Java Access Control Mechanism - Controlling access to members of a class

Access level modifiers determine whether other classes can use a particular field or invoke a particular method. There are two levels of access control:

- At the top level—public, or *package-private* (no explicit modifier).
- At the member level—public, private, protected, or *package-private* (no explicit modifier).

A class may be declared with the modifier *public*, in which case that class is visible to all classes everywhere. If a class has *no modifier* (the default, also known as package-private), it is visible only within its own package.

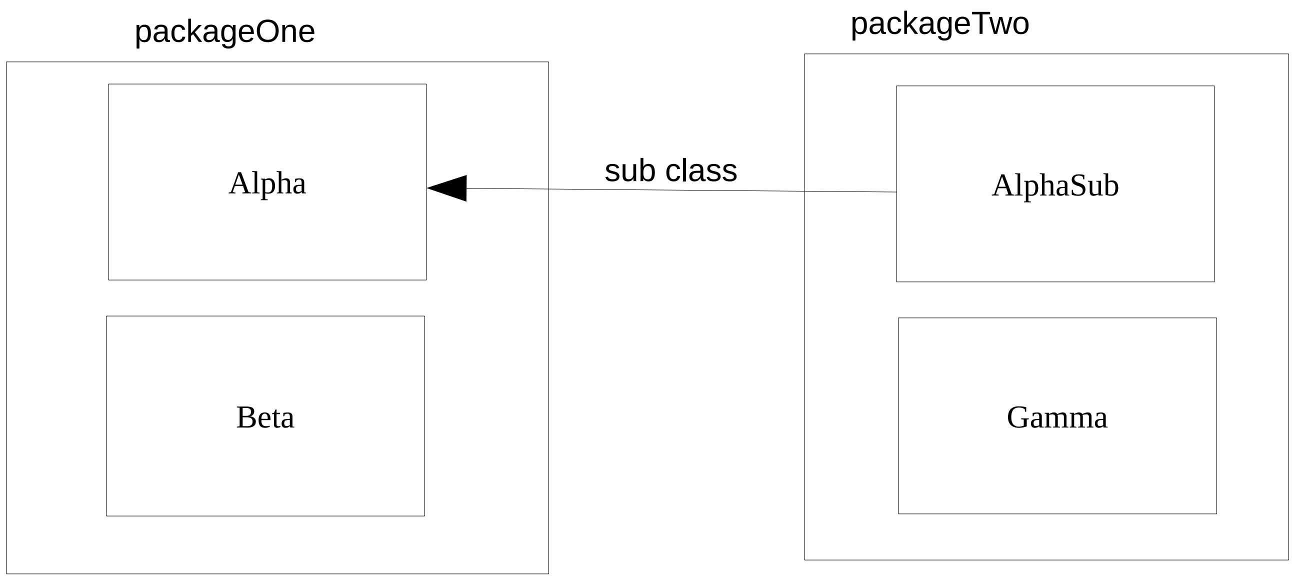
At the member level, you can also use the *public* modifier or *no modifier* (package-private) just as with top-level classes, and with the same meaning. For members, there are two additional access modifiers: *private* and *protected*. The private modifier specifies that the member can only be accessed in its own class. The protected modifier specifies that the member can only be accessed within its own package (as with package-private) and, in addition, by a subclass of its class in another package.

The following table shows the access to members permitted by each modifier.

Access Levels				
Modifier	**Class**	**Package**	**Subclass**	**World**
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

The first data column indicates whether the class itself has access to the member defined by the access level. As you can see, a class always has access to its own members. The second column indicates whether classes in the same package as the class (regardless of their parentage) have access to the member. The third column indicates whether subclasses of the class — declared outside this package — have access to the member. The fourth column indicates whether all classes have access to the member.

Access levels affect you in two ways. First, when you use classes that come from another source, such as the classes in the Java platform, access levels determine which members of those classes your own classes can use. Second, when you write a class, you need to decide what access level every member variable and every method in your class should have.



AlphaSub is a subclass of Alpha, but belongs to a package different from the one Alpha belongs to.

The following table shows where the members of the Alpha class are visible for each of the access modifiers that can be applied to them.

Visibility of members of Alpha class to others

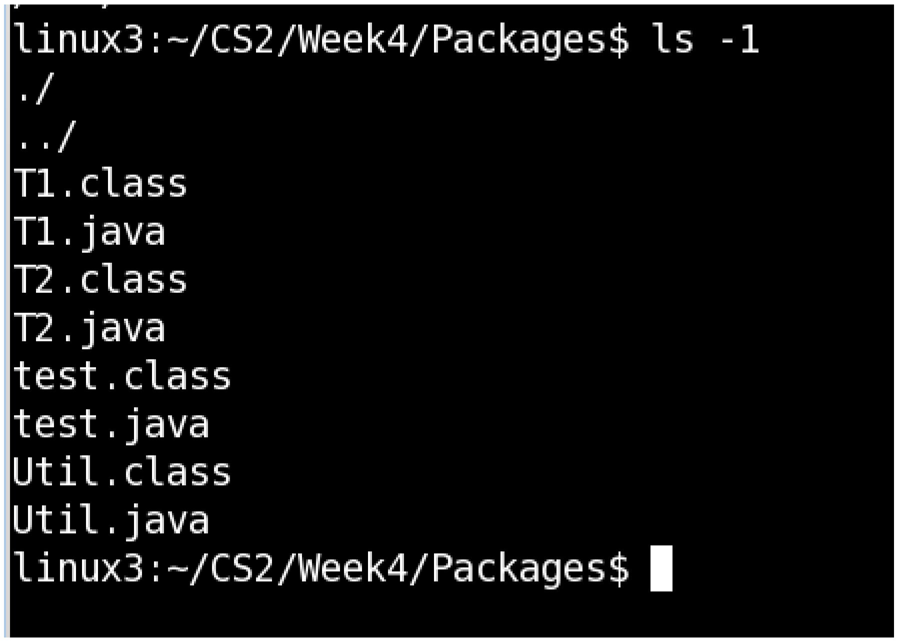
Modifier	Alpha	Beta	AlphaSub	Gamma
`public`	**Y**	Y	Y	Y
`protected`	**Y**	Y	Y	N
no modifier	**Y**	Y	N	N
`private`	**Y**	N	N	N

If a member of Alpha class has *no modifier* (package-private) specification, then members of the same package can see that member but a subclass of Alpha in a different package can not.

Java Packages

Suppose you have 4 Java source files Util.java, T1.java, T2.java and test.java. They are all sitting in the same directory and none of the source files have any "package" statements in them. So the classes belong to the *default package*.

Here is a directory listing of the current source and class files



```
linux3:~/CS2/Week4/Packages$ ls -1
./
../
T1.class
T1.java
T2.class
T2.java
test.class
test.java
Util.class
Util.java
linux3:~/CS2/Week4/Packages$
```

Each source file is shown below:

```
public class Util {
   public static void printMessage(String s){
        System.out.println(s);
   }
   public static void printDate(){
        Date d= new Date();
        System.out.println(d);
   }
}
-----------------------------------------------------
public class T1 {
   public void f(){
        Util.printDate();
        Util.printMessage("T1");
        myApp2.shared.Util.printMessage("from T1");
   }
   public static void main(String[] args){
        T1 t1=new T1();
        t1.f();
    }
}
------------------------------------------------------
public class T2 {
    public void f(){
        Util.printDate();
        Util.printMessage("T2");
    }
    public static void main(String[] args){
        T2 t2=new T2();
        t2.f();
    }
}
------------------------------------------------------
public class test{
    public static void main(String[] args){
        T1 t1=new T1();
        t1.f();
        T2 t2=new T2();
        t2.f();
    }
}
```

Suppose you decide to move the classes into separate **packages** as shown below. You are creating a package named myApp, and “subpackages” myApp.shared, myApp.test1 and myApp.test2 to place the different source files, as shown in the table below.

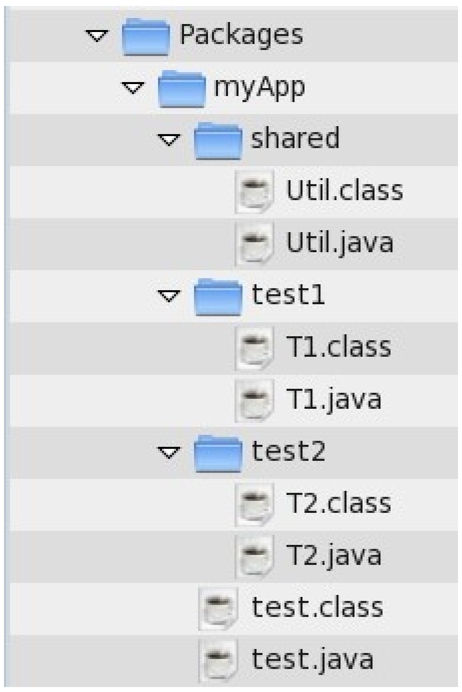
How will you modify the source files, and where will you place each of them? How will you compile and run the test class?

Destination Packages

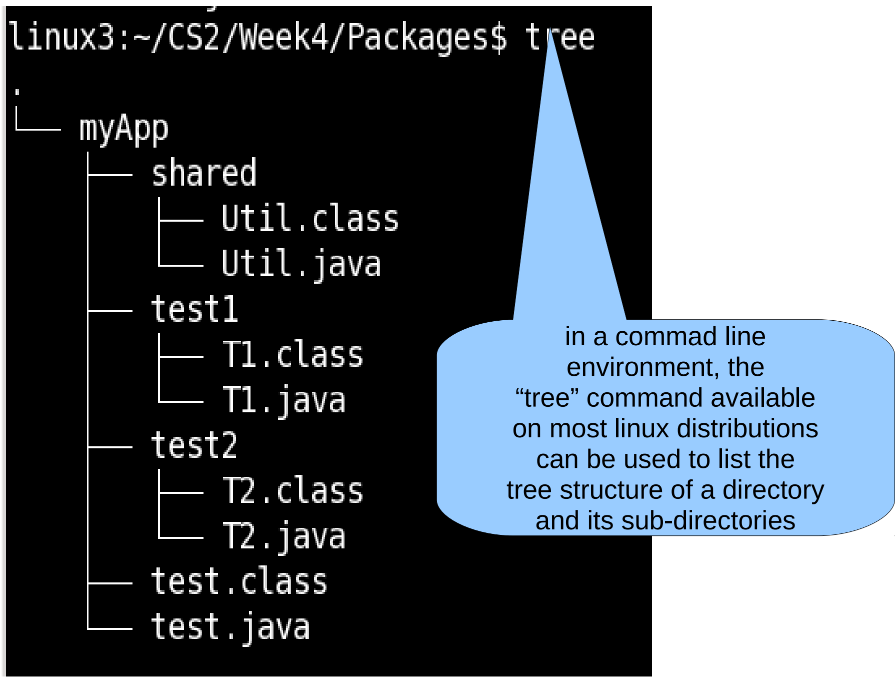
Package Name	**Class Name**
myApp.shared	Util
myApp.test1	T1
myApp.test2	T2
myApp	test

Util.java moved to myApp/shared/
T1.java moved to myApp/test1/
T2.java moved to myApp/test2/
test.java moved to myApp/

The new file locations are shown in the tree diagrams below. What we have done is to create a myApp package, along with myApp.shared, myApp.test1 and myApp.test2 packages.



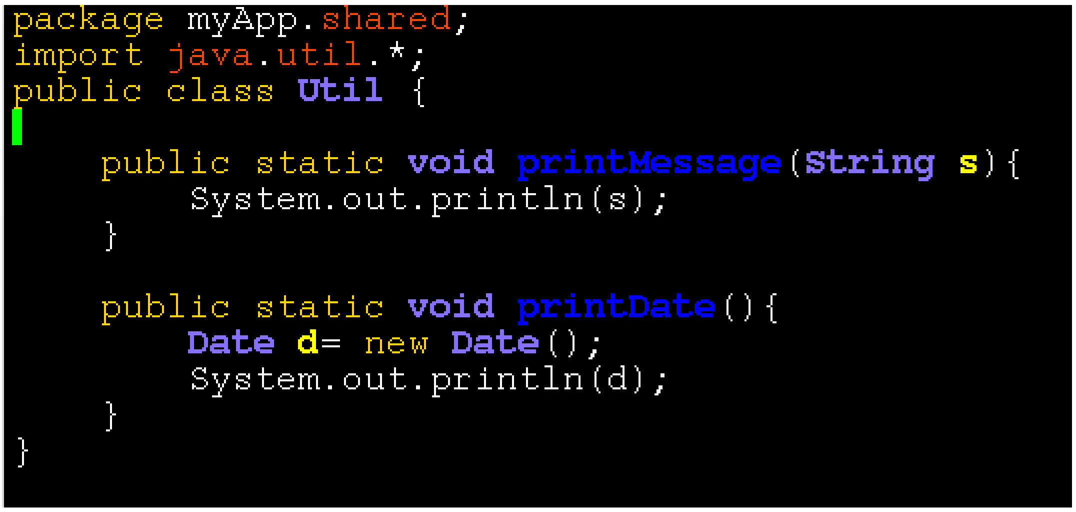
```
Packages
  myApp
    shared
      Util.class
      Util.java
    test1
      T1.class
      T1.java
    test2
      T2.class
      T2.java
    test.class
    test.java
```



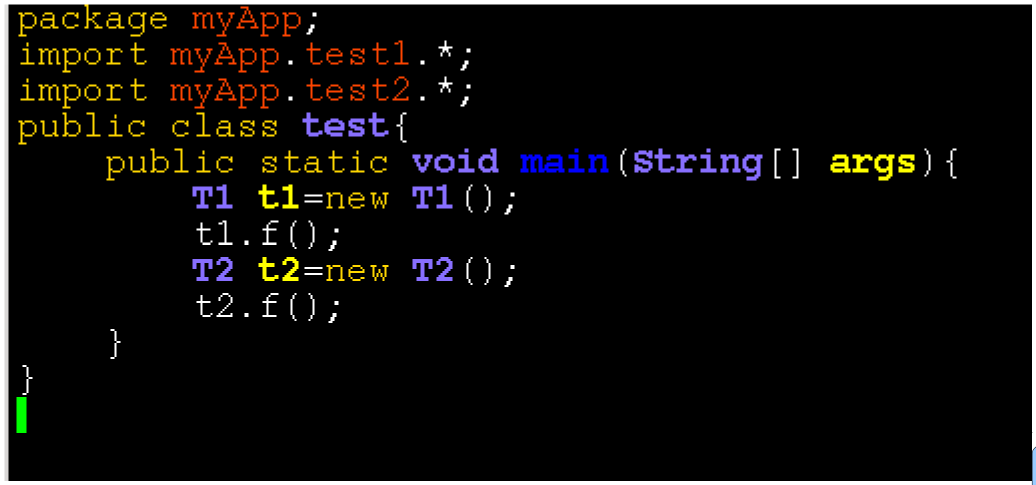
```
linux3:~/CS2/Week4/Packages$ tree
.
└── myApp
    ├── shared
    │   ├── Util.class
    │   └── Util.java
    ├── test1
    │   ├── T1.class
    │   └── T1.java
    ├── test2
    │   ├── T2.class
    │   └── T2.java
    ├── test.class
    └── test.java
```

in a commad line environment, the “tree” command available on most linux distributions can be used to list the tree structure of a directory and its sub-directories

Each source file need to have, as their first non-comment line, a *package* statement saying which package that source file belongs to. The modifications for the source files are shown below. The root of the package is ~/CS2/Week4/Packages. *test.java* is in ~/CS2/Week4/Packages/myApp directory and has *package myApp;* as its first line. T1.java sits in ~/CS2/Week4/Packages/myApp/test1/ and has *package myApp/test1;* as its first line. T2.java sits in ~/CS2/Week4/Packages/myApp/test2/ and has *package myApp/test2;* as its first line. Util.java sits in ~/CS2/Week4/Packages/myApp/shared/ and has *package myApp/shared;* as its first line.

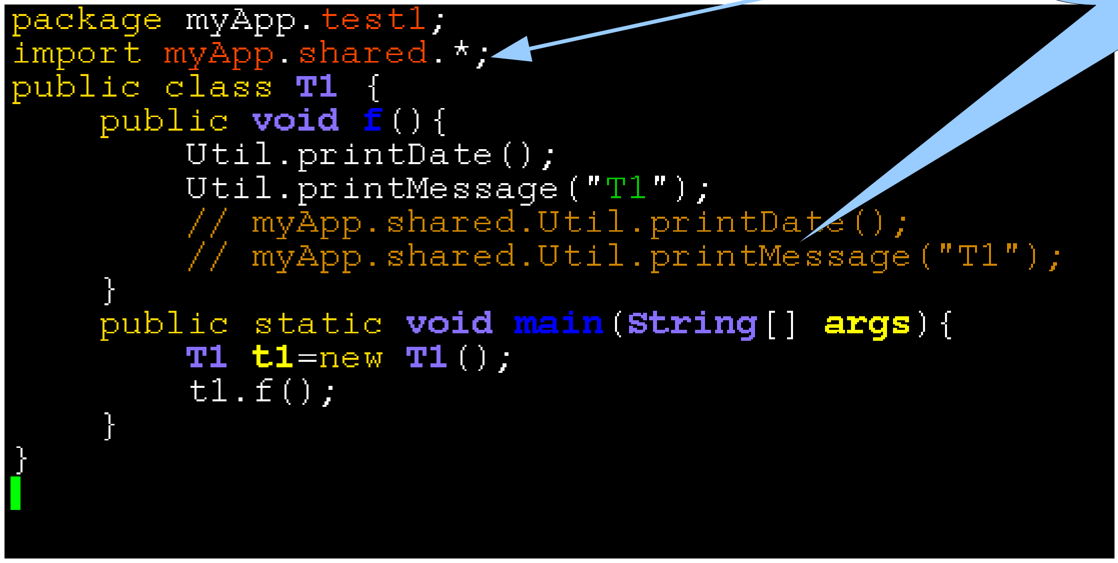


```java
package myApp.shared;
import java.util.*;
public class Util {

    public static void printMessage(String s){
        System.out.println(s);
    }

    public static void printDate(){
        Date d= new Date();
        System.out.println(d);
    }
}
```

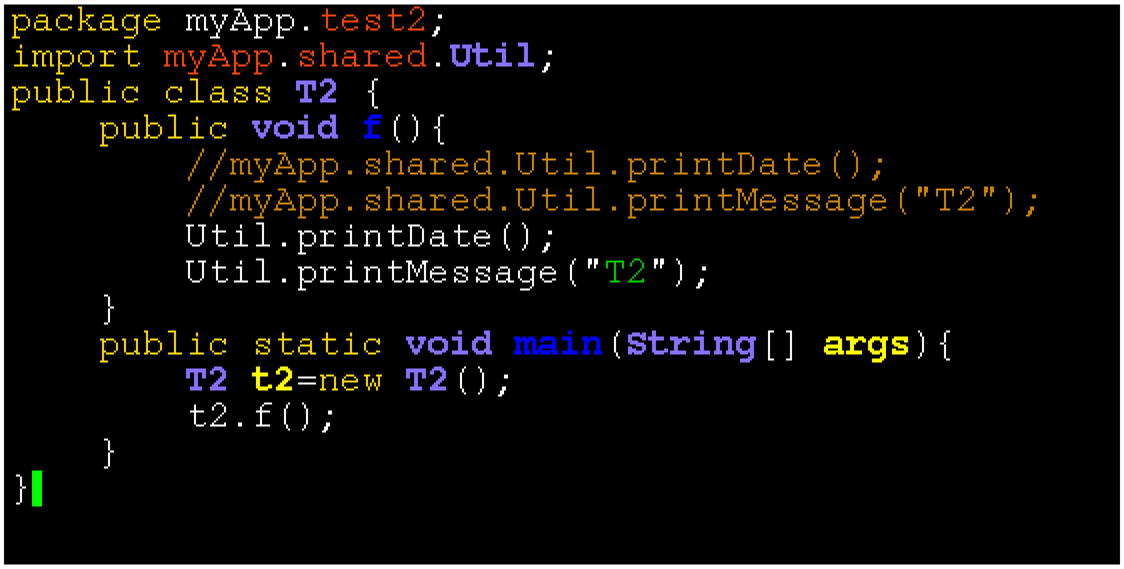


```java
package myApp;
import myApp.test1.*;
import myApp.test2.*;
public class test{
    public static void main(String[] args){
        T1 t1=new T1();
        t1.f();
        T2 t2=new T2();
        t2.f();
    }
}
```

If you don't have the import statement,
these will work



```java
package myApp.test1;
import myApp.shared.*;
public class T1 {
    public void f(){
        Util.printDate();
        Util.printMessage("T1");
        // myApp.shared.Util.printDate();
        // myApp.shared.Util.printMessage("T1");
    }
    public static void main(String[] args){
        T1 t1=new T1();
        t1.f();
    }
}
```



```
package myApp.test2;
import myApp.shared.Util;
public class T2 {
    public void f(){
        //myApp.shared.Util.printDate();
        //myApp.shared.Util.printMessage("T2");
        Util.printDate();
        Util.printMessage("T2");
    }
    public static void main(String[] args){
        T2 t2=new T2();
        t2.f();
    }
}
```

To compile the java source files in myApp package, you need to execute "javac" from myApp's parent directory:

```
linux3:~/CS2/Week4/Packages$ javac myApp/*.java
```

The shortcut '*.java' compiles all java source files in myApp/ along with any other source files necessary in other sub-directories. Instead of using this shortcut you could compile each source file separately. The following screenshot shows both processes. To run the test.class compiled program,

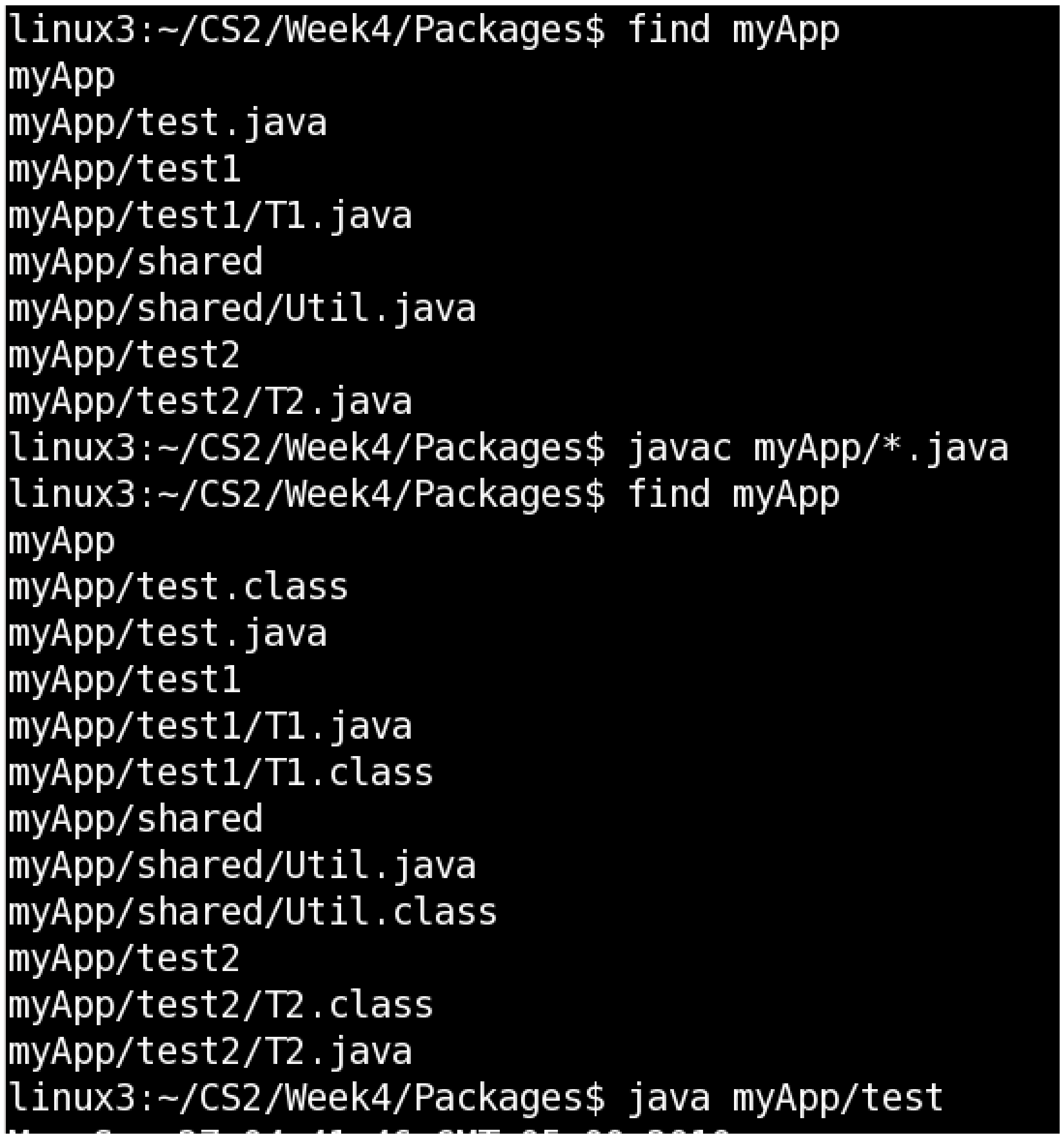
```
linux3:~/CS2/Week4/Packages$ java myApp/test
```

It is also important to note that a statement such as

import myApp.;*

does **not** import classes from myApp/shared/. You explicitly need to say:

import myApp/shared/;*

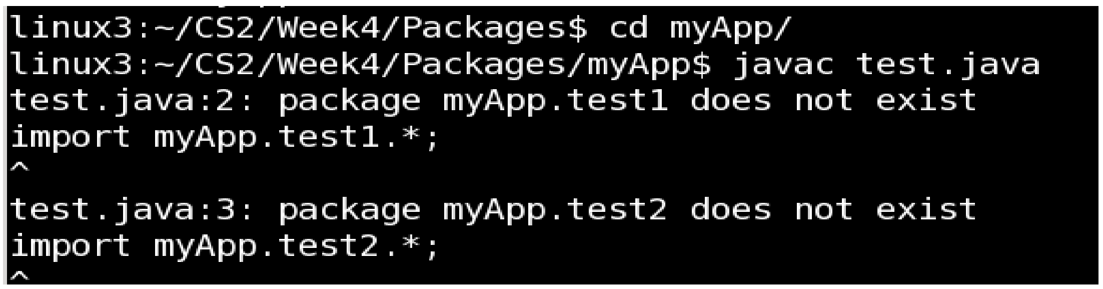


```
linux3:~/CS2/Week4/Packages$ find myApp
myApp
myApp/test.java
myApp/test1
myApp/test1/T1.java
myApp/shared
myApp/shared/Util.java
myApp/test2
myApp/test2/T2.java
linux3:~/CS2/Week4/Packages$ javac myApp/*.java
linux3:~/CS2/Week4/Packages$ find myApp
myApp
myApp/test.class
myApp/test.java
myApp/test1
myApp/test1/T1.java
myApp/test1/T1.class
myApp/shared
myApp/shared/Util.java
myApp/shared/Util.class
myApp/test2
myApp/test2/T2.class
myApp/test2/T2.java
linux3:~/CS2/Week4/Packages$ java myApp/test
```

It is important to note that

```
linux3:~/CS2/Week4/Packages/myApp$javac *.java
```

will not work:



```
linux3:~/CS2/Week4/Packages$ cd myApp/
linux3:~/CS2/Week4/Packages/myApp$ javac test.java
test.java:2: package myApp.test1 does not exist
import myApp.test1.*;
^
test.java:3: package myApp.test2 does not exist
import myApp.test2.*;
^
```