

Java Packages

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

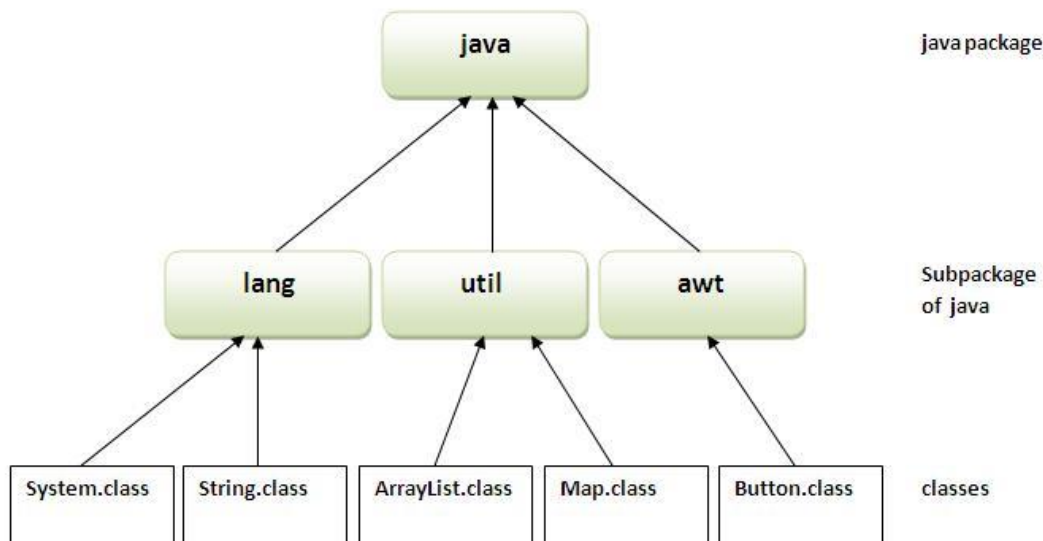
A **java package** is a group of similar types of classes, interfaces, and sub-packages.

Packages in java can be categorized in two forms:

- built-in packages like lang, util, awt, javax, swing, net, io, , sql etc.-predefined packages
- user-defined package

Advantages of Java Package

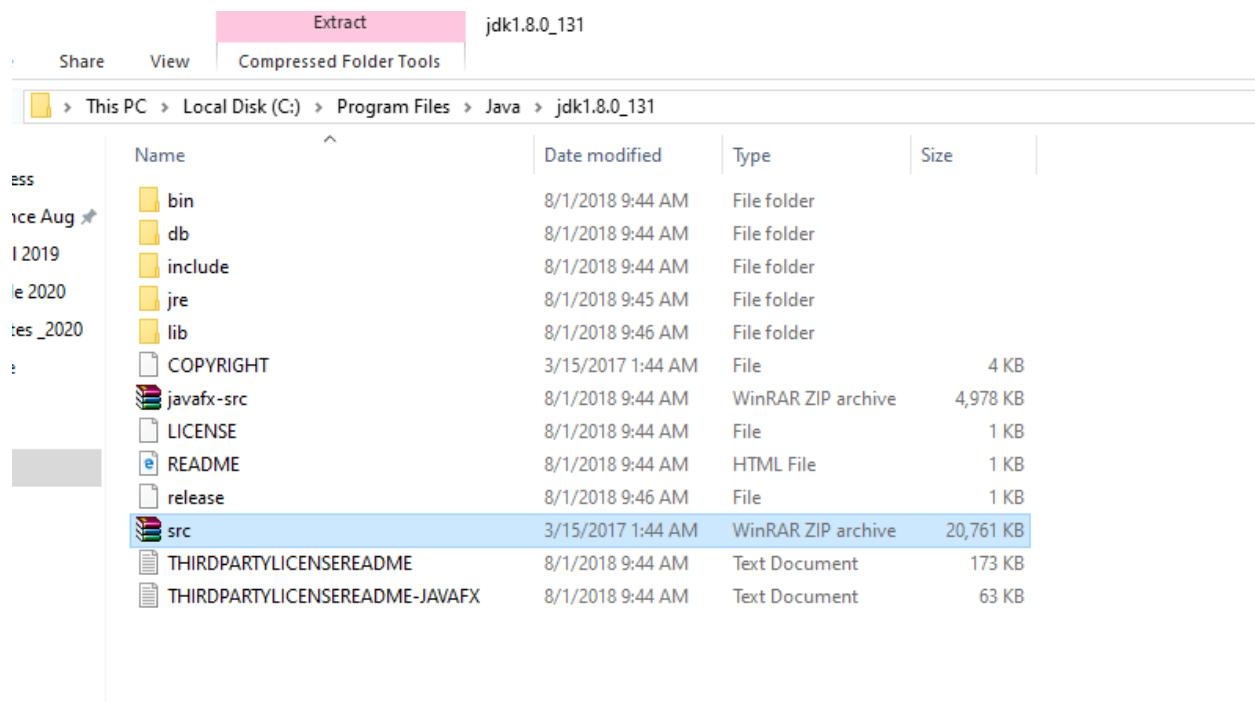
- Java package is used to categorize the classes and interfaces so that they can be easily maintained.
- Java package provides access protection/security to the classes and interfaces. So that an outside person cannot access it directly.
- It improves the modularity of the application.
- Java package removes naming collision.



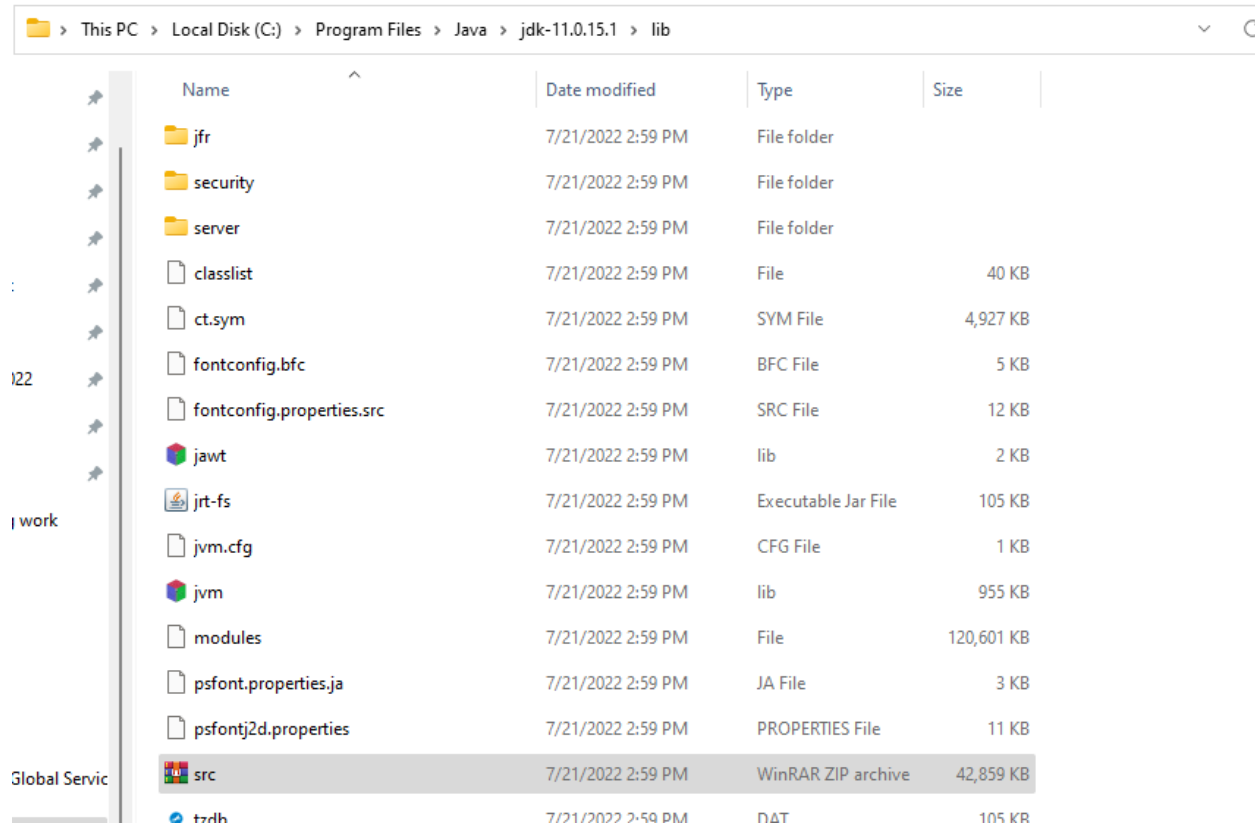
If you want to see the .class files, they're in **lib\rt.jar** in the JRE directory (.jar is the same as .zip, so you can open it with anything that can open zip files).

> This PC > Local Disk (C:) > Program Files > Java > jre1.8.0_131 > lib					
	Name	Date modified	Type	Size	
ss	content-types	8/1/2018 9:50 AM	Properties Source ...	6 KB	
ce Aug	charsets	8/1/2018 9:50 AM	Executable Jar File	2,966 KB	
2019	classlist	8/1/2018 9:50 AM	File	83 KB	
e 2020	content-types	8/1/2018 9:50 AM	Properties Source ...	6 KB	
es_2020	currency.data	8/1/2018 9:50 AM	DATA File	5 KB	
	deploy	8/1/2018 9:50 AM	Executable Jar File	4,921 KB	
	flavormap	8/1/2018 9:50 AM	Properties Source ...	4 KB	
	fontconfig.bfc	8/1/2018 9:50 AM	BFC File	4 KB	
	fontconfig.properties.src	8/1/2018 9:50 AM	SRC File	11 KB	
	hijrah-config-umalqura	8/1/2018 9:50 AM	Properties Source ...	14 KB	
	javafx	8/1/2018 9:50 AM	Properties Source ...	1 KB	
	javaws	8/1/2018 9:50 AM	Executable Jar File	919 KB	
	jce	8/1/2018 9:50 AM	Executable Jar File	114 KB	
	jfr	8/1/2018 9:50 AM	Executable Jar File	548 KB	
	jfxswt	8/1/2018 9:50 AM	Executable Jar File	34 KB	
	jsse	8/1/2018 9:50 AM	Executable Jar File	570 KB	
	jvm.hprof	8/1/2018 9:50 AM	Text Document	5 KB	
	logging	8/1/2018 9:50 AM	Properties Source ...	3 KB	
	management-agent	8/1/2018 9:50 AM	Executable Jar File	1 KB	
	meta-index	8/1/2018 9:50 AM	File	3 KB	
	net	8/1/2018 9:50 AM	Properties Source ...	5 KB	
	plugin	8/1/2018 9:50 AM	Executable Jar File	1,878 KB	
	psfont.properties.ja	8/1/2018 9:50 AM	JA File	3 KB	
	psfontj2d	8/1/2018 9:50 AM	Properties Source ...	11 KB	
	resources	8/1/2018 9:50 AM	Executable Jar File	3,411 KB	
	rt	8/1/2018 9:50 AM	Executable Jar File	53,231 KB	
	sound	8/1/2018 9:50 AM	Properties Source ...	2 KB	
	tzdh.dat	8/1/2018 9:50 AM	DAT File	104 KB	

To access the source code, please refer to the "src.zip" file located within the JDK directory.



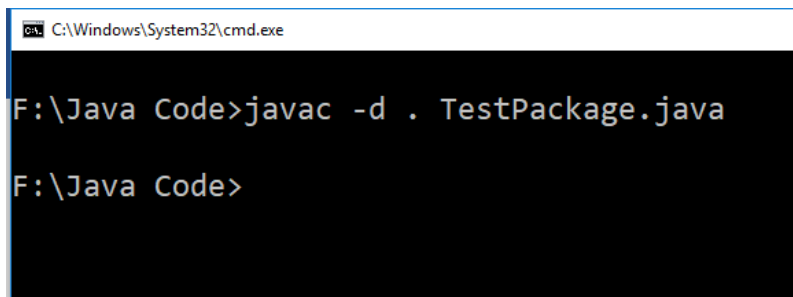
In Java 9
C:\Program Files\Java\jdk-11.0.15.1\lib



My first Java package

```
package mypackage;
public class TestPackage{
    public static void main(String args[]){
        System.out.println("This is my package");
    }
}
```

Compilation command:



```
C:\Windows\System32\cmd.exe

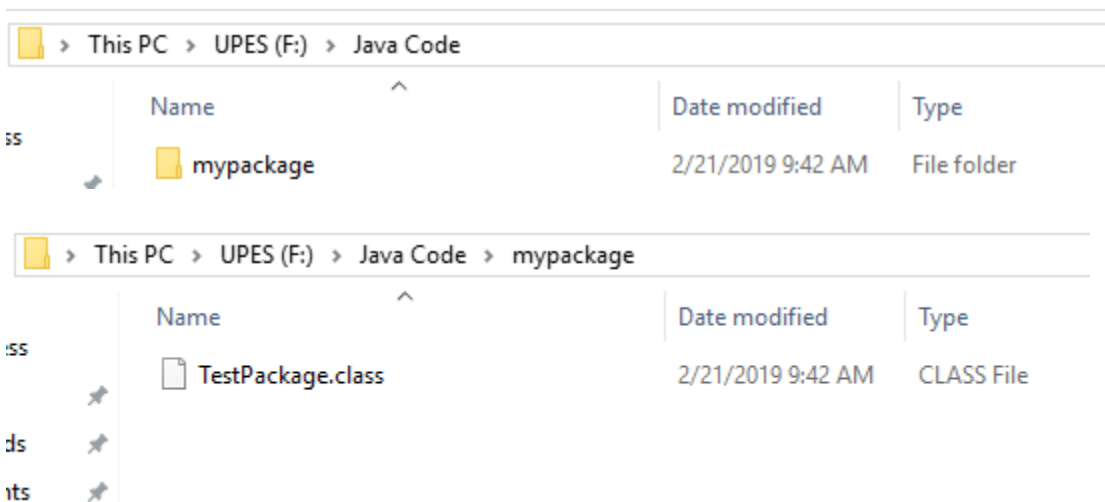
F:\Java Code>javac -d . TestPackage.java

F:\Java Code>
```

The -d is a switch that tells the compiler where to put the class file i.e. it represents the destination. And . (dot) represents the current working directory/folder.

After the compilation of this program, the compiler will generate two things:

1. mypackage folder at **F:\Java Code** location
2. TestPackage.class file at mypackage folder.



Some Wrong Commands:

```
F:\Java Code>javac-d.TestPackage.java
'javac-d.TestPackage.java' is not recognized as an int
operable program or batch file.

F:\Java Code>javac -d.TestPackage.java
javac: file not found: -d.TestPackage.java
Usage: javac <options> <source files>
use -help for a list of possible options

F:\Java Code>javac -d .TestPackage.java
javac: directory not found: .TestPackage.java
Usage: javac <options> <source files>
use -help for a list of possible options
```

Execution command:

```
F:\Java Code>javac -d . TestPackage.java

F:\Java Code>java mypackage.TestPackage
This is my package
```

Creation of package at user-defined location:

```
package mypackage1;
public class TestPackage1{
    public static void main(String args[]){
        System.out.println("This is my package1");
    }
}
```

Valid:

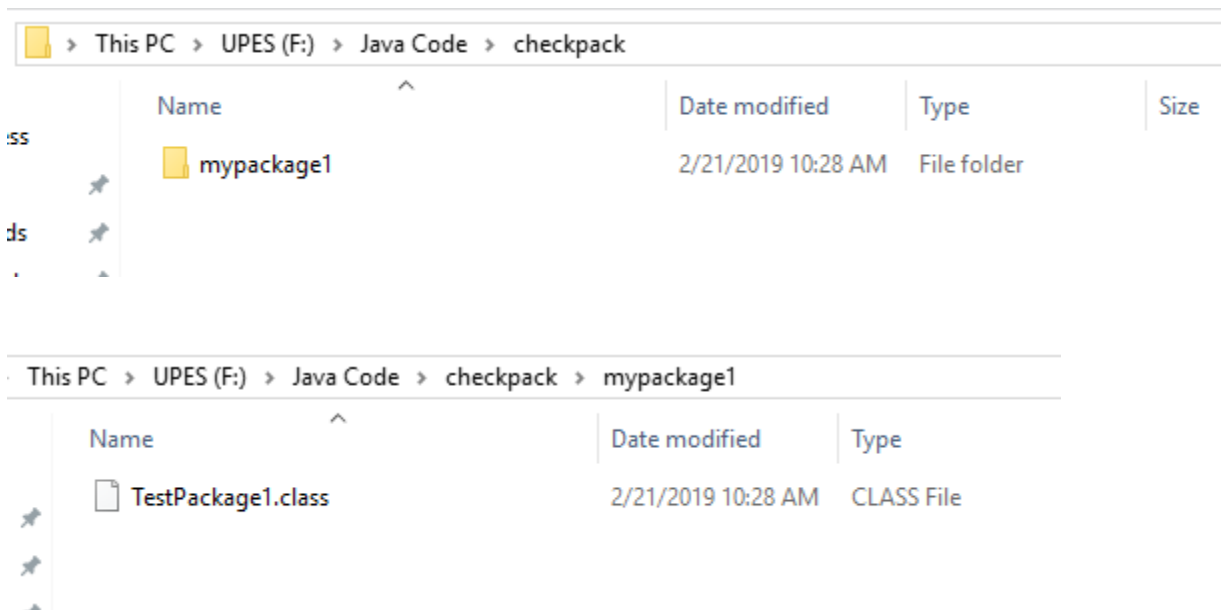
```
F:\Java Code>javac -d "F:\Java Code\checkpack" TestPackage1.java  
F:\Java Code>
```

Invalid command :

```
F:\Java Code\checkpack>javac -d F:\Java Code\checkpack TestPackage1.java  
javac: invalid flag: Code\checkpack  
Usage: javac <options> <source files>  
use -help for a list of possible options
```

After the compilation of this program, the compiler will generate two things:

1. Mypackage1 folder at **F:\Java Code\checkpack** location
2. TestPackage1.class file at mypackage1 folder.



```
F:\Java Code>cd checkpack  
  
F:\Java Code\checkpack>java mypackage1.TestPackage1  
This is my package1
```

There are three techniques to access the package from outside the package. (From one package to another package)

1. **import package.*;**

The statement `import package.*;` is used in Java to import all the classes and interfaces from a specific package into your current Java file.

Creating a package named “upes”

```
1 package upes;
2 public class Ifm{
3     public void ifmData()
4     {
5         System.out.println("Data of IFM Students");
6     }
7 }
```

```
F:\Java Code>javac -d . Ifm.java
```

Creating another package named “adminupes” and importing “upes” package.

```
package adminUpes;
import upes.*;

class Admin{
    public static void main(String args[]){
        Ifm i = new Ifm();
        i.ifmData();
    }
}
```

```
F:\Java Code>javac -d . Admin.java
```

```
F:\Java Code>java adminUpes.Admin
Data of IFM Students
```

2. import package.classname;

The statement `import package.classname;` is used in Java to specifically import a single class (classname) from a specific package (package).

Creation of package named “upes”

```
1 package upes;
2 public class Ifm{
3     public void ifmData()
4     {
5         System.out.println("Data of IFM Students");
6     }
7 }
```

Creation of package named “adminUpes” and importing “Ifm” class from “upes” package

```
1 package adminUpes;
2 import upes.Ifm;
3
4 class Admin{
5     public static void main(String args[]){
6         Ifm i = new Ifm();
7         i.ifmData();
8     }
9 }
```

Output:

```
F:\Java Code>javac -d . Ifm.java
F:\Java Code>javac -d . Admin.java
F:\Java Code>java adminUpes.Admin
Data of IFM Students
```


3. fully qualified name.

Fully qualified name (FQN) refers to the complete name of a class or interface, including the package name to which it belongs. A fully qualified name uniquely identifies a specific class or interface within the entire Java ecosystem.

```
package adminUpes;  
//import upes.Ifcm;  
  
class Admin{  
    public static void main(String args[]){  
        upes.Ifcm i = new upes.Ifcm();  
        i.ifmData();  
    }  
}
```

```
F:\Java Code>javac -d . Ifm.java  
  
F:\Java Code>javac -d . Admin.java  
  
F:\Java Code>java adminUpes.Admin  
Data of IFM Students
```

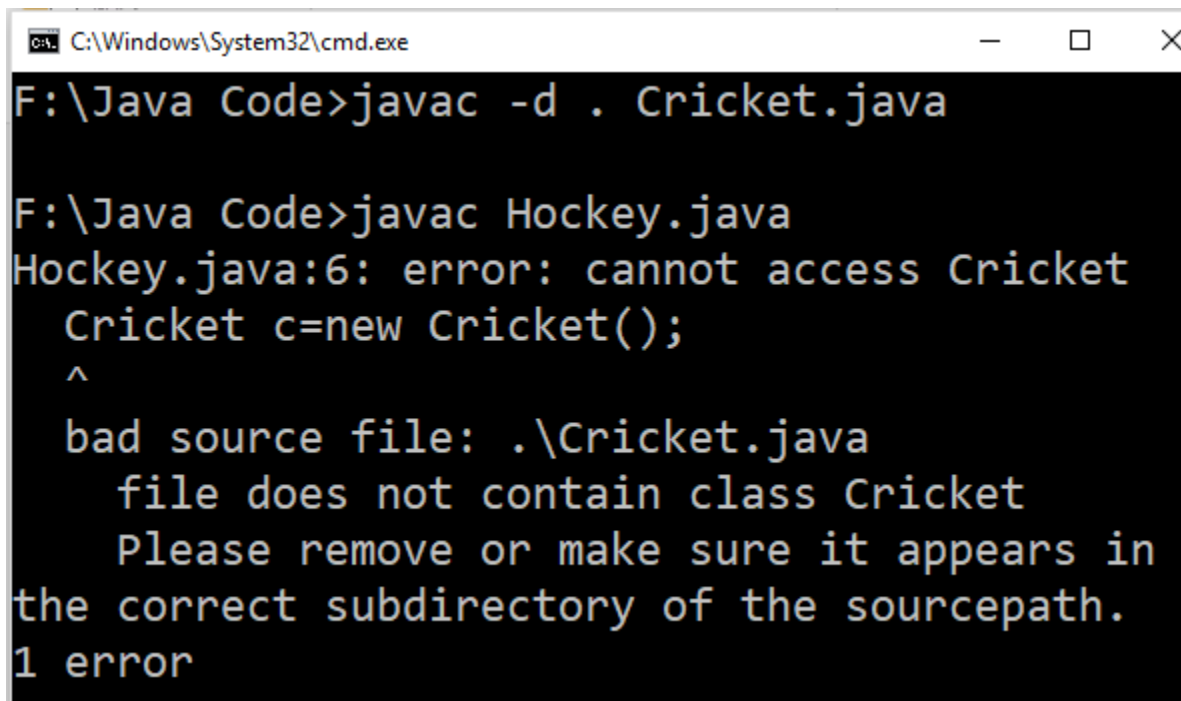
According to compiler version behavior, there are only two ways to access the package from outside the package. (class → package).

1. import package.*;

```
package Sport;  
public class Cricket{  
    String name="Subham";  
    public void run()  
    {  
        System.out.println(name+ " is a Cricket Player ");  
    }  
}
```

`import Sport.*;` //will give compile time error, this technique will work with predefined packages only. It may be a compiler issue/bug.

```
class Hockey{
    public static void main(String args[]){
        Cricket c=new Cricket();
        c.run();
    }
}
```

A screenshot of a Windows command prompt window. The title bar shows 'C:\Windows\System32\cmd.exe'. The command prompt shows the following sequence of commands and output:
F:\Java Code>javac -d . Cricket.java

F:\Java Code>javac Hockey.java
Hockey.java:6: error: cannot access Cricket
 Cricket c=new Cricket();
 ^
bad source file: .\Cricket.java
 file does not contain class Cricket
 Please remove or make sure it appears in
the correct subdirectory of the sourcepath.
1 error

2. `import package.classname;`

`import Sport.Cricket;`

```
class Hockey{
    public static void main(String args[]){
        Cricket c=new Cricket();
        c.run();
    }
}
```

```
F:\Java Code>javac -d . Cricket.java

F:\Java Code>javac Hockey.java

F:\Java Code>java Hockey
Subham is a Cricket Player
```

3. fully qualified name.

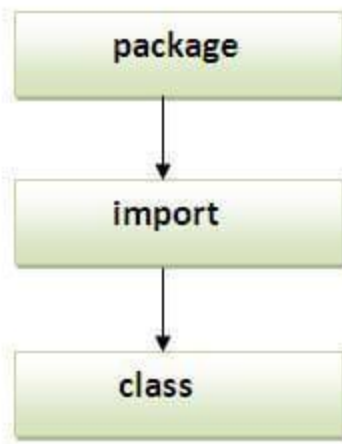
```
//import Sport.Cricket—no need to write here
class Hockey{
    public static void main(String args[]){
        Sport.Cricket c=new Sport.Cricket();
        c.run();
    }
}
```

```
F:\Java Code>javac -d . Cricket.java

F:\Java Code>javac Hockey.java

F:\Java Code>java Hockey
Subham is a Cricket Player
```

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



Subpackage in java

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

Let's take an example, Sun Microsystems has defined 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 sub-packages such as lang, net, io, etc., and put the Input/Output related classes in the io package, Server and ServerSocket classes in net packages, and so on.

package upes.socs.ifm;

```
class Data{  
    public static void main(String args[]){  
        System.out.println("data of subpackage");  
    }  
}
```

```
F:\Java Code>javac -d . Data.java  
  
F:\Java Code>java upes.socs.ifm.Data  
data of subpackage
```

This PC > UPES (F:) > Java Code > upes > socs > ifm			
	Name	Date modified	Type
is	Data.class	2/22/2019 12:13 PM	CLASS File

Note: In any java program there should be only at most 1 package statement. If we are taking more than one package statement we will get compile time error.

```
package pack1;  
package pack2;//error
```

```
class MyPackage1  
{  
  //  
  --statements—  
}
```

compile time error: class, interface, and enum expected.

Note: If any java program the first non-comment statement should be a package statement(if it is available).

```
import java.util.*;  
package pack1;//error  
class MyPackage2  
{  
  
}
```

Compile time error: class, interface, and enum expected.

Structure of Java source file:

