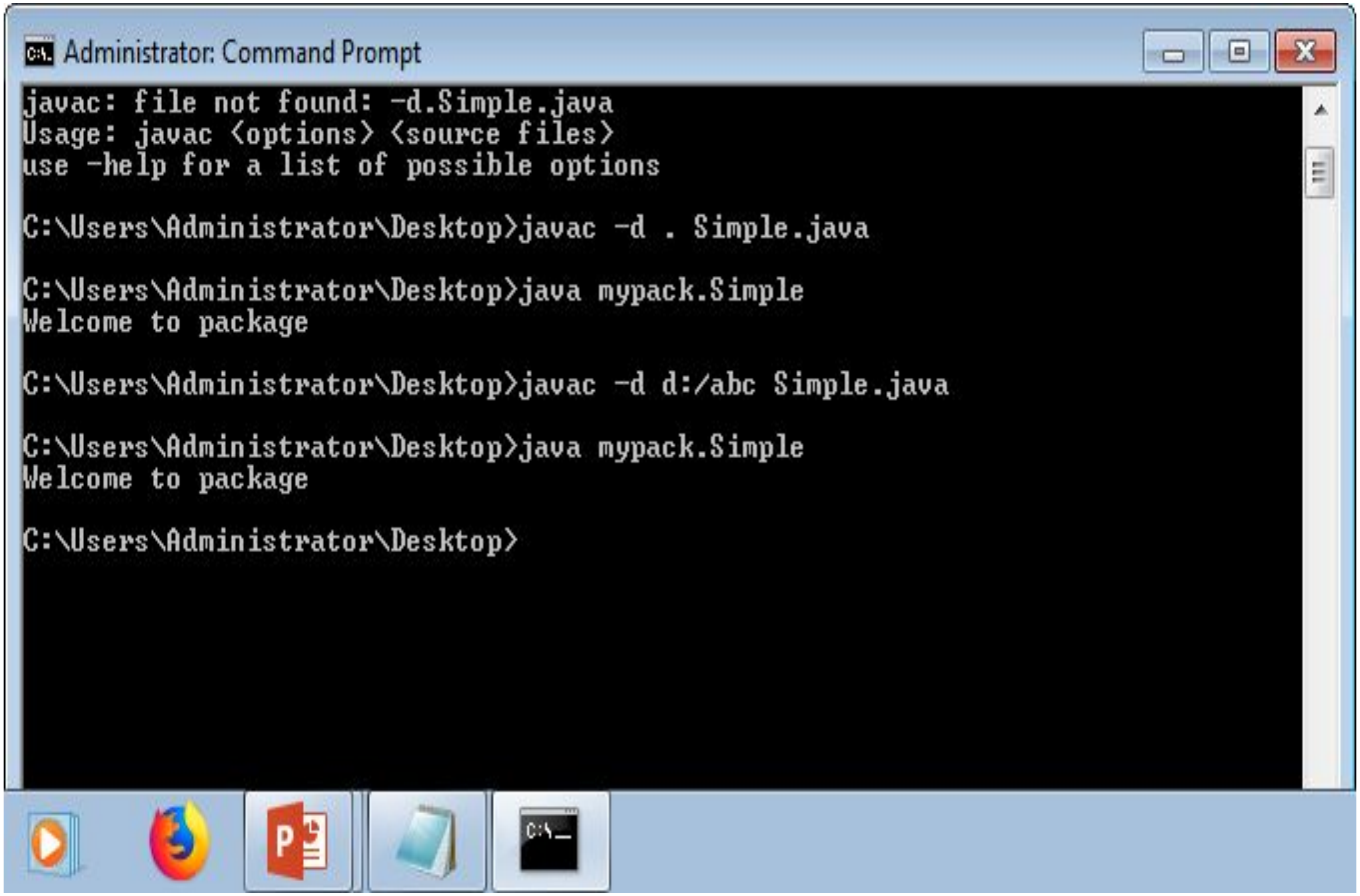
```
//save as Simple.java
package mypack;
public class Simple{
 public static void main(String args[]){
    System.out.println("Welcome to package
");
   }
}
```

How to run java package program

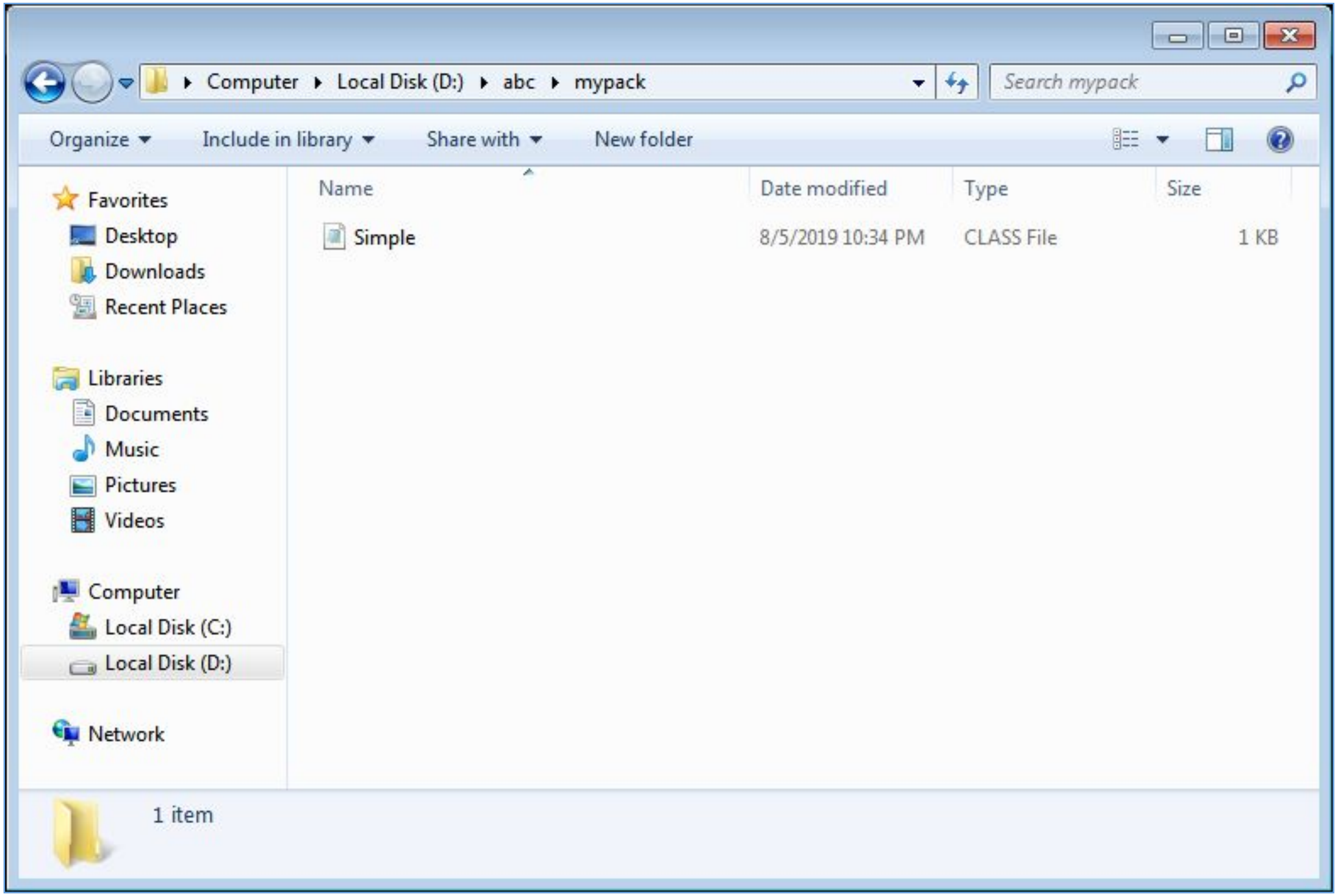
You need to use fully qualified name e.g. mypack.Simple etc to run the class.

To Compile: javac -d . Simple.java

To Run: java mypack.Simple



```
javac: file not found: -d.Simple.java
Usage: javac <options> <source files>
use -help for a list of possible options

C:\Users\Administrator\Desktop>javac -d . Simple.java

C:\Users\Administrator\Desktop>java mypack.Simple
Welcome to package

C:\Users\Administrator\Desktop>javac -d d:/abc Simple.java

C:\Users\Administrator\Desktop>java mypack.Simple
Welcome to package

C:\Users\Administrator\Desktop>
```

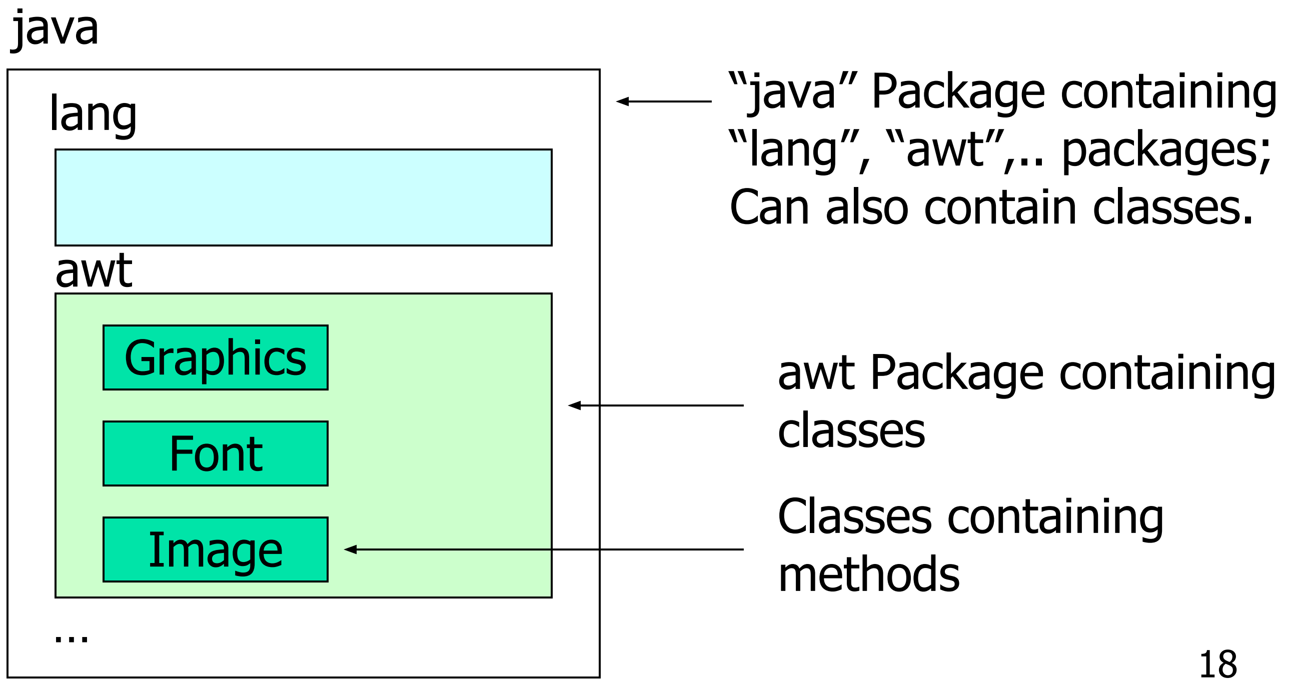


Name	Date modified	Type	Size
Simple	8/5/2019 10:34 PM	CLASS File	1 KB

1 item

Using System Packages

- The packages are organised in a hierarchical structure. For example, a package named "java" contains the package "awt", which in turn contains various classes required for implementing GUI (graphical user interface).


java
lang
awt
Graphics
Font
Image
...
"java" Package containing "lang", "awt",.. packages; Can also contain classes.
awt Package containing classes
Classes containing methods

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.

Accessing Classes from Packages

- There are two ways of accessing the classes stored in packages:
  - Using fully qualified class name
    - java.lang.Math.sqrt(x);
  - Import package and use class name directly.
    - import java.lang.Math
    - Math.sqrt(x);
- Selected or all classes in packages can be imported:

```
import package.class;
import package.*;
```

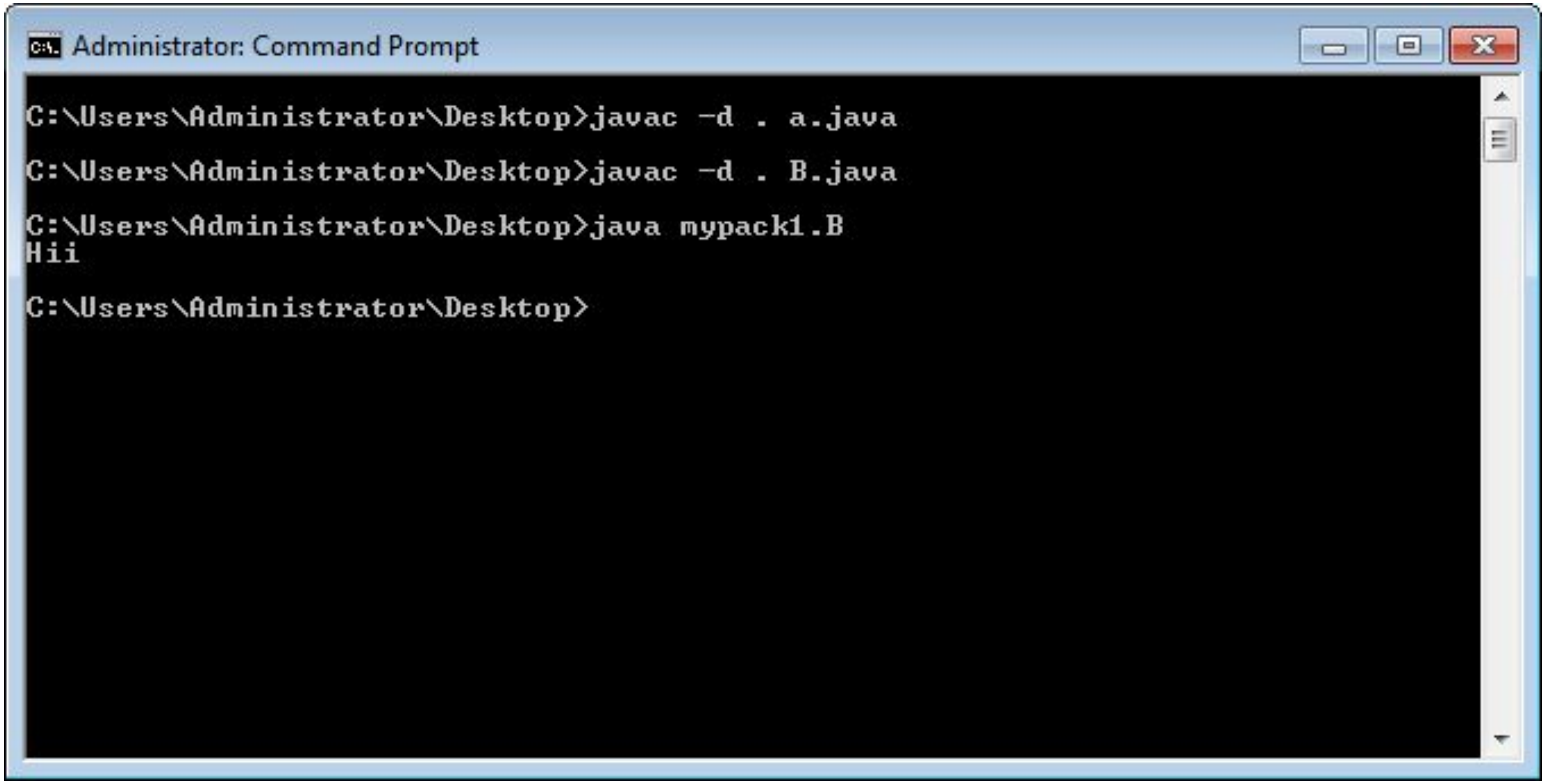
- Implicit in all programs: import java.lang.*;
- package statement(s) must appear first

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

```
//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();
  }
}
```



```
C:\Users\Administrator\Desktop>javac -d . a.java

C:\Users\Administrator\Desktop>javac -d . B.java

C:\Users\Administrator\Desktop>java mypack1.B
Hii

C:\Users\Administrator\Desktop>
```

Using packagename.classname

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

Example

```
//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();
  }
}
```

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

```
//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[]){
   pack.A obj = new pack.A();
obj.msg();
  }
}
```

User defined Packages

What is the naming convention to declare package?

- **Package names** are written in all lower case to avoid conflict with the **names** of classes or interfaces. Companies use their reversed Internet domain **name** to begin their **package names**—for example, com.example.mypackage for a **package** named mypackage created by a programmer

Example

```
package in.ac.charusat.ce.java;

class MyPackage
{    public static void main(String[] args)
     {
          System.out.println("Hello World!");
     }
}
class B
{
}
class C
{
}
```

Creating Packages

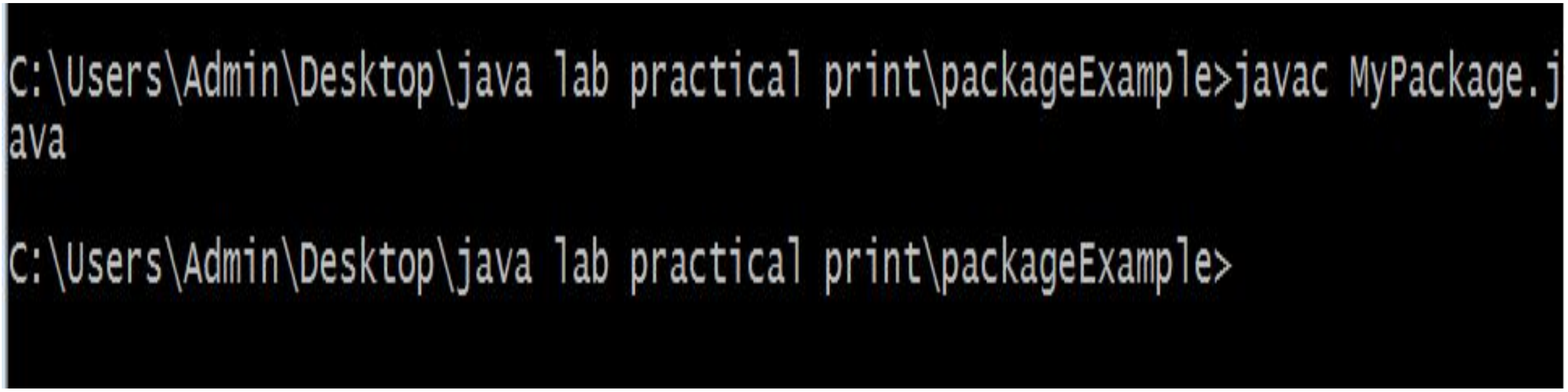
- Java supports a keyword called "package" for creating user-defined packages.
- The package statement must be the first statement in a Java source file (except comments and white spaces) followed by one or more classes.

```
package myPackage;
public class ClassA {
        // class body
}
class ClassB {
 // class body
}
```

- Package name is "myPackage" and classes are considered as part of this package; The code is saved in a file called "ClassA.java" and located in a directory called "myPackage".

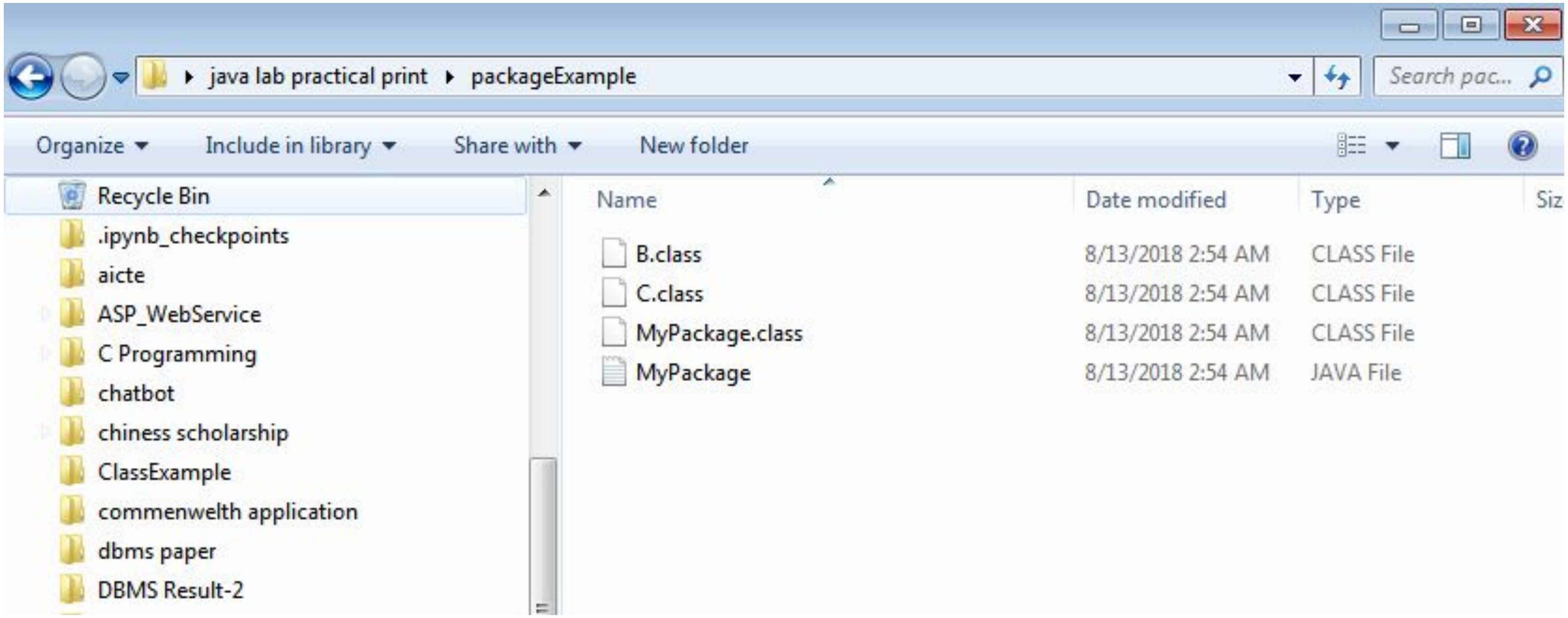
How to compile it?

javac MyPackage.java



```
C:\Users\Admin\Desktop\java lab practical print\packageExample>javac MyPackage.j
ava

C:\Users\Admin\Desktop\java lab practical print\packageExample>
```

How many classes will be generated?



java lab practical print ▸ packageExample

Organize ▾ Include in library ▾ Share with ▾ New folder

Recycle Bin
.ipynb_checkpoints
aicte
ASP_WebService
C Programming
chatbot
chiness scholarship
ClassExample
commenwelth application
dbms paper
DBMS Result-2

Name	Date modified	Type	Siz
B.class	8/13/2018 2:54 AM	CLASS File	
C.class	8/13/2018 2:54 AM	CLASS File	
MyPackage.class	8/13/2018 2:54 AM	CLASS File	
MyPackage	8/13/2018 2:54 AM	JAVA File	

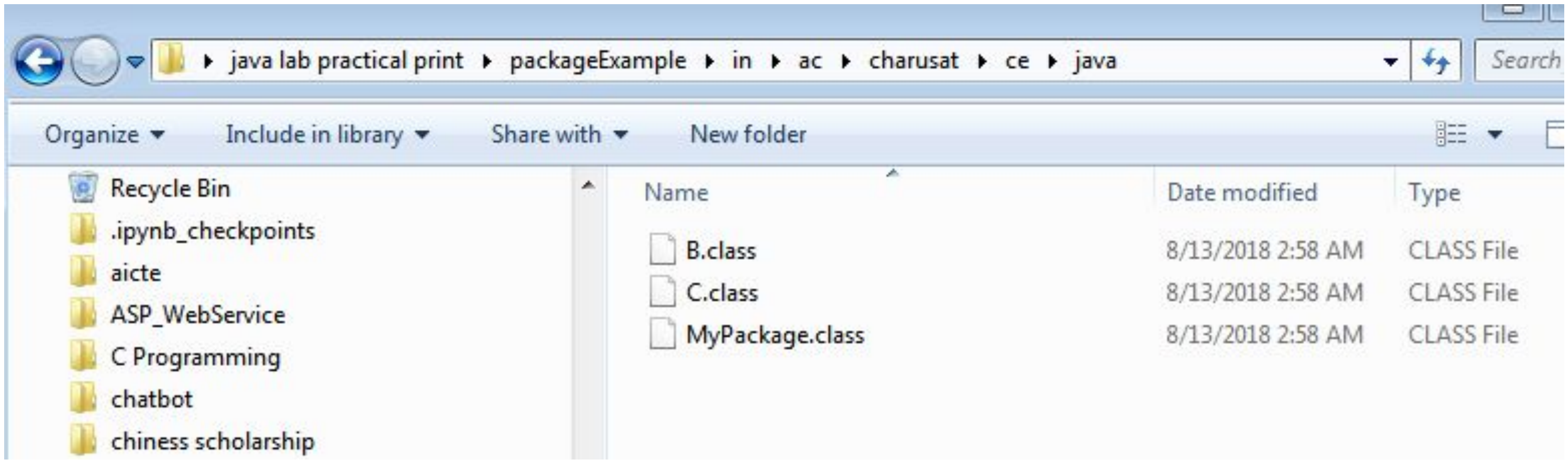
Compilation command

javac –d . MyPackage.java

-d create directory structure

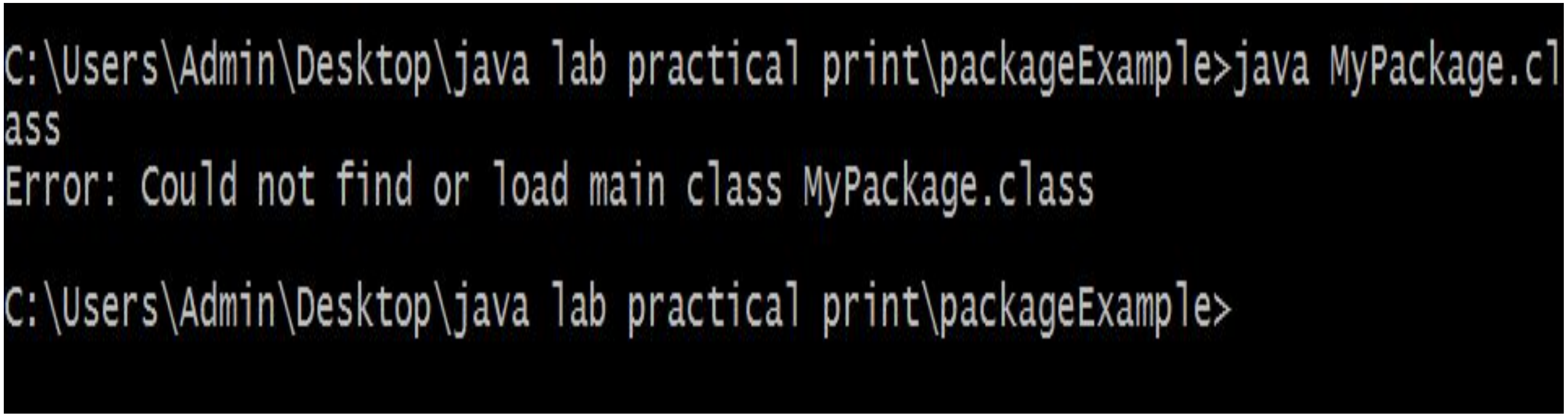
. place the folders structure in current directory

Check the directory structure after compilation



Name	Date modified	Type
B.class	8/13/2018 2:58 AM	CLASS File
C.class	8/13/2018 2:58 AM	CLASS File
MyPackage.class	8/13/2018 2:58 AM	CLASS File

How to execute the application?

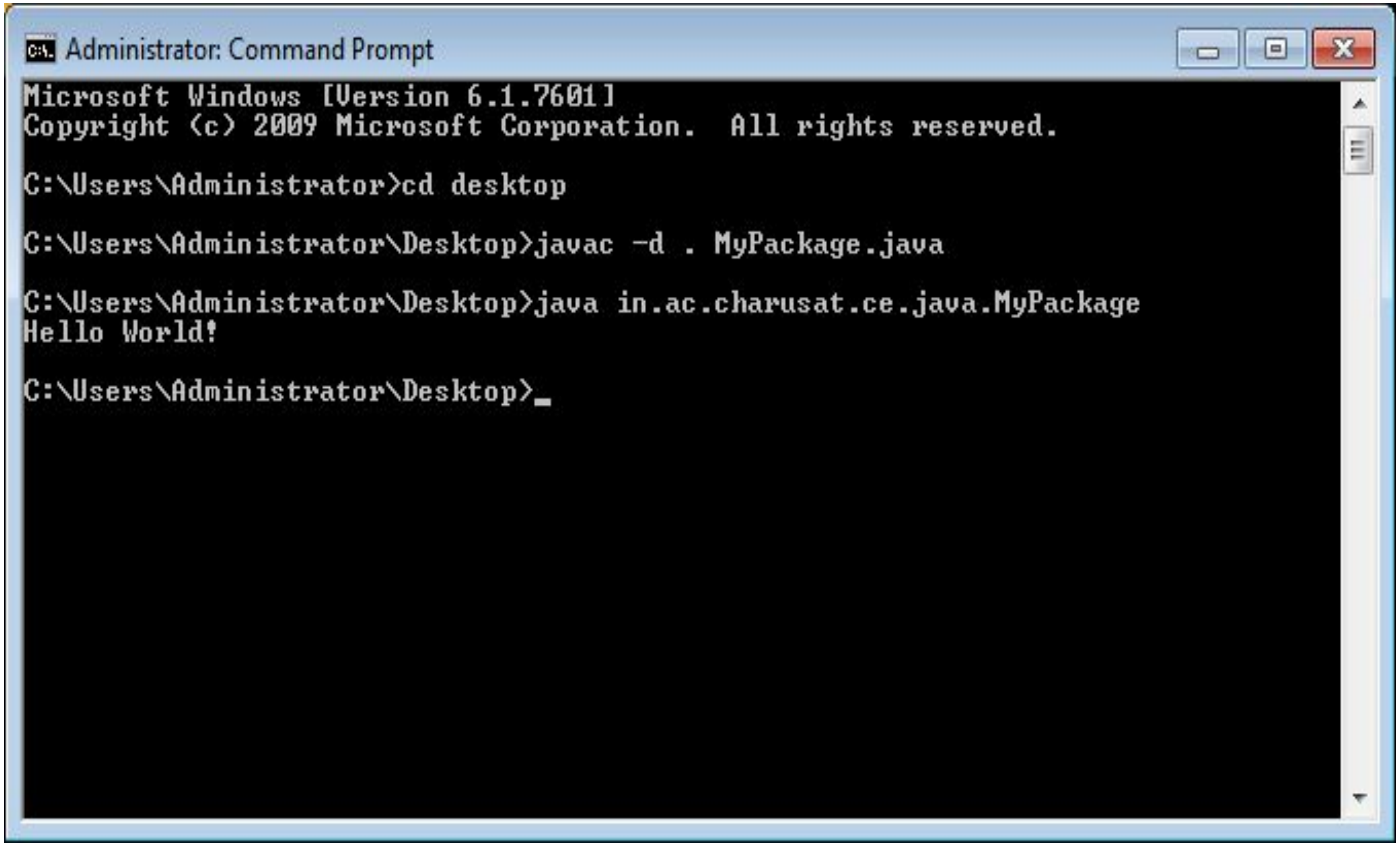


```
C:\Users\Admin\Desktop\java lab practical print\packageExample>java MyPackage.cl
ass
Error: Could not find or load main class MyPackage.class

C:\Users\Admin\Desktop\java lab practical print\packageExample>
```

How to execute the application?

Execute java application with fully qualified name

```
java in.ac.charusat.ce.java.MyPackage
```



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Administrator>cd desktop

C:\Users\Administrator\Desktop>javac -d . MyPackage.java

C:\Users\Administrator\Desktop>java in.ac.charusat.ce.java.MyPackage
Hello World!

C:\Users\Administrator\Desktop>_
```

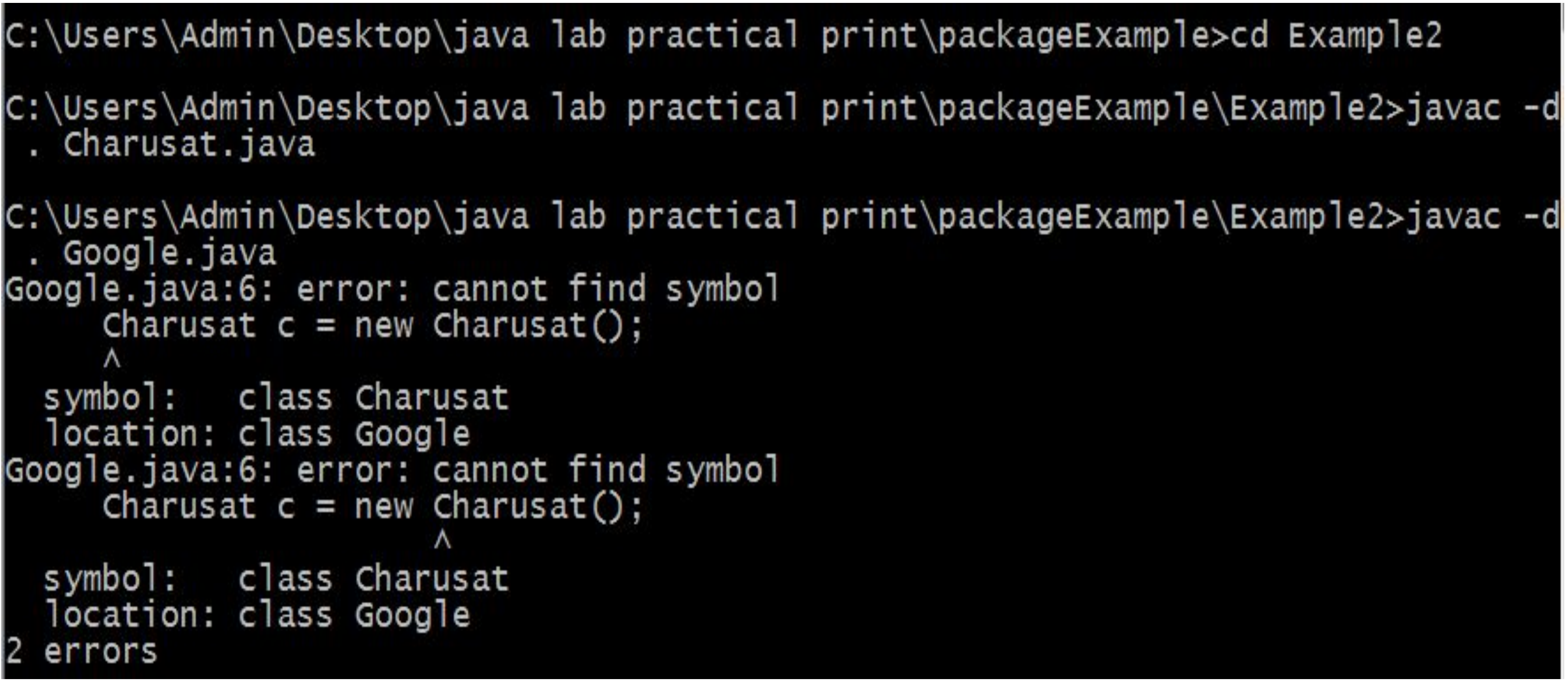
How to create two packages and use one package class inside another package?

Example

```
package in.ac.charusat.state;
class Charusat
{
   void gujaratState()
   {
    System.out.println("Charusat at Gujarat");
   }
}
```

```
package com.google.requiredinfo;
class Google
{
   public static void main(String[] args)
  {
     Charusat c = new Charusat();
     c.gujaratState();
}
}
```

Compile the application



```
C:\Users\Admin\Desktop\java lab practical print\packageExample>cd Example2

C:\Users\Admin\Desktop\java lab practical print\packageExample\Example2>javac -d
 . Charusat.java

C:\Users\Admin\Desktop\java lab practical print\packageExample\Example2>javac -d
 . Google.java
Google.java:6: error: cannot find symbol
     Charusat c = new Charusat();
     ^
  symbol:   class Charusat
  location: class Google
Google.java:6: error: cannot find symbol
     Charusat c = new Charusat();
                      ^
  symbol:   class Charusat
  location: class Google
2 errors
```

There are 2 errors

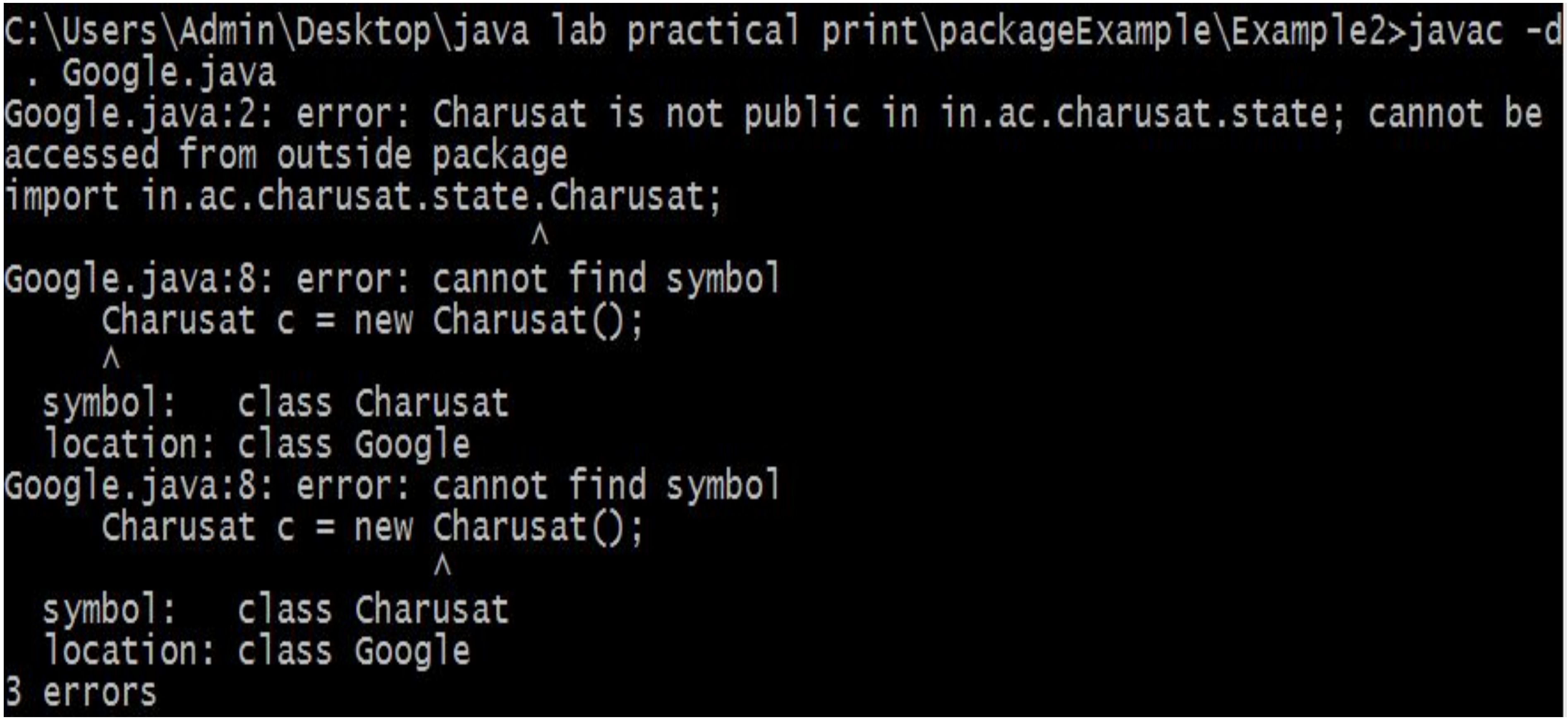
1. Whenever we are using other package classes, we must import the package first of all.

```
package com.google.requiredinfo;
import in.ac.charusat.state.Charusat;
class Google
{
   public static void main(String[] args)
  {
     Charusat c = new Charusat();
     c.gujaratState(); }
}
```

(1)

```
package com.google.requiredinfo;
import in.ac.charusat.state.*;
class Google
{
   public static void main(String[] args)
  {
     Charusat c = new Charusat();
     c.gujaratState(); }
}
```

Will it compile or not?



```
C:\Users\Admin\Desktop\java lab practical print\packageExample\Example2>javac -d
 . Google.java
Google.java:2: error: Charusat is not public in in.ac.charusat.state; cannot be
accessed from outside package
import in.ac.charusat.state.Charusat;
                           ^
Google.java:8: error: cannot find symbol
     Charusat c = new Charusat();
     ^
  symbol:   class Charusat
  location: class Google
Google.java:8: error: cannot find symbol
     Charusat c = new Charusat();
                      ^
  symbol:   class Charusat
  location: class Google
3 errors
```

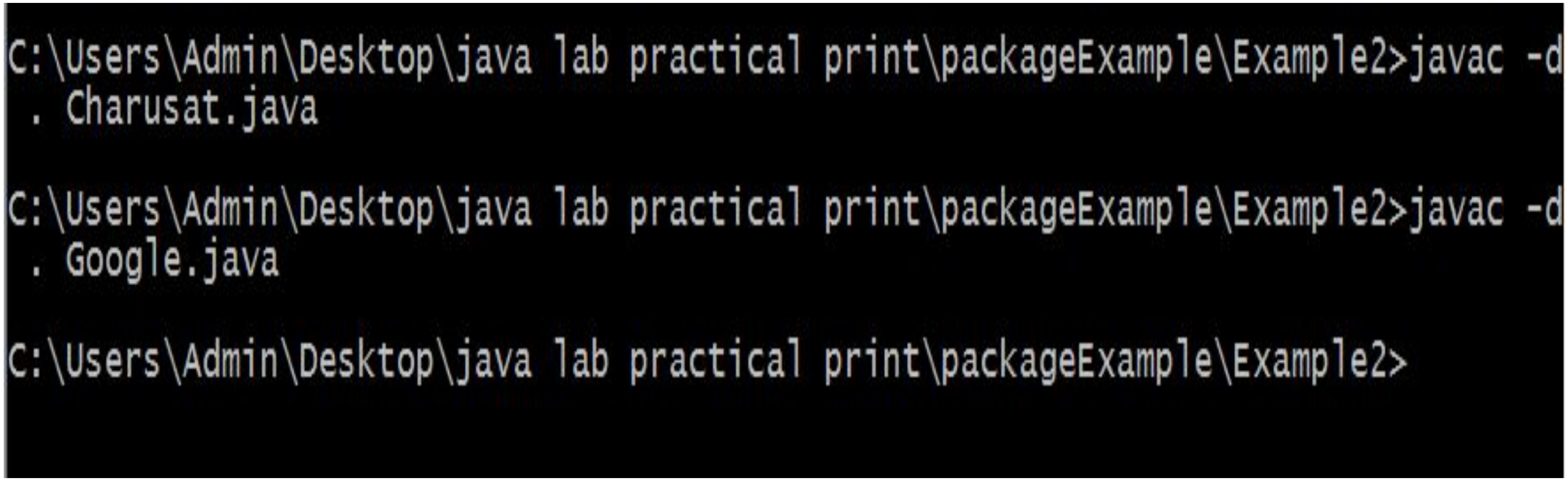
Error -2

Class is not public

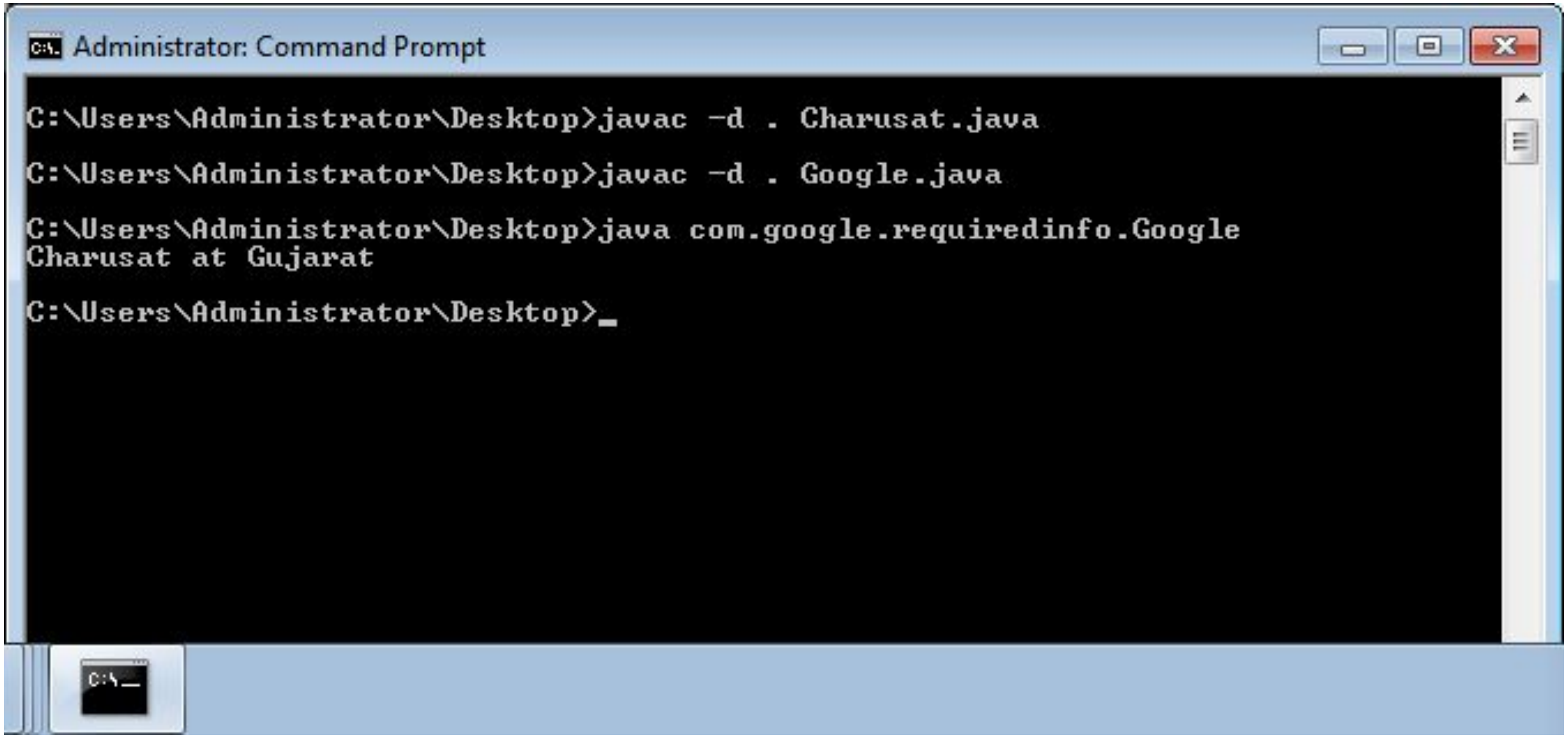
Method is not public

```
package in.ac.charusat.state;
public class Charusat
{
   public void gujaratState()
   {
    System.out.println("Charusat at Gujarat");
   }
}
```

execute it?



```
C:\Users\Admin\Desktop\java lab practical print\packageExample\Example2>javac -d
 . Charusat.java

C:\Users\Admin\Desktop\java lab practical print\packageExample\Example2>javac -d
 . Google.java

C:\Users\Admin\Desktop\java lab practical print\packageExample\Example2>
```



```
C:\Users\Administrator\Desktop>javac -d . Charusat.java

C:\Users\Administrator\Desktop>javac -d . Google.java

C:\Users\Administrator\Desktop>java com.google.requiredinfo.Google
Charusat at Gujarat

C:\Users\Administrator\Desktop>_
```

Homework

Try the same example with protected modifier and compare with default modifier.

Example-2 Creating Sub Packages

- Classes in one ore more source files can be part of the same packages.
- As packages in Java are organised hierarchically, sub-packages can be created as follows:
  - package myPackage.Math
  - package myPackage.secondPakage.thirdPackage
- Store “thirdPackage” in a subdirectory named “myPackage\secondPackage”. Store “secondPackage” and “Math” class in a subdirectory “myPackage”.

Accessing a Package

- As indicated earlier, classes in packages can be accessed using a fully qualified name or using a short-cut as long as we import a corresponding package.
- The general form of importing package is:
  - import package1[.package2][…].classname
  - Example:
    - import myPackage.ClassA;
    - import myPackage.secondPackage.ClassB;
  - All classes/packages from higher-level package can be imported as follows:
    - import myPackage.*;

Do this exercise- Using a Package

- Let us store the code listing below in a file named "ClassA.java" within subdirectory named "myPackage" within the current directory (say "abc").

```
        package myPackage;
        public class ClassA {
         // class body
         public void display()
         {
                        System.out.println("Hello, I am ClassA");
          }
        }
        class ClassB {
         // class body
}
```

Using a Package

- Within the current directory ("abc") store the following code in a file named "ClassX.java"

```
import myPackage.ClassA;

public class ClassX
{
        public static void main(String args[])
        {
                ClassA objA = new ClassA();
                objA.display();
        }
}
```

Compiling and Running

- When ClassX.java is compiled, the compiler compiles it and places .class file in current directly. If .class of ClassA in subdirectory “myPackage” is not found, it comples ClassA also.
- Note: It does not include code of ClassA into ClassX
- When the program ClassX is run, java loader looks for ClassA.class file in a package called “myPackage” and loads it.

Using a Package

- Let us store the code listing below in a file named “ClassA.java” within subdirectory named “secondPackage” within the current directory (say “abc”).

```
package secondPackage;
public class ClassC {
 // class body
        public void display()
        {
                System.out.println("Hello, I am ClassC");
        }
}
```

Using a Package

- Within the current directory (“abc”) store the following code in a file named “ClassY.java”

```
import myPackage.ClassA;
import secondPackage.ClassC;
public class ClassY
{
	public static void main(String args[])
	{
		ClassA objA = new ClassA();
		ClassC objC = new ClassC();
		objA.display();
		objC.display();
	}
}
```

Output

```
Hello, I am ClassA
Hello, I am ClassC
```

Protection and Packages

- All classes (or interfaces) accessible to all others in the same package.
- Class declared public in one package is accessible within another. Non-public class is not
- Members of a class are accessible from a difference class, as long as they are not *private*
- *protected* members of a class in a package are accessible to subclasses in a different class

Visibility - Revisited

- *Public* keyword applied to a class, makes it available/visible everywhere. Applied to a method or variable, completely visible.
- *Private* fields or methods for a class only visible within that class. Private members are *not* visible within subclasses, and are *not* inherited.
- *Protected* members of a class are visible within the class, subclasses and *also* within all classes that are in the same package as that class.

Visibility Modifiers

Accessible to:	public	protected	Package (default)	private
Same Class	Yes	Yes	Yes	Yes
Classin package	Yes	Yes	Yes	No
Subclassin different package	Yes	Yes	No	No
Non-subclass different package	Yes	No	No	No

- 1) Private

The private access modifier is accessible only within the class.

```
class A{
private int data=40;
private void msg(){System.out.println("Hello java");}
}
public class Simple{
 public static void main(String args[]){
   A obj=new A();
   System.out.println(obj.data);//Compile Time Error
   obj.msg();//Compile Time Error
   }
}
```

Role of Private Constructor

If you make any class constructor private, you cannot create the instance of that class from outside the class. For example:

```
class A{
private A(){}//private constructor
void msg(){System.out.println("Hello java");}
}
public class Simple{
 public static void main(String args[]){
   A obj=new A();//Compile Time Error
 }
```

2) Default

If you don't use any modifier, it is treated as **default** by default.

The default modifier is accessible only within package.

It cannot be accessed from outside the package. It provides more accessibility than private.

But, it is more restrictive than protected, and public.

```java
//save by A.java
package pack;
class A{
  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();//Compile Time Error
   obj.msg();//Compile Time Error
  }
}
```

the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

3) Protected

- The **protected access modifier** is accessible within package and outside the package but through inheritance only.
- The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.
- It provides more accessibility than the default modifier.

```
//save by A.java
package pack;
public class A{
protected void msg(){System.out.println("Hello");}
}

//save by B.java
package mypack;
import pack.*;

class B extends A{
 public static void main(String args[]){
   B obj = new B();
   obj.msg();
 }
}
```

4) Public

The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

```
//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();
  }
}
```

Adding a Class to a Package

- Consider an existing package that contains a class called "Teacher":

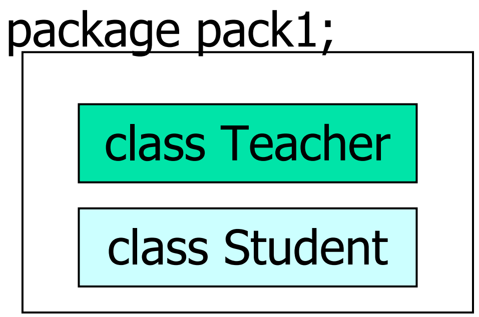
```
package pack1;
public class Teacher
{
  // class body
}
```

- This class is stored in "Teacher.java" file within a directory called "pack1".
- How do we add new public class called "Student" to this package.

Adding a Class to a Package

- Define the public class "Student" and place the package statement before the class definition as follows:

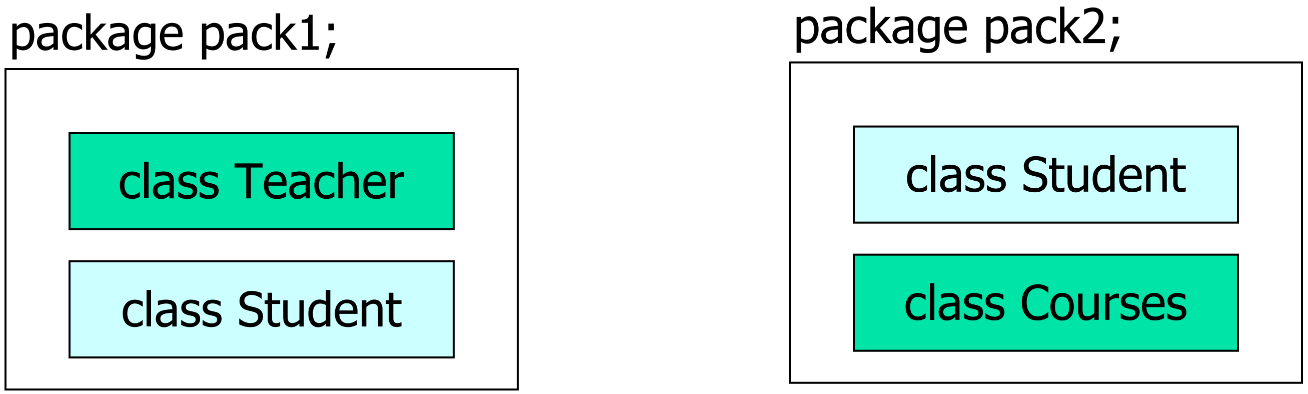
```
package pack1;
public class Student
{
  // class body
}
```



- Store this in "Student.java" file under the directory "pack1".
- When the "Student.java" file is compiled, the class file will be created and stored in the directory "pack1". Now, the package "pack1" will contain both the classes "Teacher" and "Student".

Packages and Name Clashing

- When packages are developed by different organizations, it is possible that multiple packages will have classes with the same name, leading to name clashing.



- We can import and use these packages like:
  - import pack1.*;
  - import pack2.*;
  - Student student1; // Generates compilation error

Handling Name Clashing

- In Java, name classing is resolved by accessing classes with the same name in multiple packages by their fully qualified name.
- Example:

```
import pack1.*;
import pack2.*;
pack1.Student student1;
pack2.Student student2;
Teacher teacher1;
Courses course1;
```

Extending a Class from Package

- A new class called “Professor” can be created by extending the “Teacher” class defined the package “pack1” as follows:

```
import pack1.Teacher;
public class Professor extends Teacher
{
  // body of Professor class
  // It is able to inherit public and protected members,
  // but not private or default members of Teacher class.
}
```

Summary

- Packages allow grouping of related classes into a single united.
- Packages are organised in hierarchical structure.
- Packages handle name classing issues.
- Packages can be accessed or inherited without actual copy of code to each program.

Any Question???