Sri SAI RAM ENGINEERING COLLEGE
INSTITUTE OF TECHNOLOGY
West Tambaram, Chennai - 44

SAIRAM DIGITAL RESOURCES

YEAR II
SEM III

CS8392

OBJECT ORIENTED PROGRAMMING
(Common to CSE, EEE, EIE, ICE, IT)

UNIT NO 1

INTRODUCTION TO OOP AND JAVA FUNDAMENTALS

1.9 Packages, JavaDoc Comments

COMPUTER SCIENCE & ENGINEERING

PACKAGE

- Is a collection of classes and interfaces
- Act as a folder
- A java package is a group of similar types of classes, interfaces and sub-packages.
- **Merits:**
  - Java package is used to categorize the classes and interfaces so that they can be easily maintained.
  - Java package provides access protection.
  - Java package removes naming collision.
  - **Packages** provide reusability of code .
  - We can crate our own **Package** or extend already available **Package.**

Package Contd..

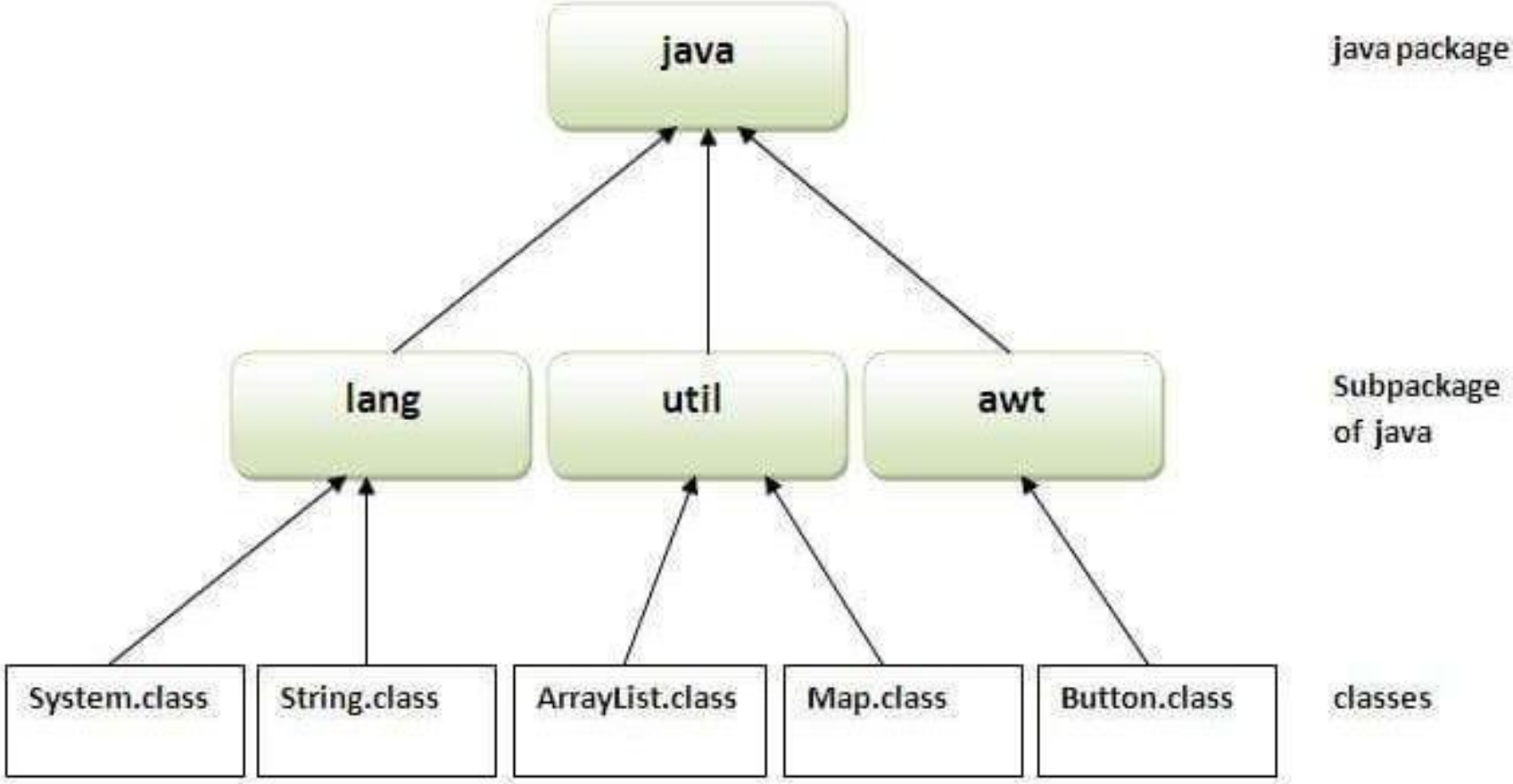
- Packages in Java is a mechanism to encapsulate a group of classes, interfaces

  and sub packages.

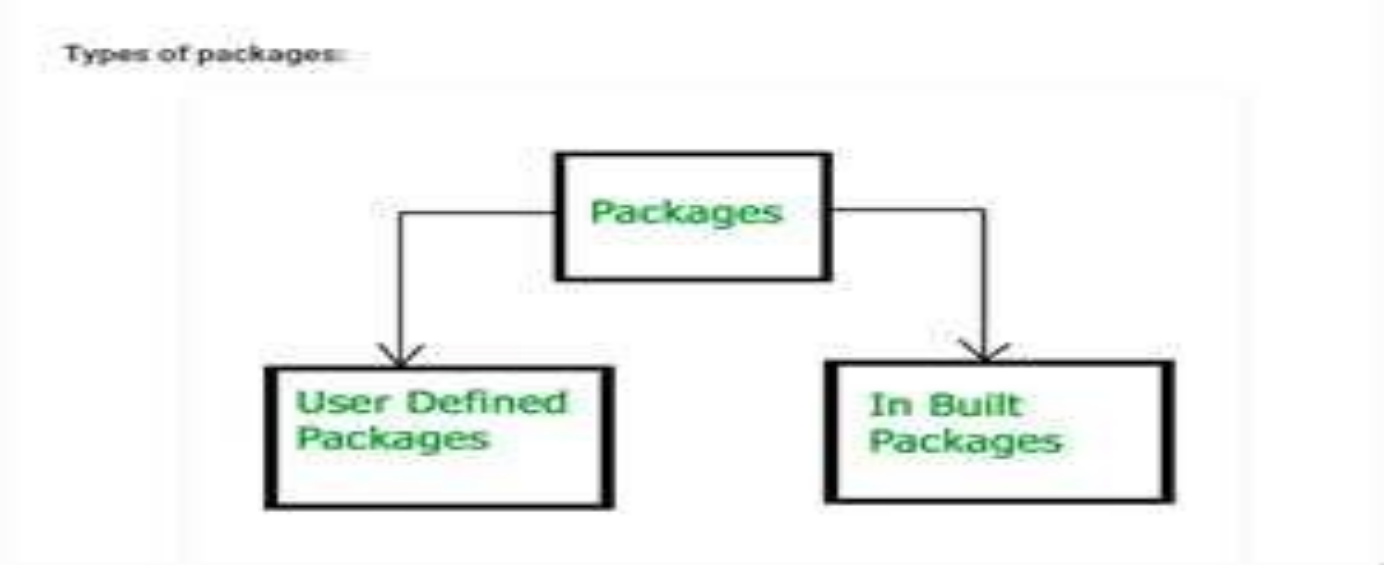
- In java there are already many predefined packages that we use while programming.

  - For example: java.lang, java.io, java.util etc.

- One of the most useful feature of java is that we can define our own packages


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

- Types of package:
  - 1) User defined package: The package we create is called user-defined package.
  - 2) Built-in package: The already defined package like java.io.*, java.lang.* etc are known as built-in packages.



Types of Packages

- **Built in Package**
  - java.io — I/O streams,scanners
  - java.net — socket, HttpCookie
  - java,awt — buttons,controls
  - java.applet — applet (client side app)
  - java.util — datastructures,date,time
  - java.lang — string, exception,thread
  - java.sql — connection, driver manager
- **User defined Package**
  - User can create their own packages using package keyword

    package packagename

    import package;

Defining a Package:

- This statement should be used in the beginning of the program to include that program in that particular package.

  package <package name>;

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

To Compile: javac -d directory javafilename.java

To Run: java packagename.classname

- The -d switch specifies the destination where to put the generated class file
- If you want to keep the package within the same directory, you can use . (dot).

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.

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.

Using packagename.classname

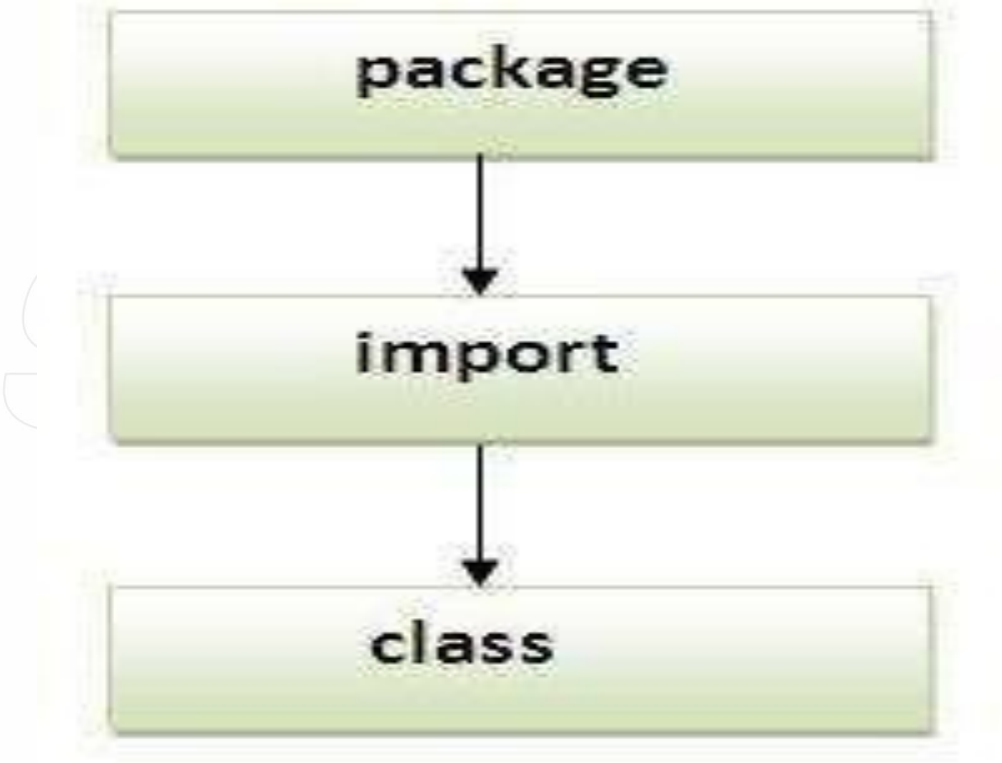
- If you import package.classname then only declared class of this package will be accessible.

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.

Note: 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 subpackage as well.

Sequence of the program must be package then import then class.



Packaging up multiple classes

Package with Multiple Public Classes

- A Java source file can have only one class declared as **public**, we cannot put two or more public classes together in a**.java** file. This is because of the restriction that the file name should be same as the name of the public class with **.java**extension.

- If we want to multiple classes under consideration are to be declared as **public**, we have to store them in **separate source files** and attach the **package** statement as the first statement in those source files.

```
//Save as A.java  package
javapoint;  Public class B{}
```

```
//Save as B.java  package
java  public class a{}
```

CLASSPATH

- It is a environmental variable, which contains the path for the default-working directory (.).
- The specific location that java compiler will consider, as the root of any package hierarchy is, controlled by Classpath
- The package name is closely associated with the directory structure used to store the classes. The classes (and other entities) belonging to a specific package are stored together in the same directory.
- There is a scenario, I want to put the class file of A.java source file in classes folder of c: drive. For example:

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

To Compile:

```
e:\sources> javac -d c:\classes Simple.java
```

To Run:

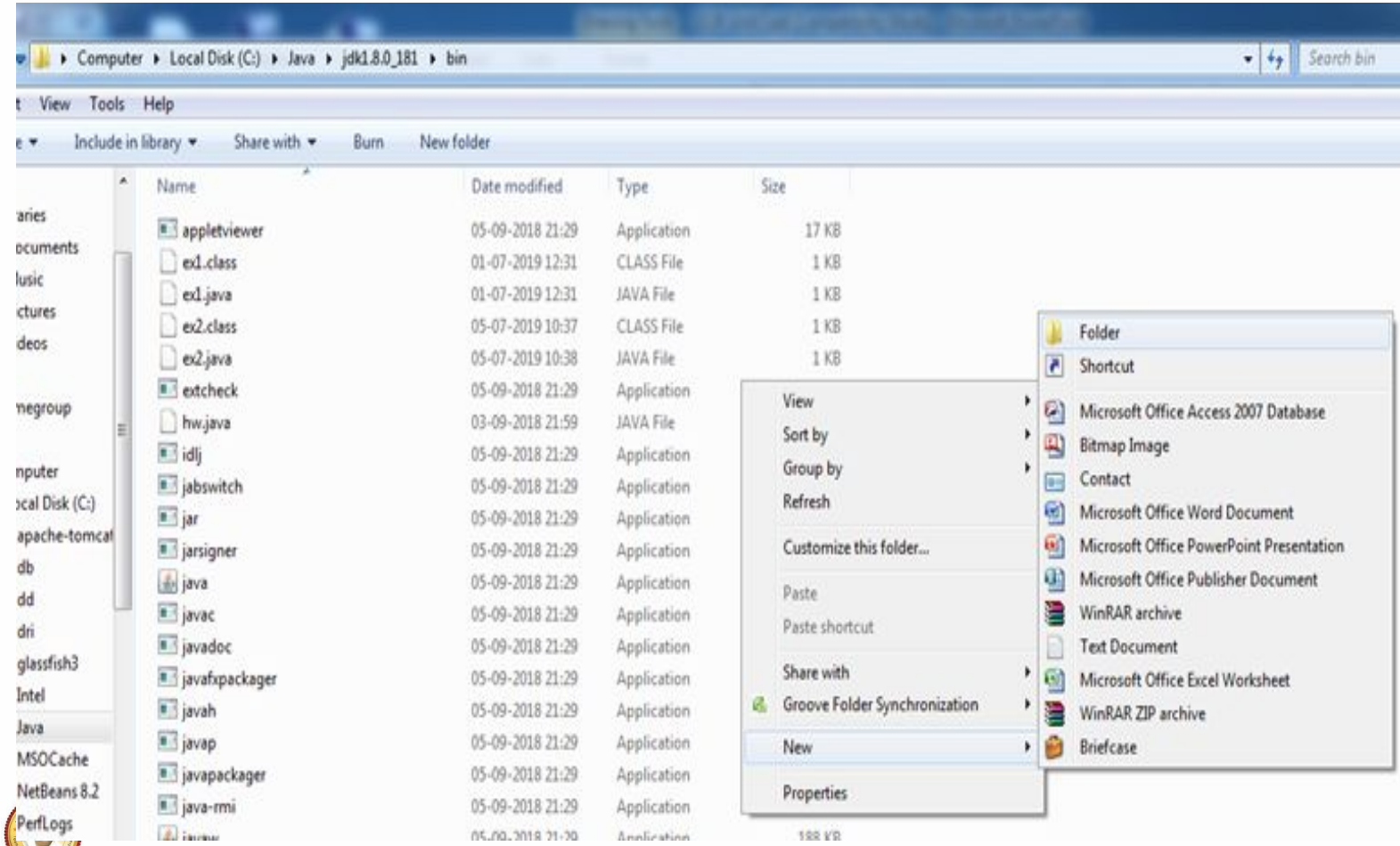
To run this program from e:\source directory, you need to set classpath of the directory where the class file resides.

```
e:\sources> set classpath=c:\classes;
e:\sources> java mypack.Simple
```

Step 1:

Create a package

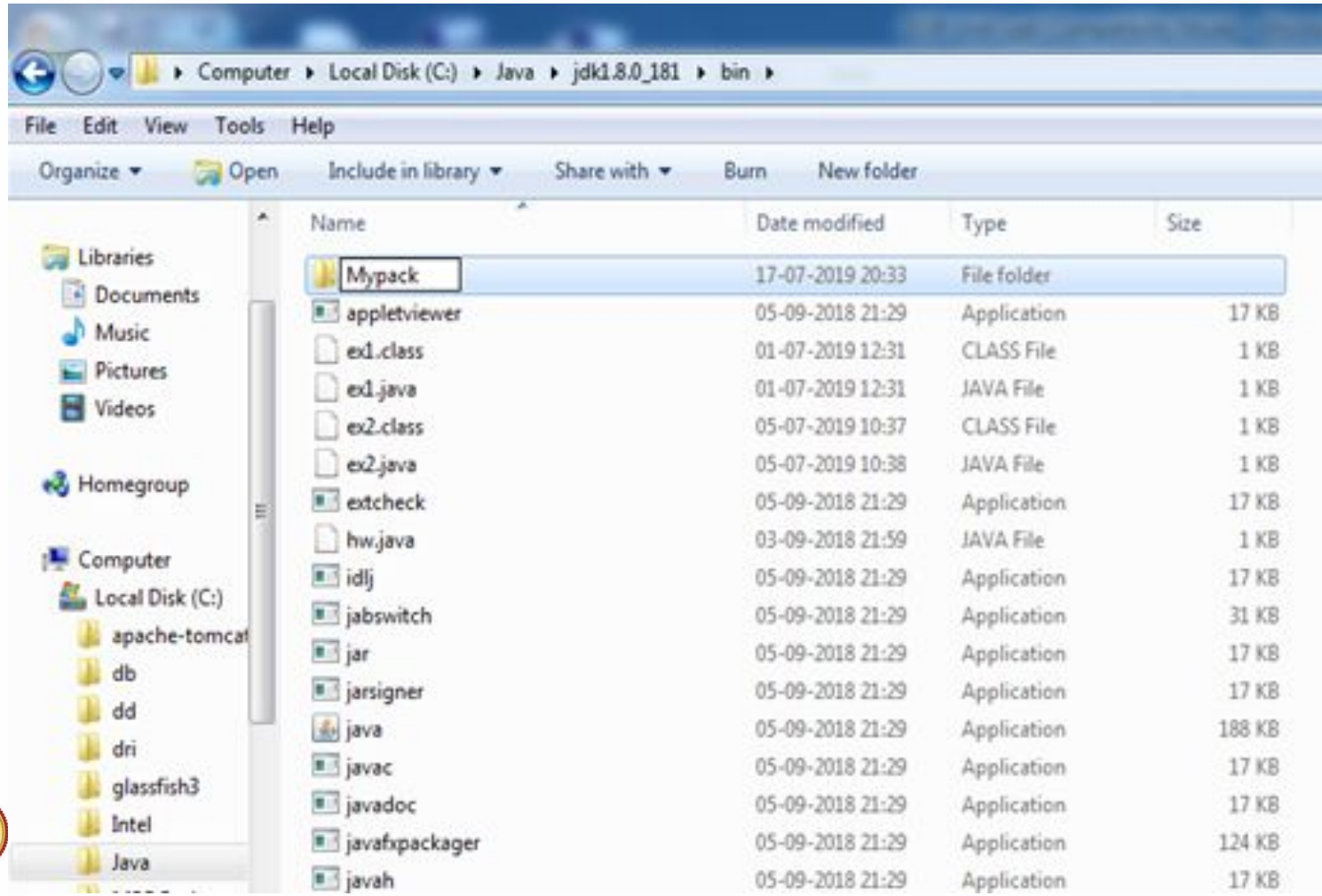
Create a new folder in jdk\bin



Step2:

Set package name

Mypack is the package name



Step 3:

Create a class within the package

Save the class in Mypack package

File Edit Search View Options Help

C:\Java\jdk1.8.0_181\bin\Mypack\factorial.java

```
package Mypack;
public class factorial
{
public int fact(int n)
{
if (n==1)
        return 1;
else
        return (n*fact(n-1));
}
}
```

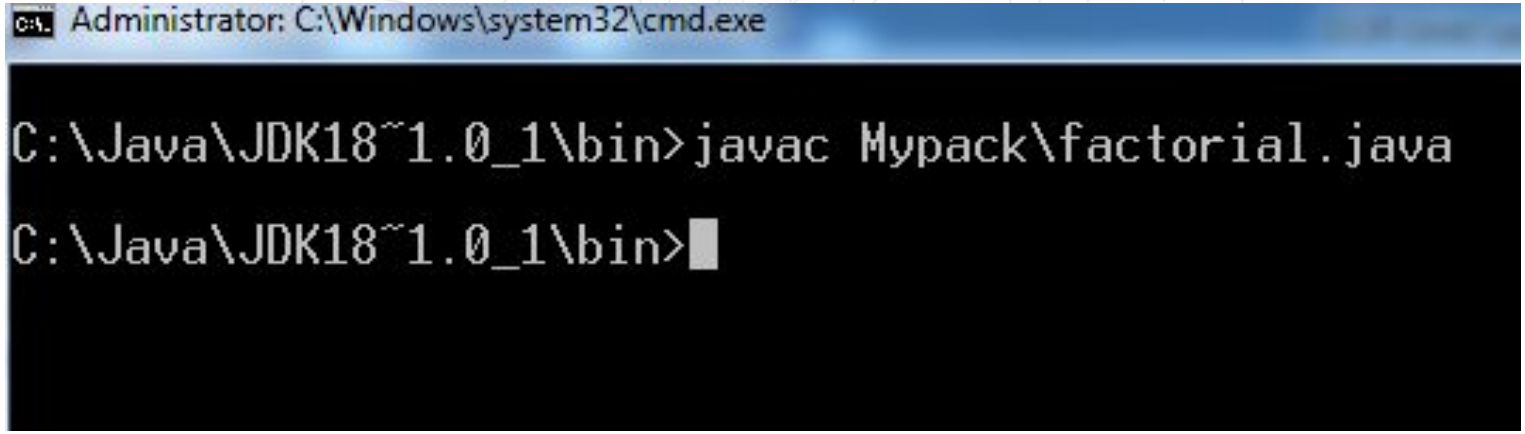
Step 4:

Compile the class

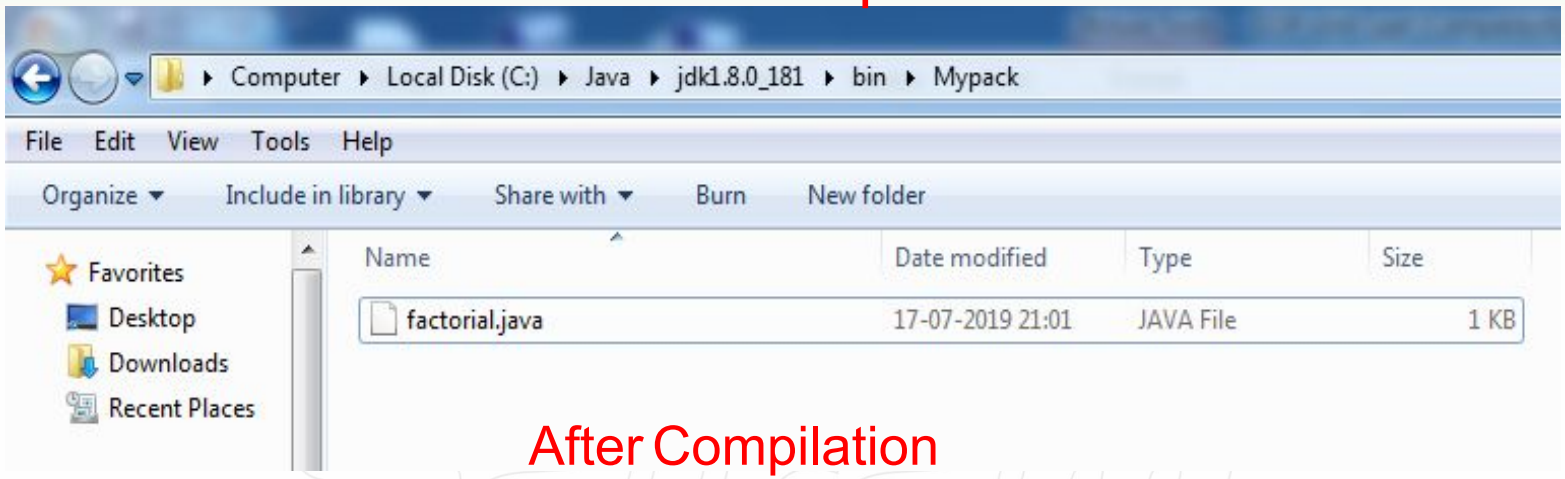
Quit the Editor

Type the command,

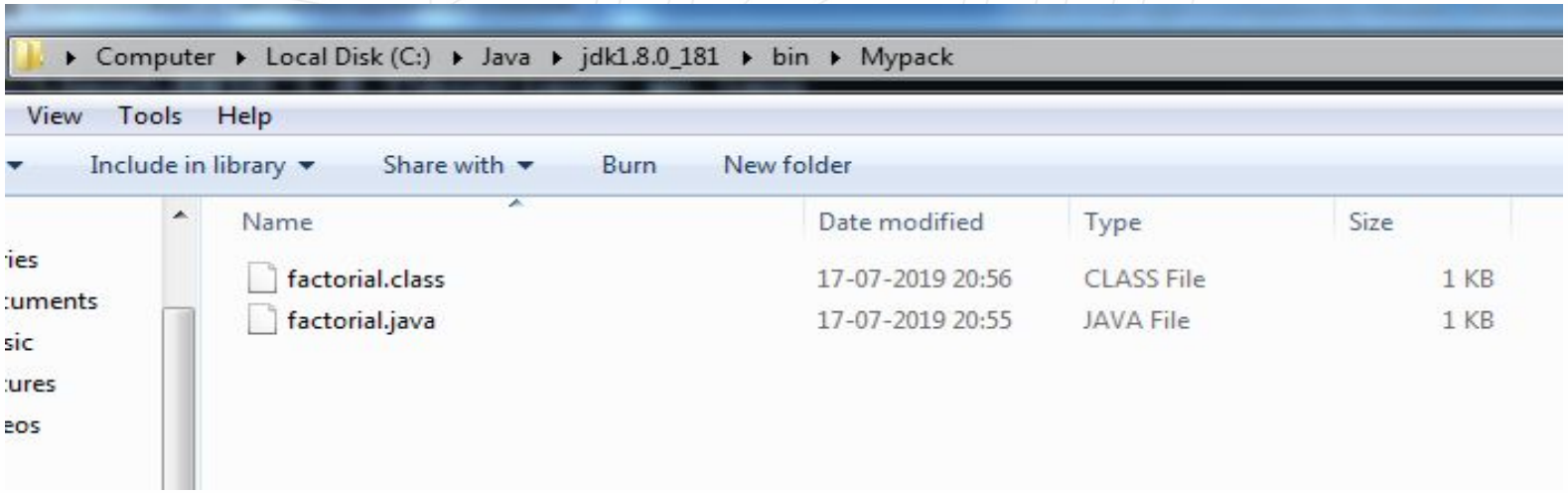
C:\java\jdk\bin> javac **Mypack\factorial.java**



Before Compilation



After Compilation



- Import the package
  - Create a class to import the package Mypack
  - Compile & run the class

File Edit Search View Options Help

C:\Java\jdk1.8.0_181\bin\mn.java

```
import Mypack.*;
import java.io.*;
class mn
{
        public static void main(String args[])
        {
                int ans;
                factorial obj=new factorial();
                ans=obj.fact(5);
                System.out.println("Answer ="+ans);
        }
}
```

Administrator: C:\Windows\system32\cmd.exe

```
C:\Java\JDK18~1.0_1\bin>edit

C:\Java\JDK18~1.0_1\bin>javac mn.java

C:\Java\JDK18~1.0_1\bin>java mn
Answer =120

C:\Java\JDK18~1.0_1\bin>
```

JAVADOC COMMENTS

- generating Java code documentation API in HTML format from Java source code.
- Javadoc comments are any multi-line comments ("/** ... */")
- EXAMPLE

```
/** The Calculator class provides methods to get addition and subtract  ion of given 2 numbers.*/
 public class Calculator {
 /** The add() method returns addition of given  numbers.*/
   public static int add(int a, int b){return a+b;}
 /** The sub() method returns subtraction of given numbers.*/  public static int sub(int a,
int b){return a-b;}
}
```

Structure of a Javadoc comment

- A Javadoc comment is set off from code by standard multi-line comment tags /* and */.
- The opening tag (called begin-comment delimiter), has an extra asterisk, as in /**.
- The first paragraph is a description of the method documented.
- Following the description are a varying number of descriptive tags, signifying:
  - The parameters of the method (@param)
  - What the method returns (@return)
  - Any exceptions the method may throw (@throws)
  - Other less-common tags such as @see (a "see also" tag)

Overview of Javadoc

- The basic structure of writing document comments is to embed them inside /** ... */.
- The Javadoc is written next to the items without any separating newline.
- Note that any import statements must precede the class declaration. The class declaration usually contains:

```
// import statements /**

* @author        Firstname Lastname <address @
                 example.com>
* @version       1.6              (current version number of
                                  program)
* @since         2010-03-31
                                  (the version of the package this
  class was first added
  to)
  */

  public class Test {   // class body}
```

Javadoc tags

Tag & Parameter	Usage	Applies to
@author *John Smith*	Describes an author.	Class, Interface, Enum
@version *version*	Provides software version entry. Max one per Class or Interface.	Class, Interface, Enum
@since *since-text*	Describes when this functionality has first existed.	Class, Interface, Enum, Field, Method
@see *reference*	Provides a link to other element of documentation.	Class, Interface, Enum, Field, Method
@param *name description*	Describes a method parameter.	Method
@return *description*	Describes the return value.	Method

Examples

```
/**
* The HelloWorld program implements an application that
* simply displays "Hello World!" to the standard output.
*
* @author  Zara Ali
* @version 1.0
* @since   2014-03-31
*/
public class HelloWorld {
    public static void main(String[] args) {
        /* Prints Hello, World! on standard output.
        System.out.println("Hello World!");
    }
}
```

- You can include required HTML tags inside the description part,
- For example, below example makes use of <h1>....</h1> for heading and <p> has been used for creating paragraph break:

```
/**
* <h1>Hello, World!</h1>
* The HelloWorld program implements an application that
* simply displays "Hello World!" to the standard output.
* <p>
* Giving proper comments in your program makes it more
* user friendly and it is assumed as a high quality code.
*
*
* @author  Zara Ali
* @version 1.0
* @since   2014-03-31
*/
public class HelloWorld {
    public static void main(String[] args) {
        /* Prints Hello, World! on standard output.
        System.out.println("Hello World!");
    }
}
```