1 Time Class Case Study: Creating Packages (Cont.)

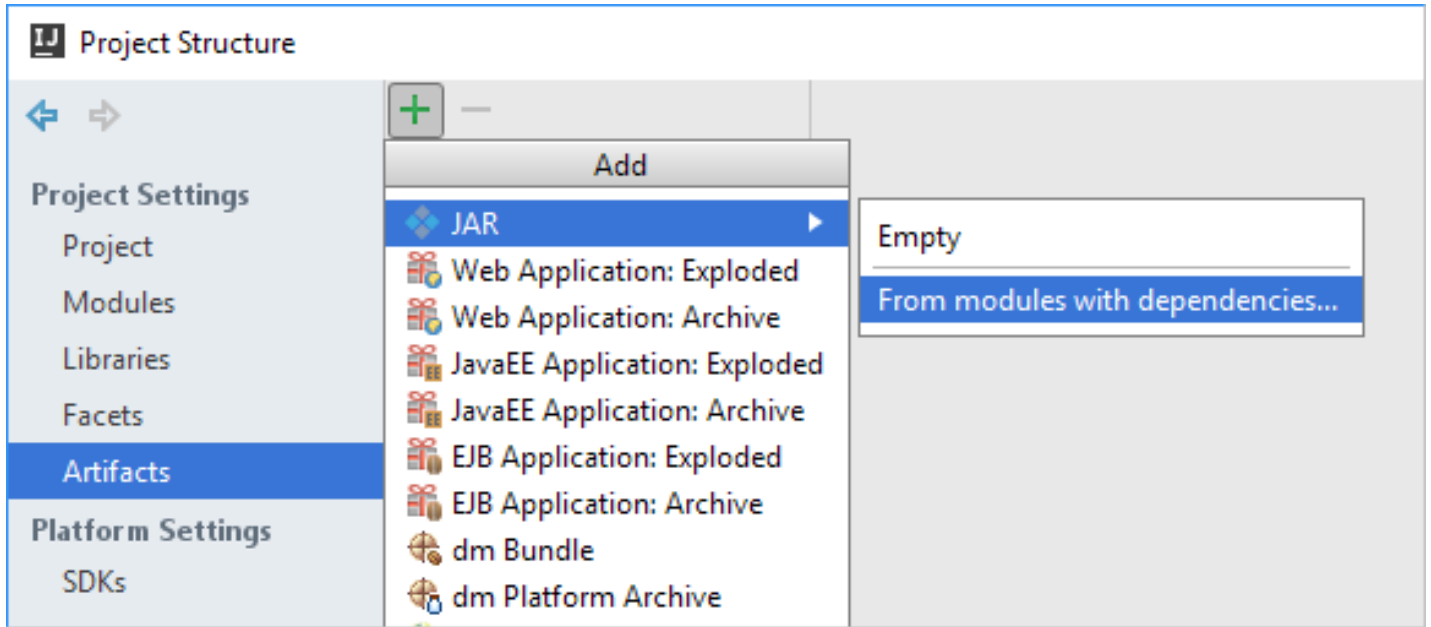
Steps for creating Jars in IntelliJ IDEA:

1.Select File | Project Structure to open the Project Structure dialog.
Assume the Project (IntelliJ Module) name is `TimePackage`

2.Under Project Settings, select `Artifacts`.

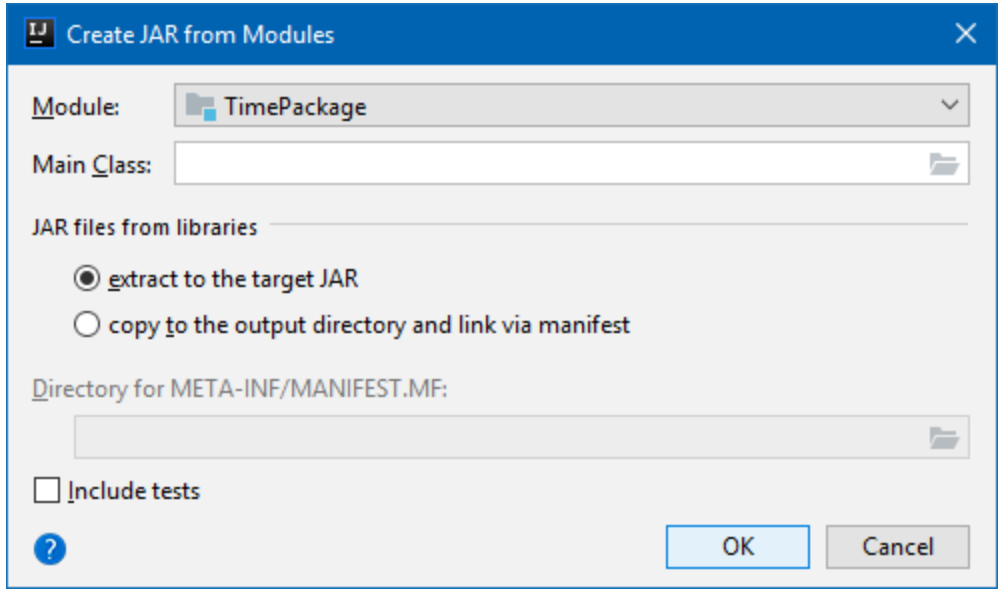
3.Click +, point to `JAR` and select

`From modules with dependencies.`



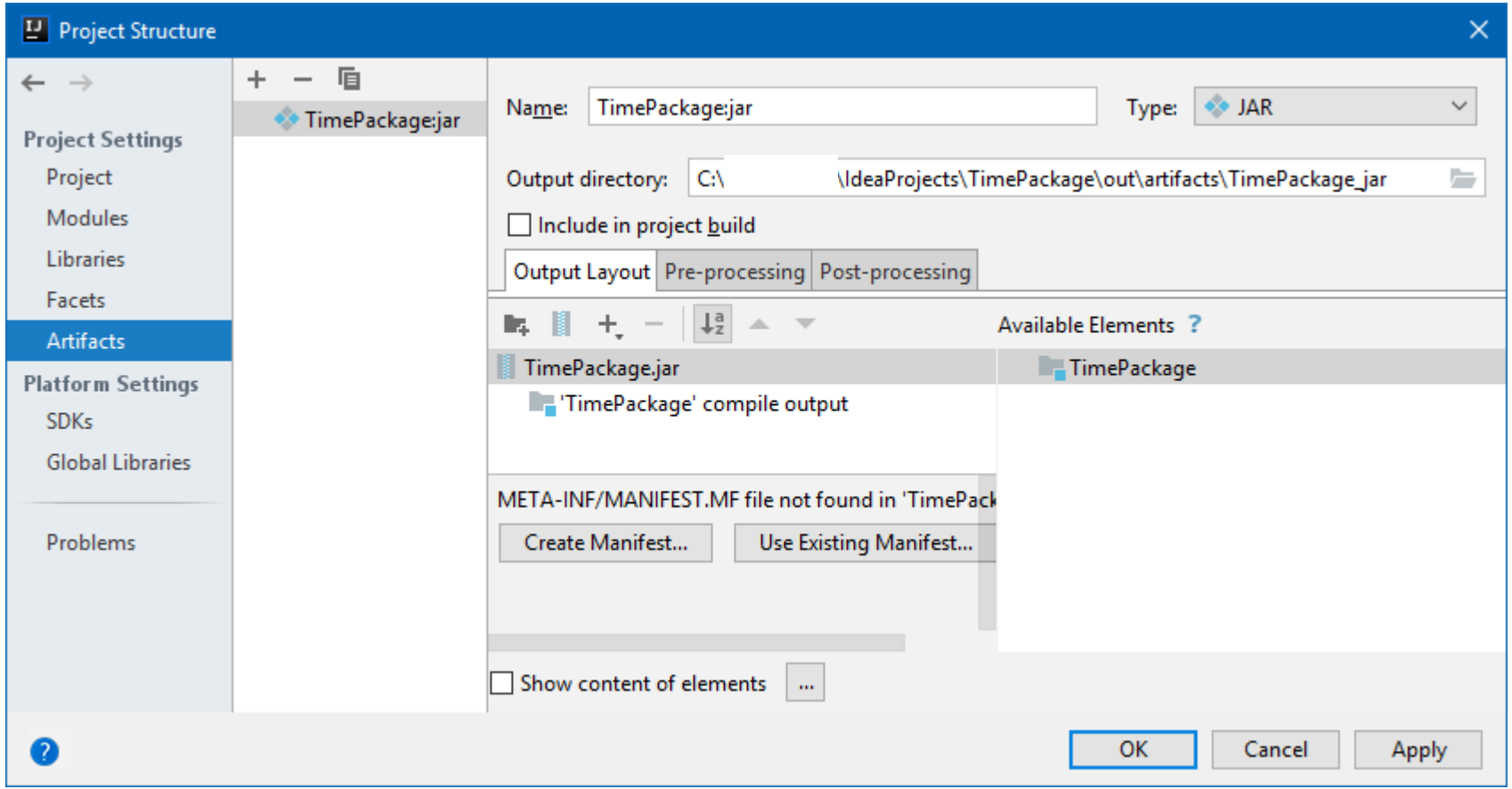
1 Time Class Case Study: Creating Packages (Cont.)

4. **In the dialog that opens, specify the Module . Eventually specify also the Main class in case you intend to build an executable Jar. (***To the right of the Main Class field, click the browse Button and select the Main class in the dialog that opens***).**



1 Time Class Case Study: Creating Packages (Cont.)

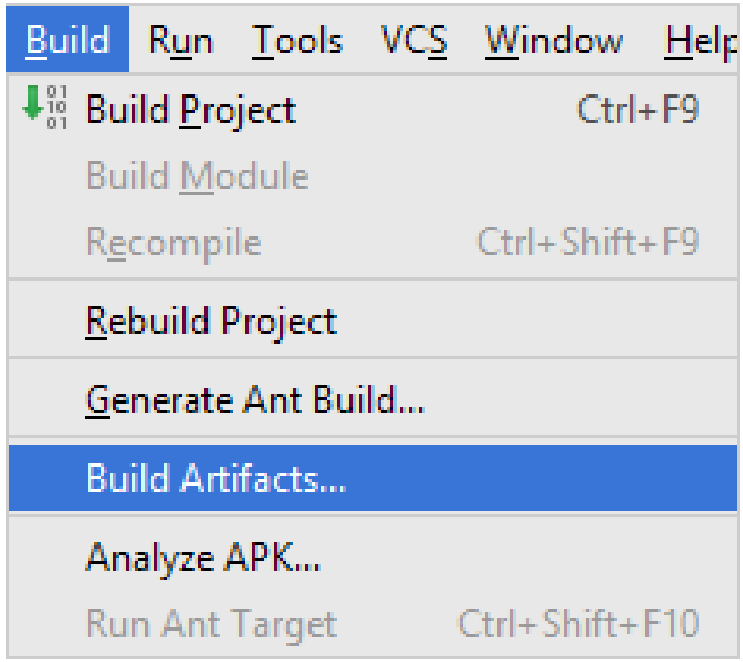
5. As a result, the artifact configuration is created, and its settings are shown in the right-hand part of the **Project Structure** dialog. **Click OK**



1 Time Class Case Study: Creating Packages (Cont.)

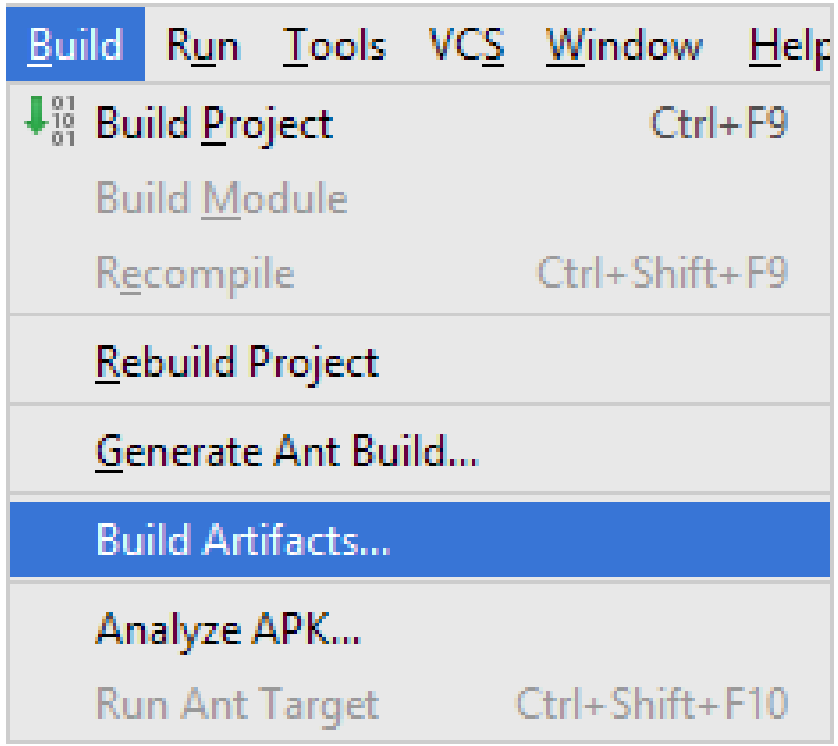
Building the JAR artifact

1. Select **Build | Build Artifacts**.



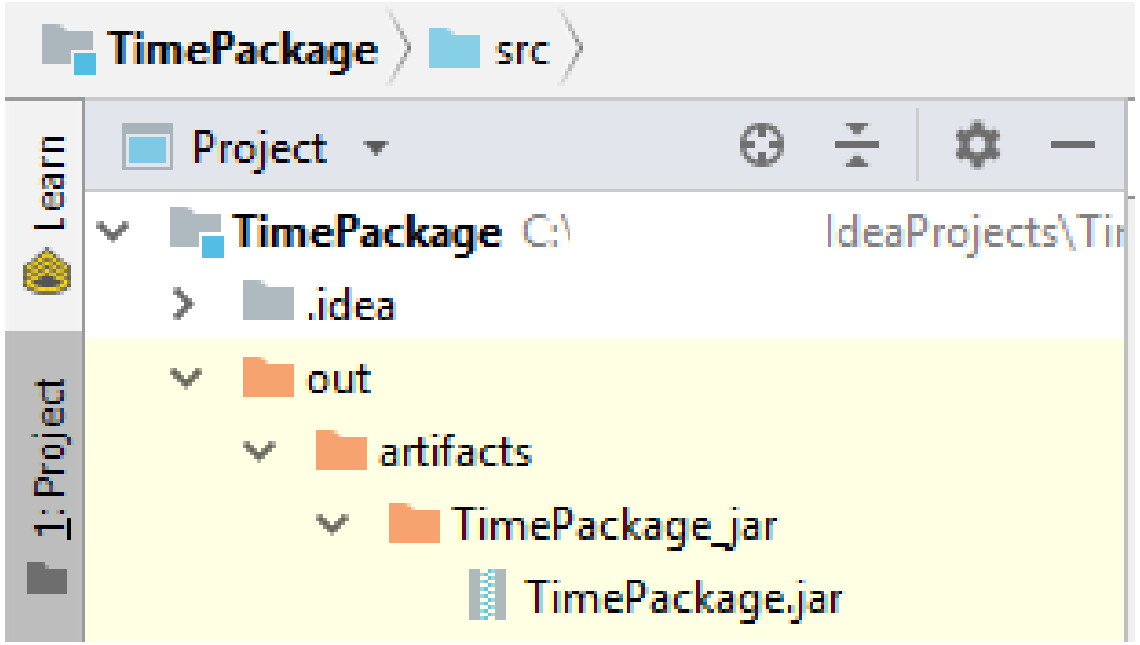
1 Time Class Case Study: Creating Packages (Cont.)

2. Point to `TimePackage:jar` and select `Build`. (*In this particular case, `Build` is the default action, so you can just press `Enter` instead.*)



1 Time Class Case Study: Creating Packages (Cont.)

If you now look at the `out/artifacts` folder, you'll find your `JAR` there.

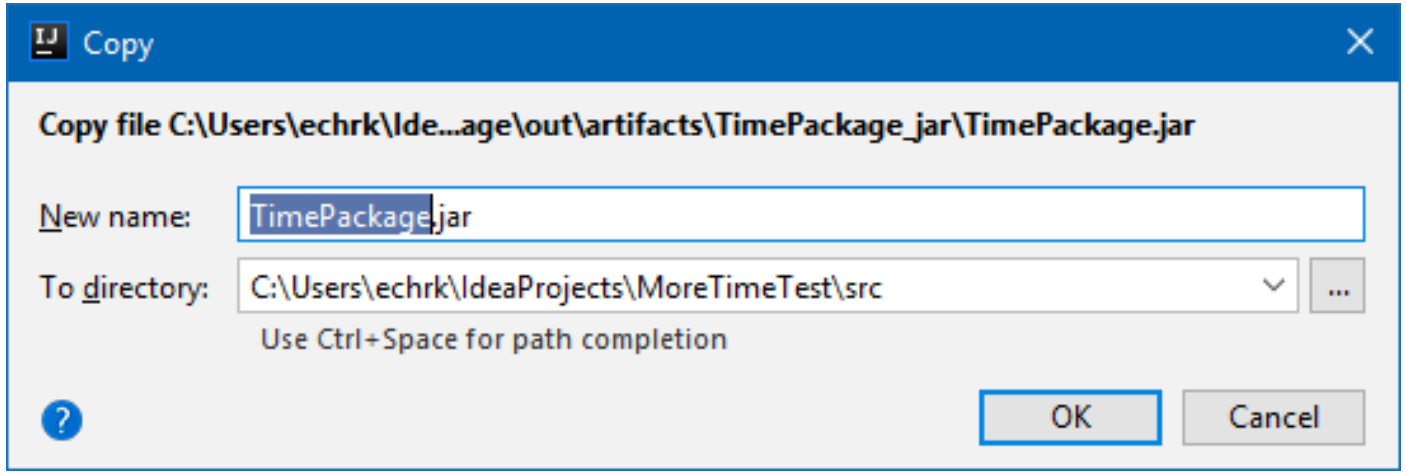


1 Time Class Case Study: Creating Packages (Cont.)

Steps for adding external jars in Project named `Time1Test` in IntelliJ IDEA:

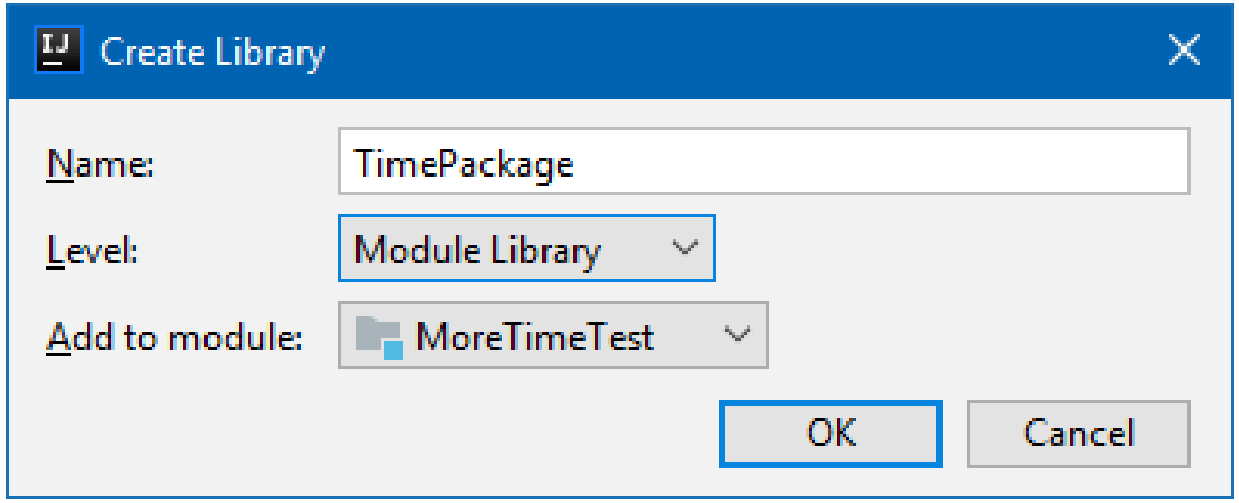
The easiest way.

1. Copy and paste `TimePackage.jar` in the `src` folder of the `Time1Test` project



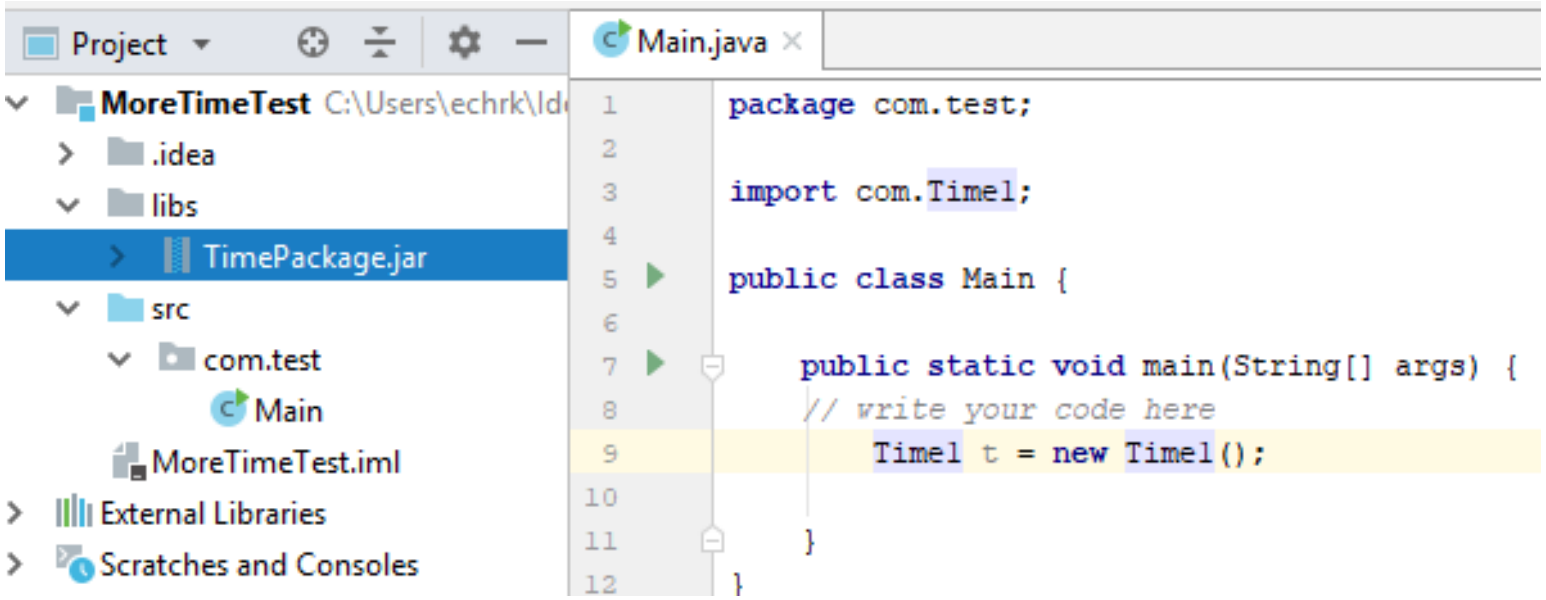
1 Time Class Case Study: Creating Packages (Cont.)

2. Right click `TimePackage.jar and select` **`Add as Library > Module Library`** in the `Time1Test` project



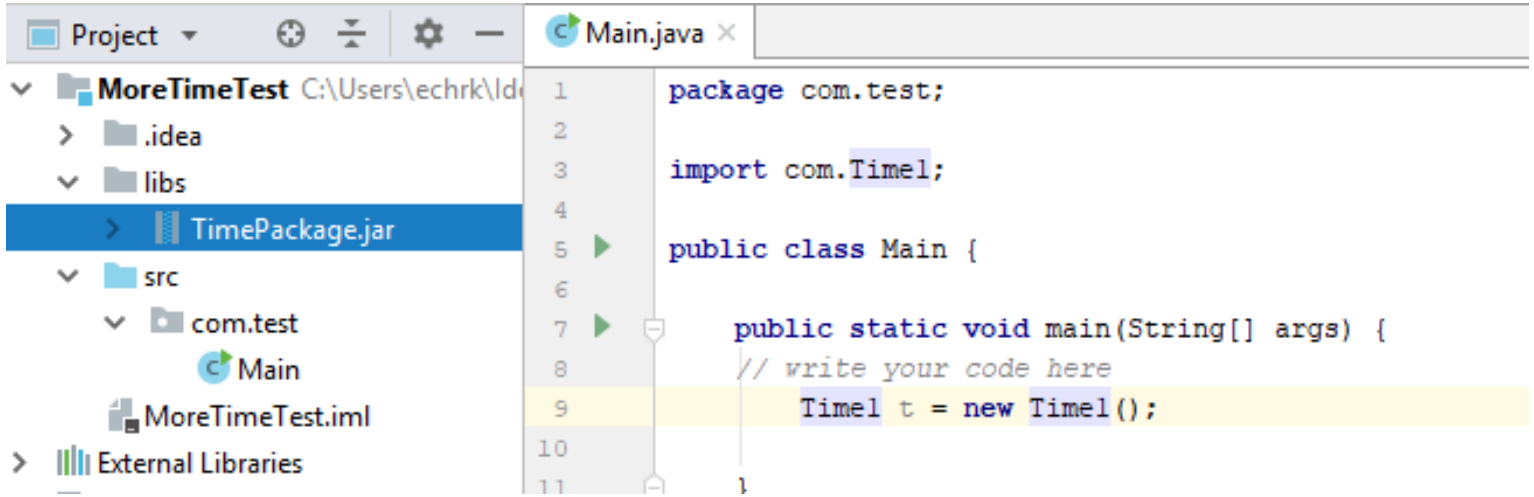
1 Time Class Case Study: Creating Packages (Cont.)

A better approach is to create \`libs` folder of the `Time1Test` project for storing inside a dedicated folder all your JAR files



1 Time Class Case Study: Creating Packages (Cont.)

The `Time1` class is now available for importing (Alt->Enter) . in the `Time1Test` project (Execute the previously described steps 1-2 for the case when the `TimePackage.jar` is copied in the `\libs` folder of the `Time1Test` project)



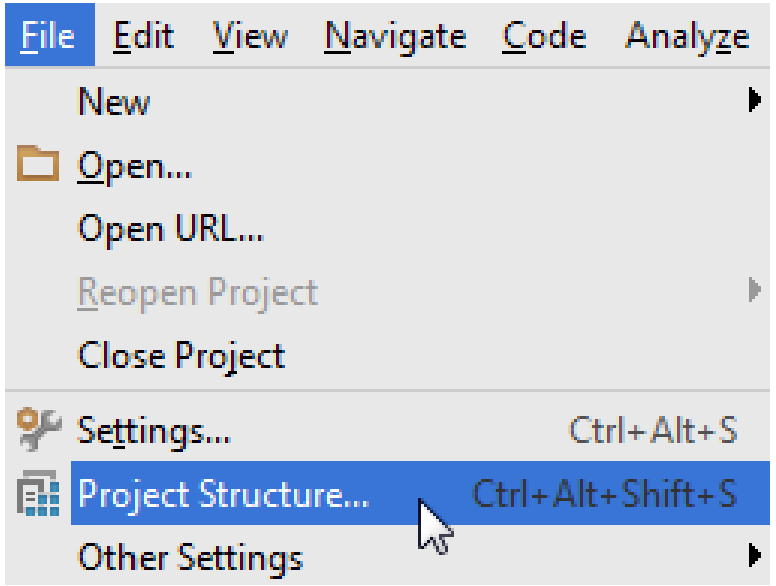
1 Time Class Case Study: Creating Packages (Cont.)

Steps for adding external jars in Project named `Time1Test` in IntelliJ IDEA:

1. Click `File` from the toolbar
2. Select `Project Structure`
   (`CTRL + SHIFT + ALT + S` on Windows/Linux,
   ⌘ + ; on Mac OS X)
3. Select `Modules` at the left panel
4. Select `the Dependencies` tab
5. Select '+' → JARs or directories

1 Time Class Case Study: Creating Packages (Cont.)

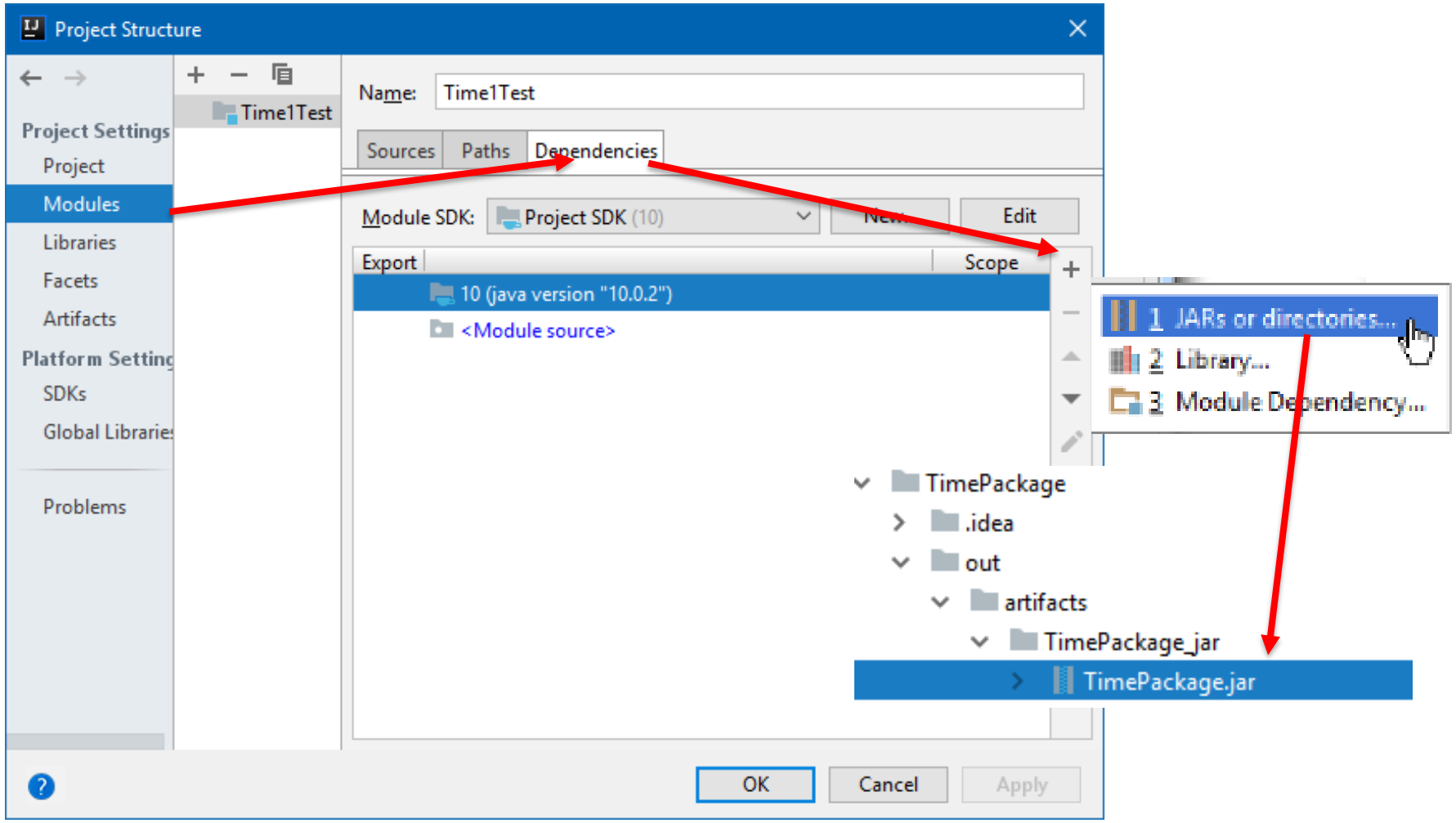
1. **File > Project Structure...**



or press Ctrl + Alt + Shift + S

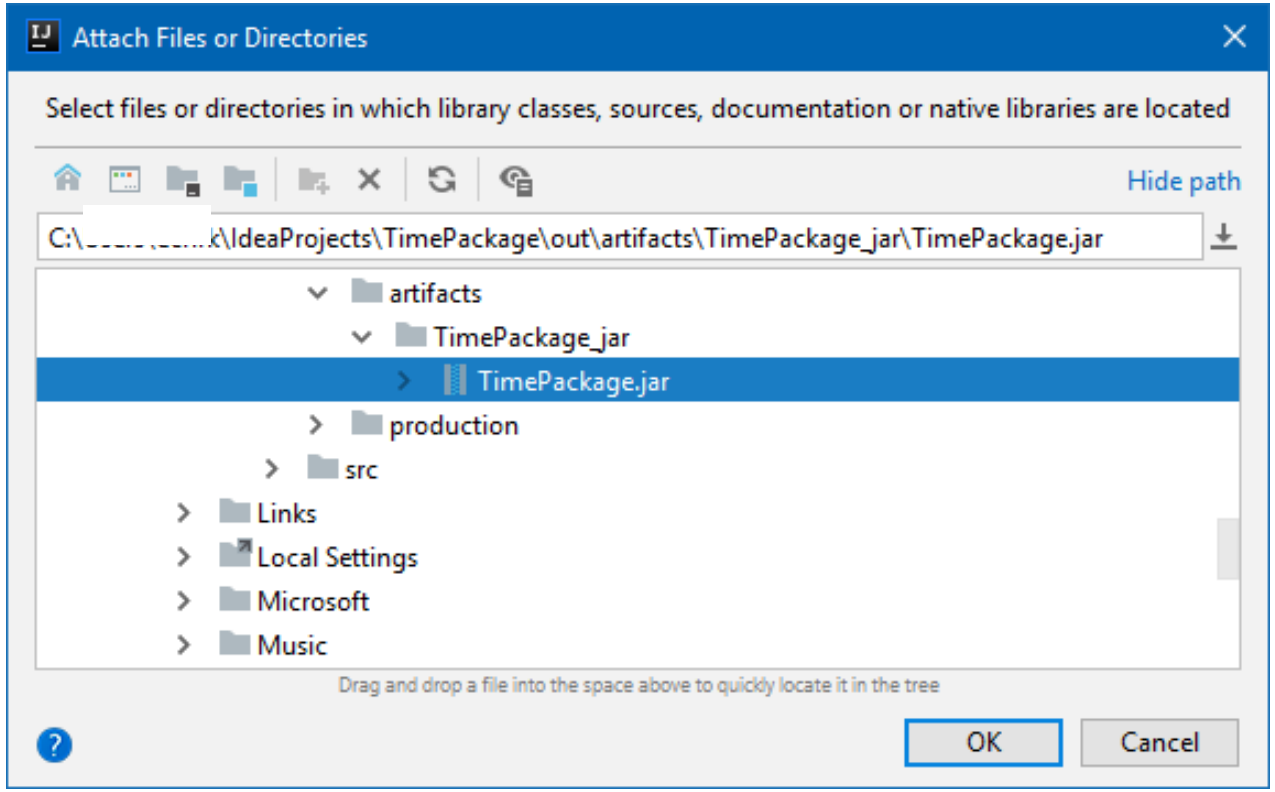
1 Time Class Case Study: Creating Packages (Cont.)

2. Project Settings > **Modules** > **Dependencies** > "**+**" sign > **JARs or directories... (*Select the previously created* JAR)**



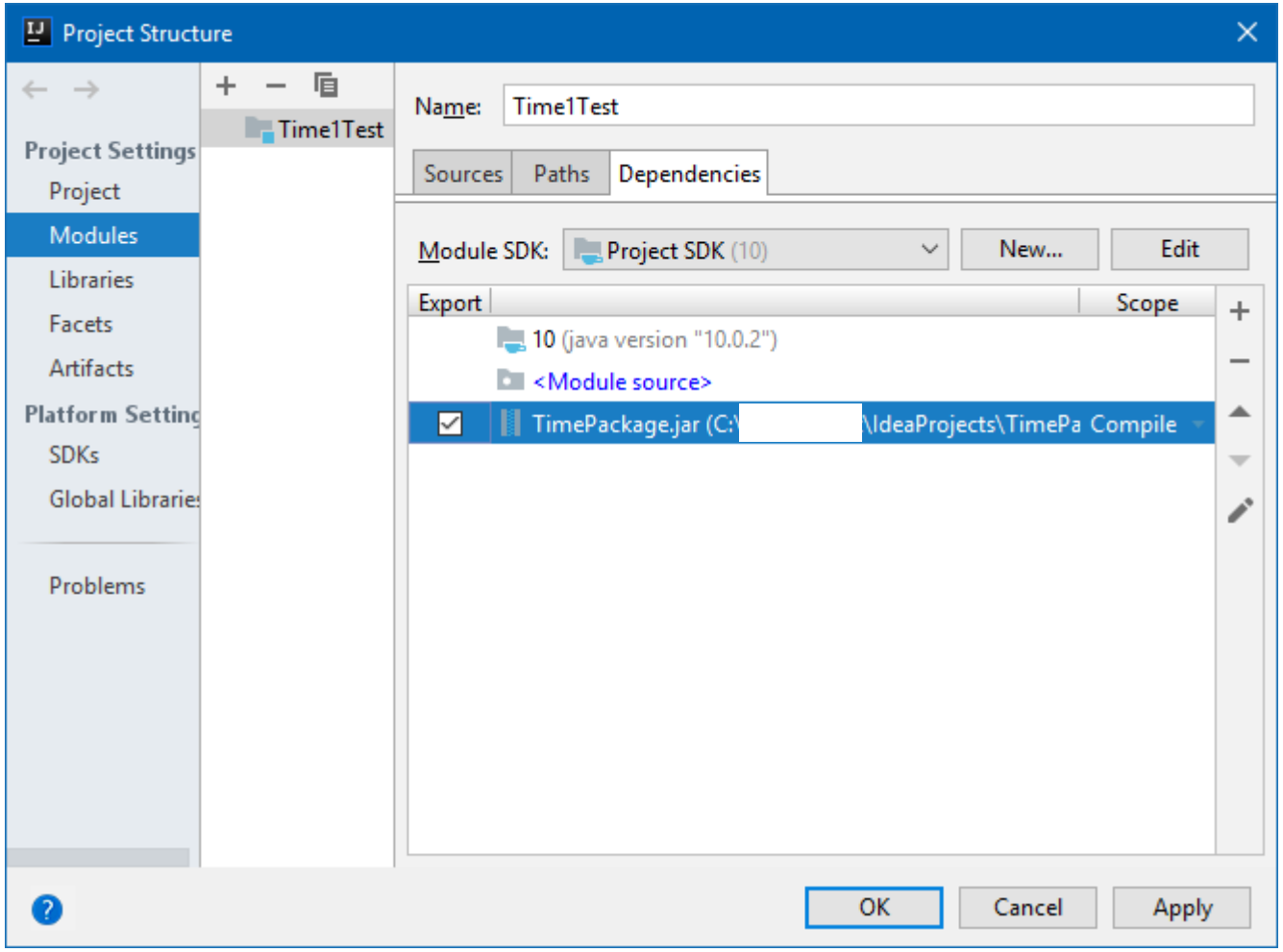
1 Time Class Case Study: Creating Packages (Cont.)

3. Select the JAR file and click on OK, then click on another OK button to confirm



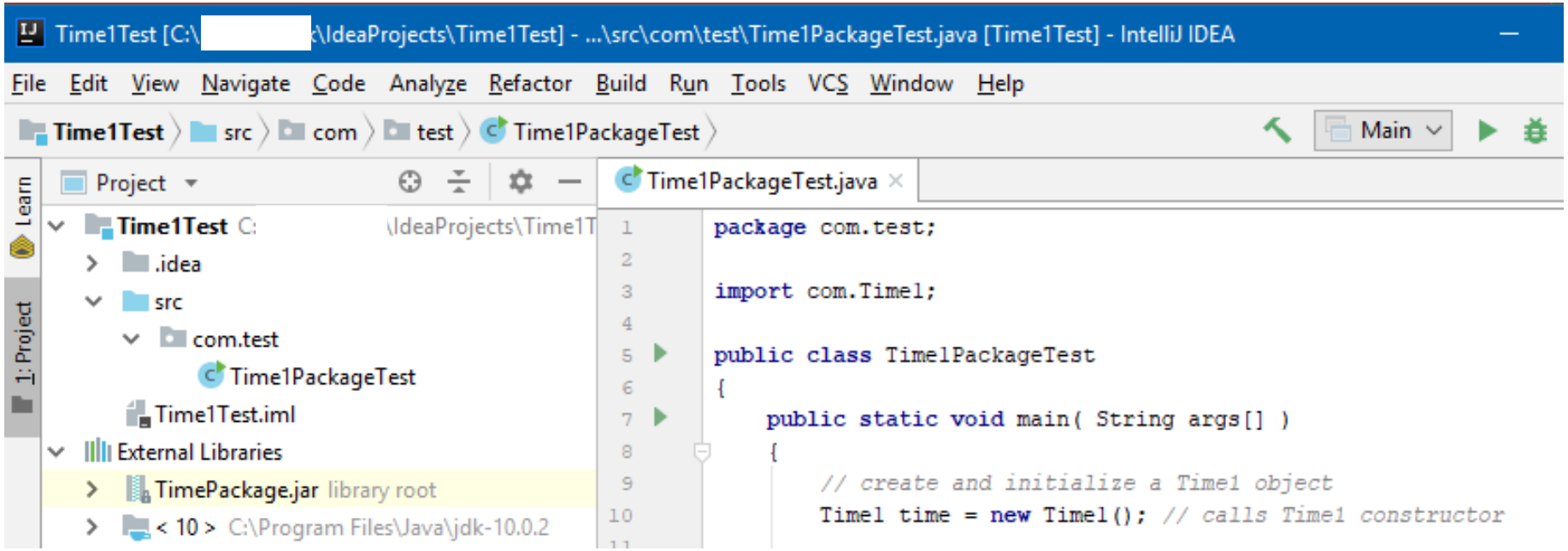
1 Time Class Case Study: Creating Packages (Cont.)

Select the JAR and click OK (Apply)



1 Time Class Case Study: Creating Packages (Cont.)

4. You can view the jar file in the "External Libraries" folder



Outline

```java
// Fig. 8.19: Time1PackageTest.java
// Time1 object used in an application.
import com.Time1; // import class Time1

public class Time1PackageTest
{
   public static void main( String args[] )
   {
      // create and initialize a Time1 object
      Time1 time = new Time1(); // calls Time1 constructor

      // output string representations of the time
      System.out.print( "The initial universal time is: " );
      System.out.println( time.toUniversalString() );
      System.out.print( "The initial standard time is: " );
      System.out.println( time.toString() );
      System.out.println(); // output a blank line

```

Single-type **`import`** declaration

Refer to the **`Time1`** class by its simple name

`Time1PackageTest`

`.java`

(1 of 2)

```java
      // change time and output updated time
      time.setTime( 13, 27, 6 );
      System.out.print( "Universal time after setTime is: " );
      System.out.println( time.toUniversalString() );
      System.out.print( "Standard time after setTime is: " );
      System.out.println( time.toString() );
      System.out.println(); // output a blank line

      // set time with invalid values; output updated time
      time.setTime( 99, 99, 99 );
      System.out.println( "After attempting invalid settings:" );
      System.out.print( "Universal time: " );
      System.out.println( time.toUniversalString() );
      System.out.print( "Standard time: " );
      System.out.println( time.toString() );
   } // end main
} // end class Time1PackageTest
```

```
The initial universal time is: 00:00:00
The initial standard time is: 12:00:00 AM

Universal time after setTime is: 13:27:06
Standard time after setTime is: 1:27:06 PM

After attempting invalid settings:
Universal time: 00:00:00
Standard time: 12:00:00 AM
```

Outline

Time1PackageTest.java

(2 of 2)

1 Time Class Case Study: Creating Packages (Cont.)

Class loader

- Locates classes that the compiler needs
  - First searches standard Java classes bundled with the JDK
  - Then searches for optional packages
    - These are enabled by Java's extension mechanism
  - Finally searches the classpath
    - List of directories or archive files separated by directory separators
      - These files normally end with `.jar` or `.zip`
      - Standard classes are in the archive file `rt.jar`

1 Time Class Case Study: Creating Packages (Cont.)

- To use a classpath other than the current directory
  - `-classpath` option for the `javac` compiler
  - Set the `CLASSPATH` environment variable
- The JVM must locate classes just as the compiler does
  - The `java` command can use other classpathes by using the same techniques that the `javac` command uses

Common Programming Error 5a.13

Specifying an explicit classpath eliminates the current directory from the classpath. This prevents classes in the current directory (including packages in the current directory) from loading properly. If classes must be loaded from the current directory, include a dot (.) in the classpath to specify the current directory.

Software Engineering Observation 1

In general, it is a better practice to use the `-classpath` option of the compiler, rather than the `CLASSPATH` environment variable, to specify the classpath for a program. This enables each application to have its own classpath.

Error-Prevention Tip 5a.3

Specifying the classpath with the `CLASSPATH` environment variable can cause subtle and difficult-to-locate errors in programs that use different versions of the same package.

2 Package Access

Package access

- Methods and variables declared without any access modifier are given package access
- This has no effect if the program consists of one class
- This does have an effect if the program contains multiple classes from the same package
  - Package-access members can be directly accessed through the appropriate references to objects in other classes belonging to the same package

```java
// Fig. 8.20: PackageDataTest.java
// Package-access members of a class are accessible by other classes
// in the same package.

public class PackageDataTest
{
   public static void main( String args[] )
   {
      PackageData packageData = new PackageData();

      // output String representation of packageData
      System.out.printf( "After instantiation:\n%s\n", packageData );

      // change package access data in packageData object
      packageData.number = 77;
      packageData.string = "Goodbye";

      // output String representation of packageData
      System.out.printf( "\nAfter changing values:\n%s\n", packageData );
   } // end main
} // end class PackageDataTest
```

Can directly access package-access members

Outline

PackageDataTest.java

(1 of 2)

```java
// class with package access instance variables
class PackageData
{
   int number; // package-access instance variable
   String string; // package-access instance variable

   // constructor
   public PackageData()
   {
      number = 0;
      string = "Hello";
   } // end PackageData constructor

   // return PackageData object String representation
   public String toString()
   {
      return String.format( "number: %d; string: %s", number, string );
   } // end method toString
} // end class PackageData
```

Package-access instance variables

```
After instantiation:
number: 0; string: Hello

After changing values:
number: 77; string: Goodbye
```

Outline

PackageDataTest.java

(2 of 2)