Unit-1 Object Oriented Programming with Java

Java Package

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.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

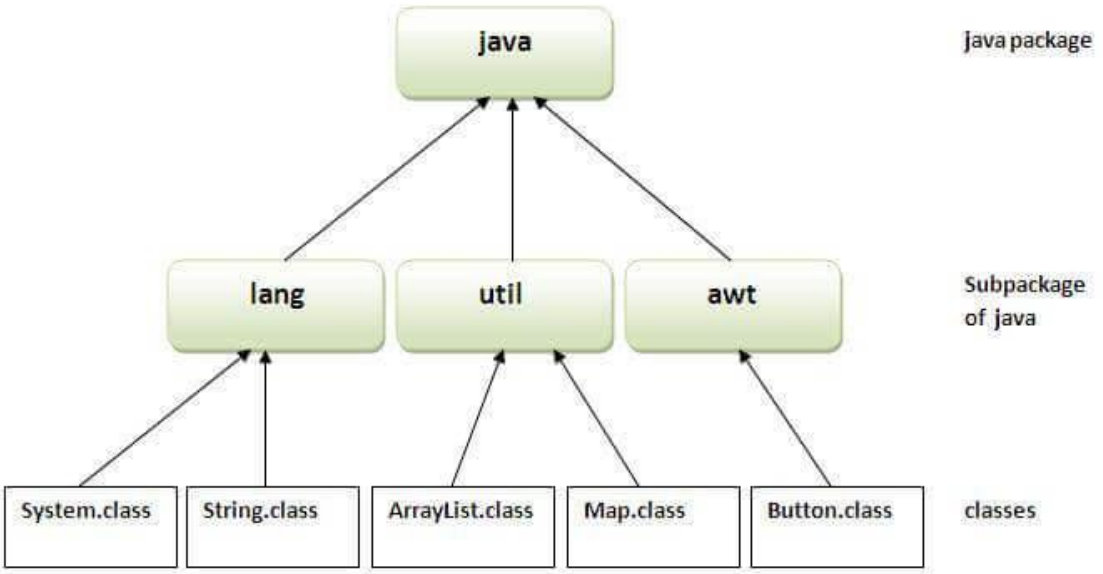
Here, we will have the detailed learning of creating and using user-defined packages.

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.



Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

Unit-1 Object Oriented Programming with Java

Simple example of java package

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

```
1. //save as Simple.java
2. package mypack;
3. public class Simple{
4.  public static void main(String args[]){
5.     System.out.println("Welcome to package");
6.    }
7. }
```

How to compile java package

If you are not using any IDE, you need to follow the **syntax** given below:

```
1. javac -d directory javafilename
```

For **example**

```
1. javac -d . Simple.java
```

The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

---

*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 of package that import the packagename.*

```
1. //save by A.java
2. package pack;
3. public class A{
```


Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

```
4.   public void msg(){System.out.println("Hello");}
5.  }
1.  //save by B.java
2.  package mypack;
3.  import pack.*;
4.
5.  class B{
6.   public static void main(String args[]){
7.    A obj = new A();
8.    obj.msg();
9.   }
10. }
```

```
Output:Hello
```

2) Using packagename.classname

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

Example of package by import package.classname

```
1.  //save by A.java
2.
3.  package pack;
4.  public class A{
5.   public void msg(){System.out.println("Hello");}
6.  }
1.  //save by B.java
2.  package mypack;
3.  import pack.A;
4.
5.  class B{
6.   public static void main(String args[]){
7.    A obj = new A();
8.    obj.msg();
9.   }
```

```
10. }
Output:Hello
```

3) 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 of package by import fully qualified name

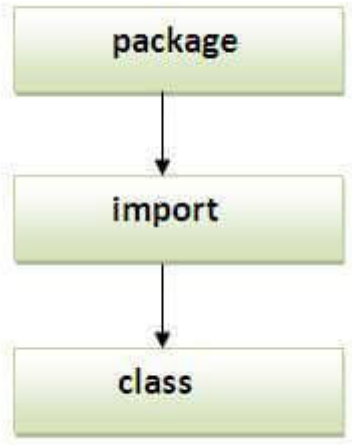
```
1. //save by A.java
2. package pack;
3. public class A{
4.   public void msg(){System.out.println("Hello");}
5. }
1. //save by B.java
2. package mypack;
3. class B{
4.   public static void main(String args[]){
5.    pack.A obj = new pack.A();//using fully qualified name
6.    obj.msg();
7.   }
8. }
Output:Hello
```

Note: If you import a package, subpackages will not be imported.

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.

Unit-1 Object Oriented Programming with Java

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



Subpackage in java

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

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

The standard of defining package is domain.company.package e.g. com.javatpoint.bean or org.sssit.dao.

Example of Subpackage

```
1. package com.javatpoint.core;
2. class Simple{
3.   public static void main(String args[]){
4.    System.out.println("Hello subpackage");
5.   }
6. }
```

To Compile: javac -d . Simple.java

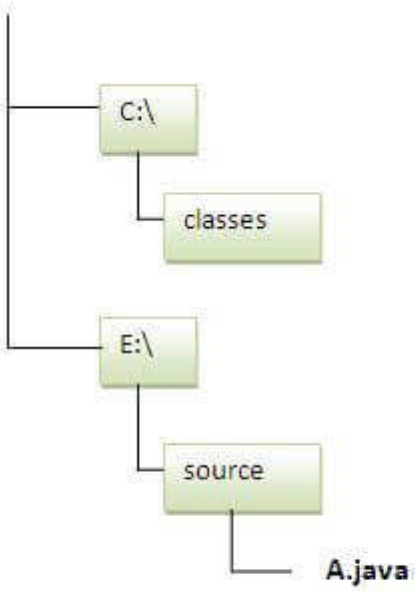
Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

To Run: java com.javatpoint.core.Simple

```
Output:Hello subpackage
```

How to send the class file to another directory or drive?

There is a scenario, I want to put the class file of A.java source file in classes folder of c: drive. For example:




```
1. //save as Simple.java
2. package mypack;
3. public class Simple{
4.  public static void main(String args[]){
5.     System.out.println("Welcome to package");
6.   }
7. }
```

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

e:\sources> set classpath=c:\classes;.;

e:\sources> java mypack.Simple

Another way to run this program by -classpath switch of java:

The -classpath switch can be used with javac and java tool.

To run this program from e:\source directory, you can use -classpath switch of java that tells where to look for class file. For example:

e:\sources> java -classpath c:\classes mypack.Simple

```
Output:Welcome to package
```

Ways to load the class files or jar files

There are two ways to load the class files temporary and permanent.

- Temporary
  - By setting the classpath in the command prompt
  - By -classpath switch
- Permanent
  - By setting the classpath in the environment variables
  - By creating the jar file, that contains all the class files, and copying the jar file in the jre/lib/ext folder.

Rule: There can be only one public class in a java source file and it must be saved by the public class name.

```
1. //save as C.java otherwise Compilte Time Error
2.
3. class A{}
```

```
4. class B{}
5. public class C{}
```

How to put two public classes in a package?

If you want to put two public classes in a package, have two java source files containing one public class, name same. For example:

```
1. //save as A.java
2.
3. package javatpoint;
4. public class A{}
1. //save as B.java
2.
3. package javatpoint;
4. public class B{}
```

What is static import feature of Java5?

Click Static Import feature of Java5.

How to Set CLASSPATH in Java

CLASSPATH: CLASSPATH is an environment variable which is used by Application ClassLoader to locate and load the .class files. The CLASSPATH defines the path, to find third-party and user-defined classes that are not extensions or part of Java platform. Include all the directories which contain .class files and JAR files when setting the CLASSPATH.

You need to set the CLASSPATH if:

- You need to load a class that is not present in the current directory or any sub-directories.
- You need to load a class that is not in a location specified by the extensions mechanism.

The CLASSPATH depends on what you are setting the CLASSPATH. The CLASSPATH has a directory name or file name at the end. The following points describe what should be the end of the CLASSPATH.

- If a JAR or zip, the file contains class files, the CLASSPATH end with the name of the zip or JAR file.
- If class files placed in an unnamed package, the CLASSPATH ends with the directory that contains the class files.
- If class files placed in a named package, the CLASSPATH ends with the directory that contains the root package in the full package name, that is the first package in the full package name.

The default value of CLASSPATH is a dot (.). It means the only current directory searched. The default value of CLASSPATH overrides when you set the CLASSPATH variable or using the -classpath command (for short -cp). Put a dot (.) in the new setting if you want to include the current directory in the search path.

If CLASSPATH finds a class file which is present in the current directory, then it will load the class and use it, irrespective of the same name class presents in another directory which is also included in the CLASSPATH.

If you want to set multiple classpaths, then you need to separate each CLASSPATH by a semicolon (;).

The third-party applications (MySQL and Oracle) that use the JVM can modify the CLASSPATH environment variable to include the libraries they use. The classes can be stored in directories or archives files. The classes of the Java platform are stored in rt.jar.

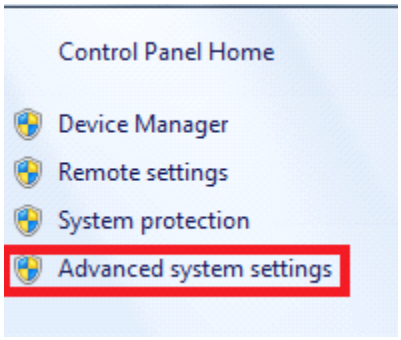
There are two ways to ways to set CLASSPATH: through Command Prompt or by setting Environment Variable.

Let's see how to set CLASSPATH of MySQL database:

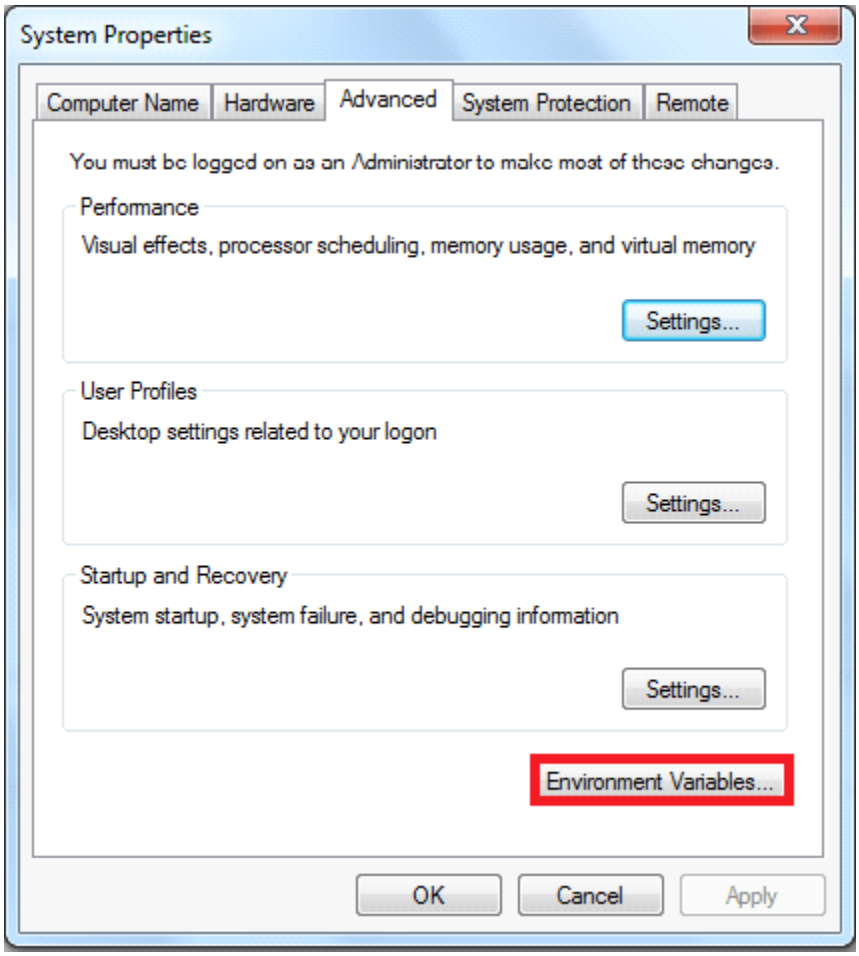
Step 1: Click on the Windows button and choose Control Panel. Select System.

Step 2: Click on **Advanced System Settings**.

Unit-1 Object Oriented Programming with Java



Step 3: A dialog box will open. Click on Environment Variables.

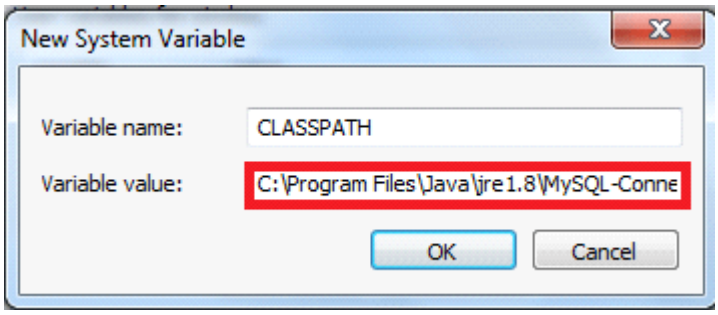


Step 4: If the CLASSPATH already exists in System Variables, click on the Edit button then put a semicolon (;) at the end. Paste the Path of MySQL-Connector Java.jar file.

If the CLASSPATH doesn't exist in System Variables, then click on the New button and type Variable name as CLASSPATH and Variable value as *C:\Program Files\Java\jre1.8\MySQL-Connector Java.jar;.;*

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

Remember: Put ;.; at the end of the CLASSPATH.



Difference between PATH and CLASSPATH

PATH	CLASSPATH
PATH is an environment variable.	CLASSPATH is also an environment variable.
It is used by the operating system to find the executable files (.exe).	It is used by Application ClassLoader to locate the .class file.
You are required to include the directory which contains .exe files.	You are required to include all the directories which contain .class and JAR files.
PATH environment variable once set, cannot be overridden.	The CLASSPATH environment variable can be overridden by using the command line option -cp or -CLASSPATH to both javac and java command.

How to Set CLASSPATH in Windows Using Command Prompt

Type the following command in your Command Prompt and press enter.

1. set CLASSPATH=%CLASSPATH%;C:\Program Files\Java\jre1.8\rt.jar;

In the above command, The set is an internal DOS command that allows the user to change the variable value. CLASSPATH is a variable name. The variable enclosed in

percentage sign (%) is an existing environment variable. The semicolon is a separator, and after the (;) there is the PATH of rt.jar file.

How ext folder works in Java

The ext directory works a bit like the CLASSPATH. ext directory is the part of the class loading mechanism. The classes which are available within JARs in the ext directory are available to Java applications.

The following table demonstrates the key difference between the CLASSPATH and Extension Mechanism:

Characteristics	CLASSPATH	Extension Mechanism
Class loading order	CLASSPATH loads after bootstrap and extension loading.	ext loads after bootstrap loading but before CLASSPATH loading.
Scope	It is an application specific. All JREs on the host is the CLASSPATH environment variable.	All JVMs are running in specific JRE java.ext.dirs.
Package name	java.class.path is used to find the directories and JAR archives containing class files.	java.ext.dirs is used to specify where the extension mechanism loads classes.
Specification	It is specified by name including the extension.jar and directory containing .class files.	All JAR files in specified directories are loaded.

The mechanism will pick up all .jar files from the extension directory even if the file does not have the .jar extension. The implementation of this is that if one can change the name of a jar placed in a classpath directory to have an extension other than .jar. The wildcard (*) does not pick it up. This technique will not work with the extension directory.

Unit-1 Object Oriented Programming with Java

Let's understand the execution process through an example.

A.java

```
1. public class A
2. {
3. public String toString()
4. {
5. return "hello";
6. }
   }
```

7. **B.java**

```
1. public class B
2. {
3. public static void main(final String[] args)
4. {
5. System.out.println(new A());
6. }
7. }
```

Compile the A.java file. we will archive the compiled A.class file into A.jar. Place this JAR file into another directory than the compiled B.class file.

To demonstrate the use of the classpath, we place the A.jar file in a directory C:\JavaPrograms and will access that JAR through wildcard (*) for B to use.

We found that B can still load the A.class while we had deleted it from the current directory. The Java launcher was explicitly looked for C:\JavaProgram. It is also possible to have the class loaded without its presence in the same directory and explicit classpath specification.

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

Unit-1 Object Oriented Programming with Java

It is often referred to as a benefit of Using the extension mechanism because all applications which are using that JRE can see the same classes without the need to specify them on the classpath explicitly.

What happens if we change the name of A.jar into A.backup in the same CLASSPATH-referenced directory. NoClassDefFoundError is encountered when we do the same because the CLASSPATH-reference does not have the .jar extension.

Java Create Jar Files

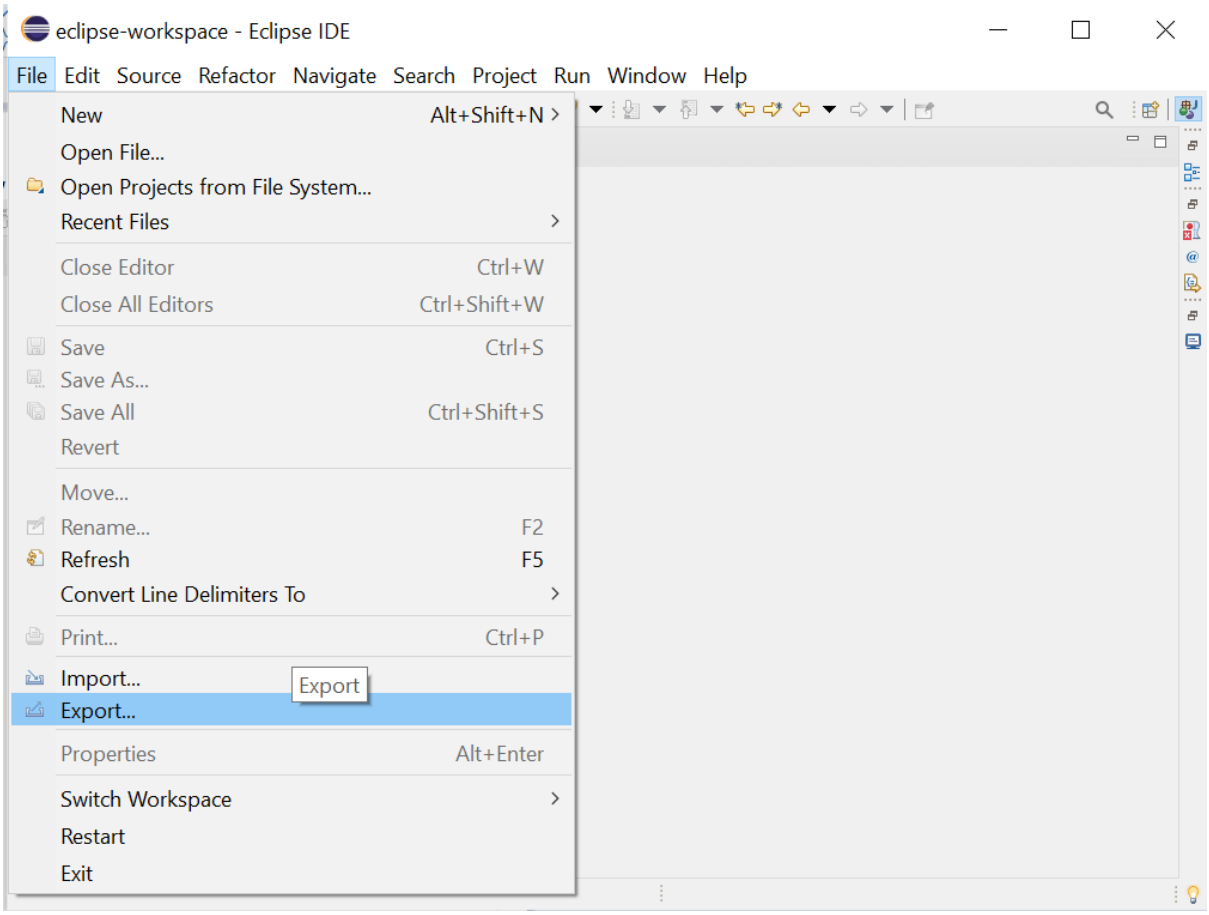
In Java, JAR stands for Java ARchive, whose format is based on the zip format. The JAR files format is mainly used to aggregate a collection of files into a single one. It is a single cross-platform archive format that handles images, audio, and class files. With the existing applet code, it is backward-compatible. In Java, Jar files are completely written in the Java programming language.

We can either download the JAR files from the browser or can write our own JAR files using **Eclipse** IDE.

The steps to bundle the source code, i.e., .java files, into a JAR are given below. In this section, we only understand how we can create JAR files using eclipse IDE. In the following steps, we don't cover how we can create an executable JAR in Java.

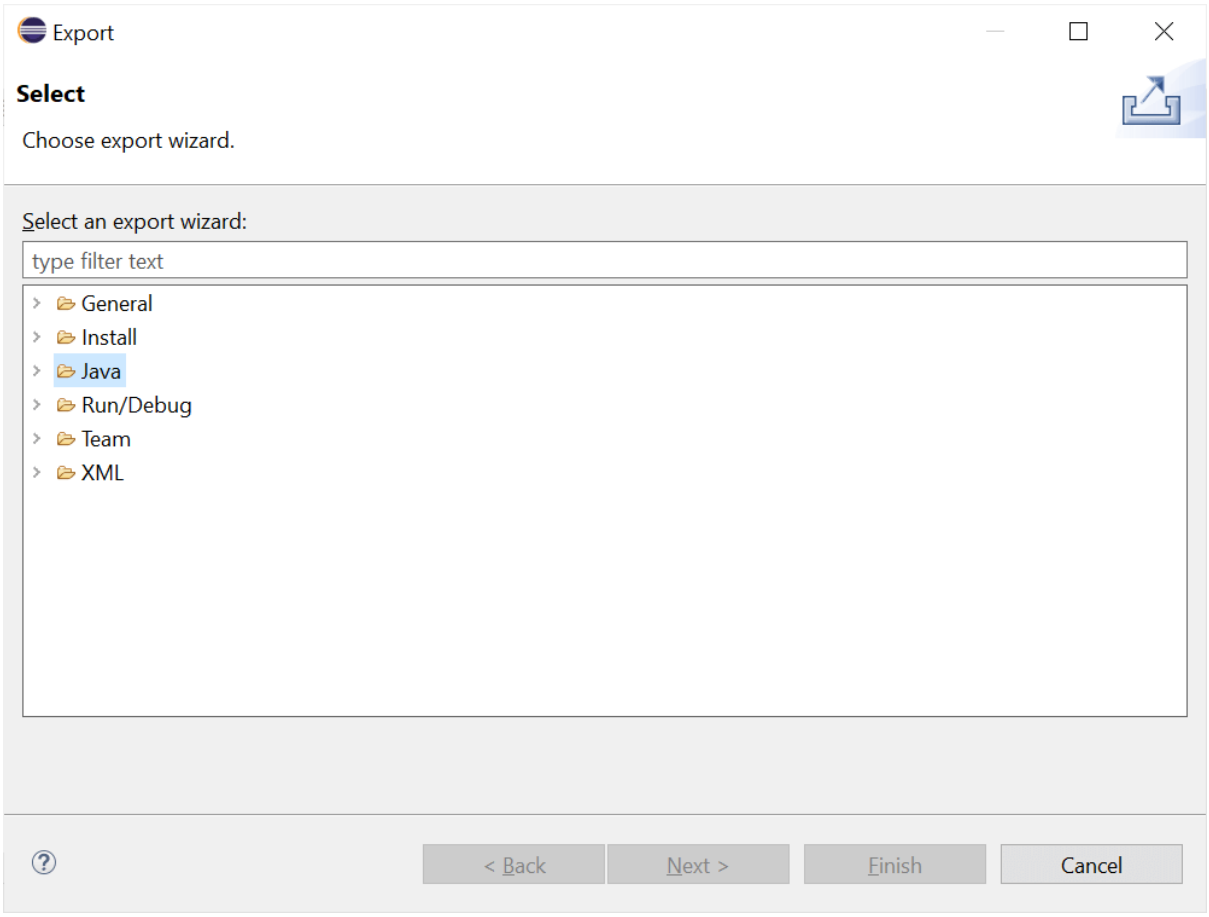
1. In the first step, we will open Eclipse IDE and select the **Export** option from the **File** When we select the Export option, the Jar File wizard opens with the following screen:

Unit-1 Object Oriented Programming with Java



Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

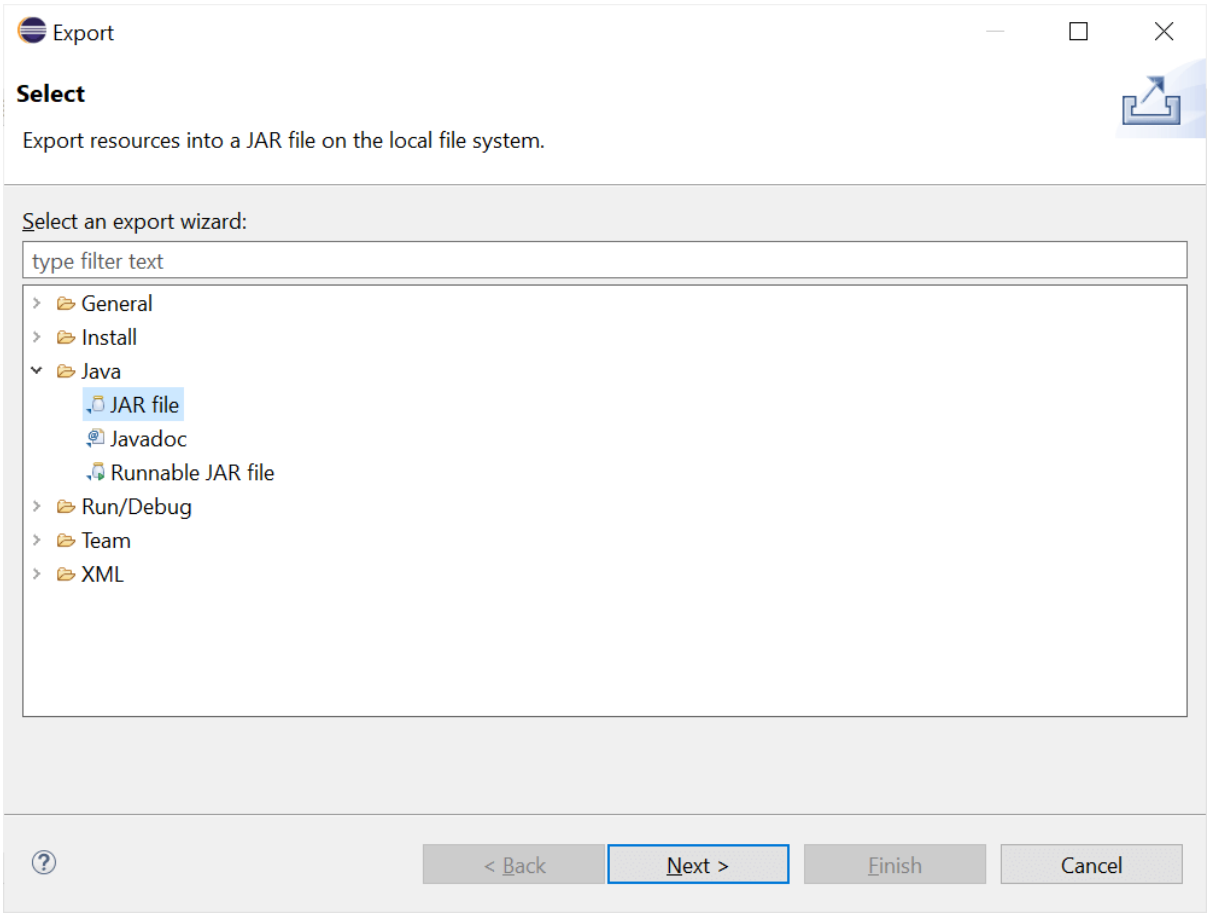
Unit-1 Object Oriented Programming with Java



2. From the open wizard, we select the Java **JAR file** and click on the **Next** The Next button opens JAR Export for JAR File Specification.

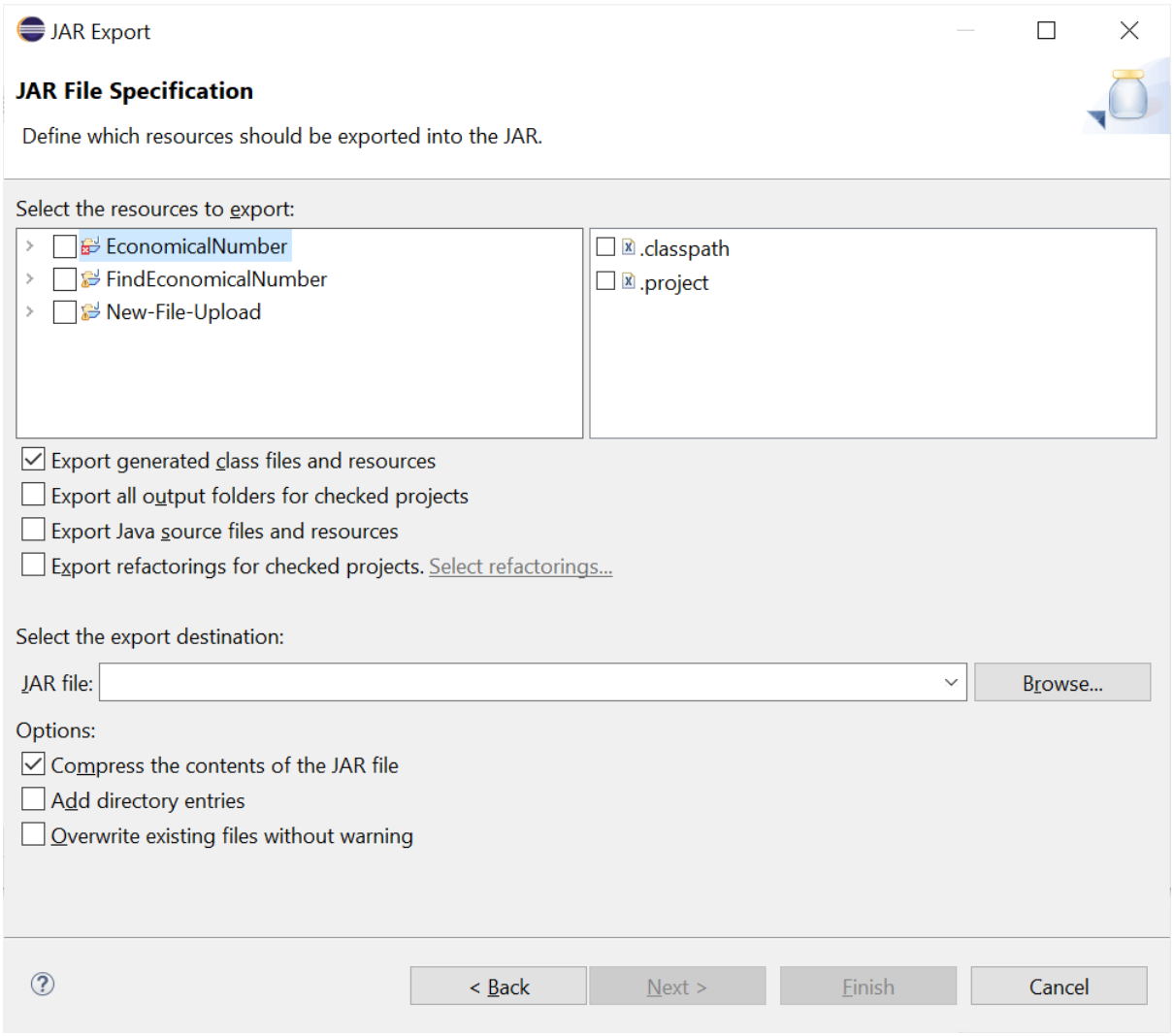
Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

Unit-1 Object Oriented Programming with Java



Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

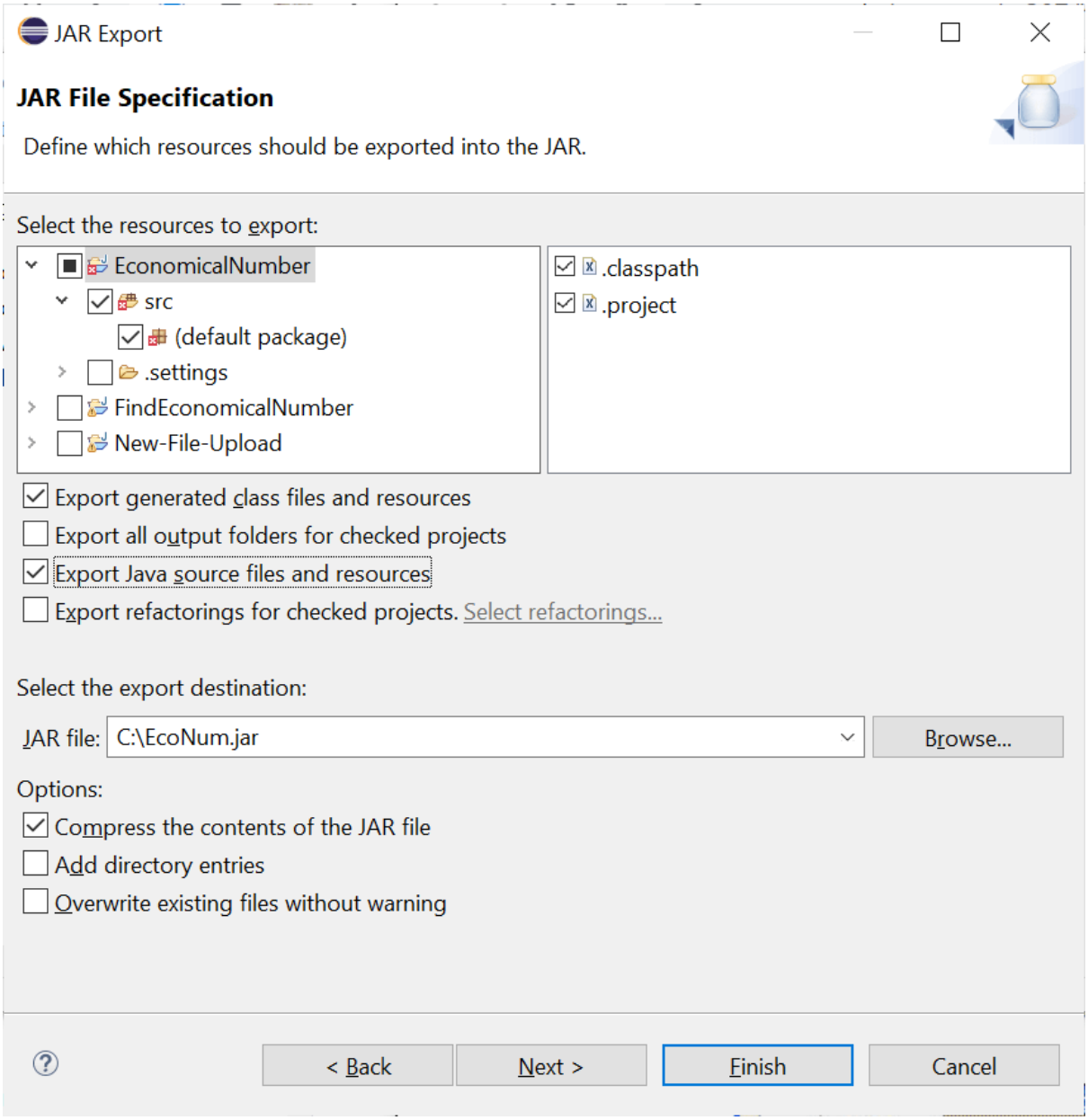
Unit-1 Object Oriented Programming with Java



3. Now, from the JAR File Specification page, we select the resources needed for exporting in the **Select the resources to export** After that, we enter the JAR file name and folder. By default, the **Export generated class files and resources** checkbox is checked. We also check the **Export Java source files and resources** checkbox to export the source code.

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

Unit-1 Object Oriented Programming with Java

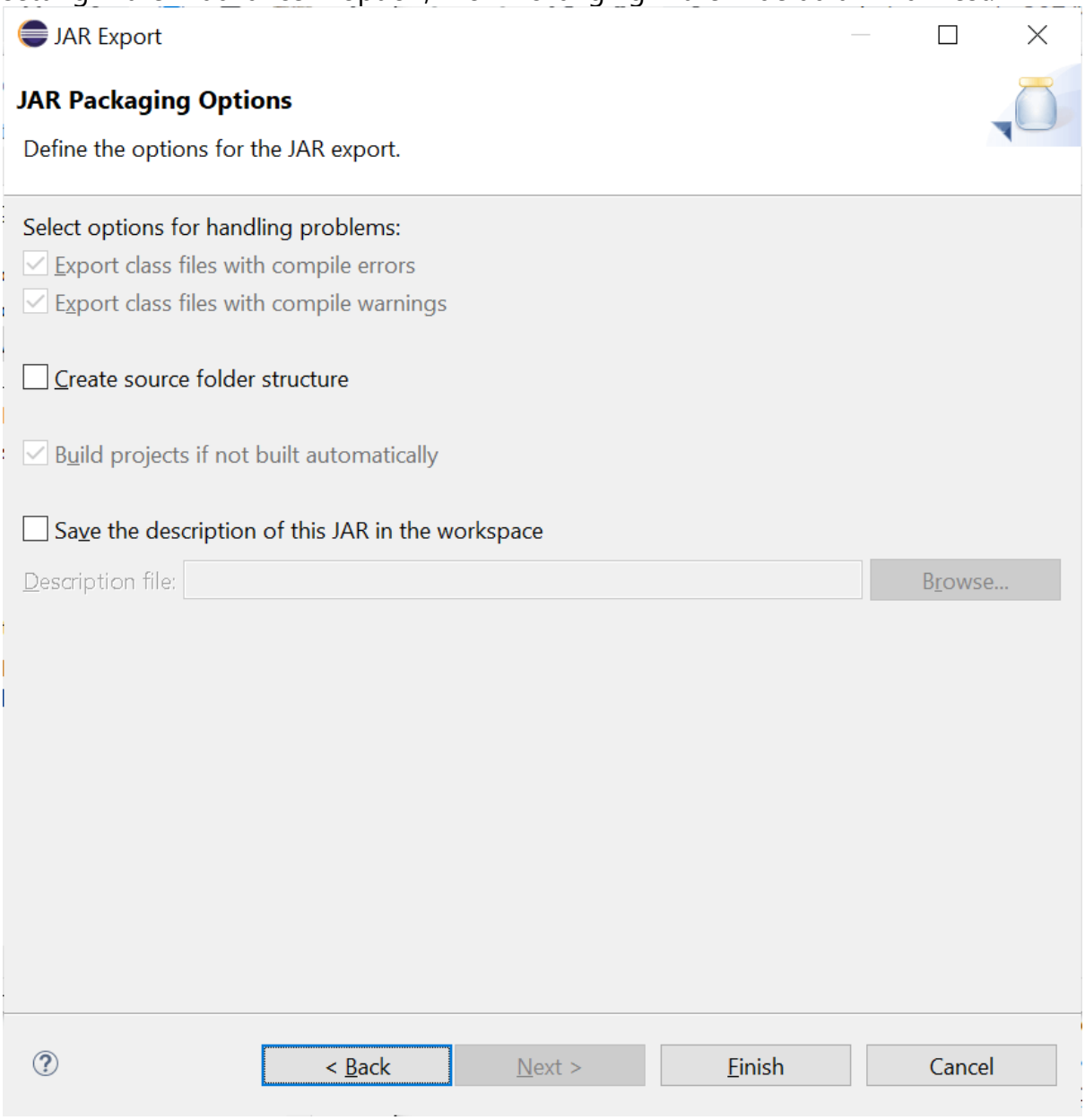


If there are other Java files or resources which we want to include and which are available in the open project, browse to their location and ensure the file or resource is checked in the window on the right.

4. On the same page, there are three more checkboxes, i.e., **Compress the content of the JAR file, Add directory entries**, and **Overwrite existing files without warning**. By default, the **Compress content of the JAR file** checkbox is checked.
5. Now, we have two options for proceeding next, i.e., **Finish** and **Next**. If we click on the **Next**, it will immediately create a JAR file to that location which we

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

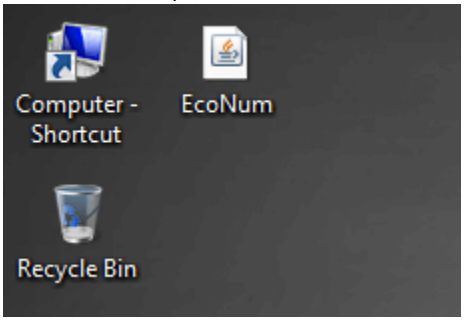
defined in the **Select the export destination**. If we click on the **Next** button, it will open the Jar **Packaging Option** wizard for creating a JAR description, setting the advance option, or changing the default manifest.



For now, we skip the **Next** and click on the **Finish** button.

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com

6. Now, we go to the specified location, which we defined in the **Select the export destination**, to ensure that the JAR file is created successfully or not.



Java Static Import

The static import feature of Java 5 facilitate the java programmer to access any static member of a class directly. There is no need to qualify it by the class name.

Advantage of static import:

- Less coding is required if you have access any static member of a class oftenly.

Disadvantage of static import:

- If you overuse the static import feature, it makes the program unreadable and unmaintainable.

Simple Example of static import

```
1. import static java.lang.System.*;
2. class StaticImportExample{
3.   public static void main(String args[]){
4.
5.    out.println("Hello");//Now no need of System.out
6.    out.println("Java");
7.
8.  }
9. }
```

Test it Now

```
Output:Hello
       Java
```

Unit-1 Object Oriented Programming with Java

What is the difference between import and static import?

The import allows the java programmer to access classes of a package without package qualification whereas the static import feature allows to access the static members of a class without the class qualification. The import provides accessibility to classes and interface whereas static import provides accessibility to static members of the class.

Faculty: SHAHRUKH KAMAL
Shahrukhkamal7@gmail.com