Packages

- A package is a collection of related classes and interfaces that provides access protection and namespace management.
- Resolves class name clashing by giving a class fully qualified name.
- Packages are created using the *package* keyword.

Example:

```
package graphics;
```

Declaring a package

- Eg 1:
- package p1;
- class A{
- }
- }

Eg2 :

package p1.p12.p123;

class B{

- Classes belonging to one package are stored in the subdirectory whose name is same as the package name.
- Package member can be accessed by using *import* keyword.
- Only public members of package are accessible to universe.

Access Specifiers

- Java provides four distinct access specifiers for class members.
  - private
  - protected
  - public
  - default / (package wide scope)

Java Class Path

- Both compiler and interpreter search for classes in the specified directory.
- The directory specification is provided by CLASSPATH environment variable.
- A class path is a list of directories or zip files to search for the class files.
- The classpath can also be passed explicitly to *javac* or *java*

Access Control

	Private	Default	Protected	Public
Class	✓	✓	✓	✓
Sub class, same package	×	✓	✓	✓
Other class, same package	×	✓	✓	✓
Sub class, diff package	×	×	✓	✓
Other class, diff package	×	×	×	✓

JAR

- JAR(java archive) is a file that has a collection of the class files required for the application.
- It helps us distribute classes for the application easily and also helps in setting the **`classpath`**.
- Important thing about the **`jar`** file is that it maintains the directory structure.
- So if we want a class **`Student.class`** to be inside **`student`** directory that can be maintained in the jar file.
- **`jar`** command can be used to create and update a **`jar`** file.
- It can also be used to extract a jar file.

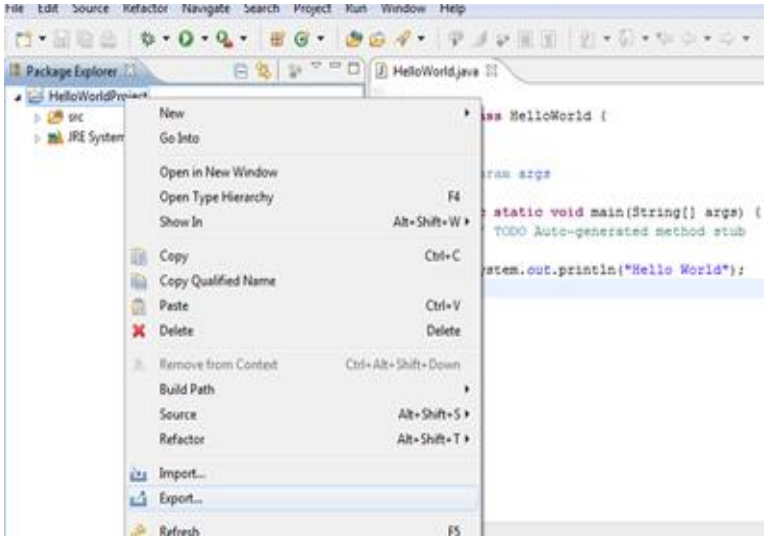
`jar` command

Please read Page 21 of Packages.pdf detailing JAR commands .

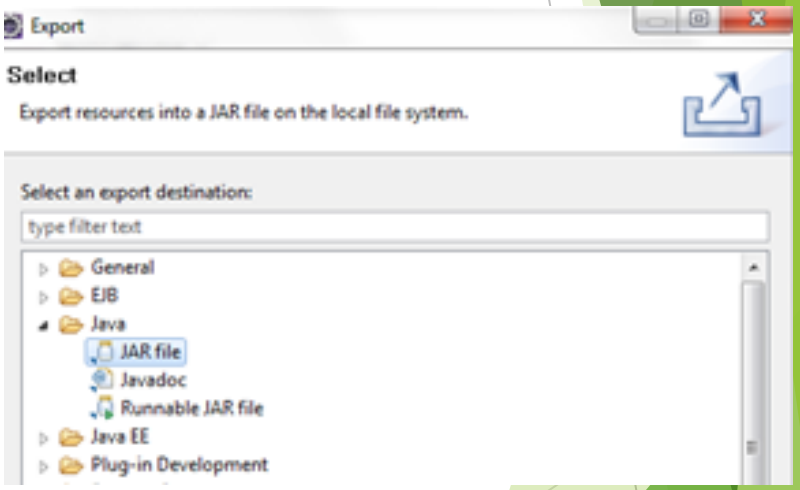
Running project outside Eclipse

- To run your Java program outside of Eclipse you need to export it as a jar file. A jar file is the standard distribution format for Java applications.

Step 1: Select your project, right click on it and select Export.



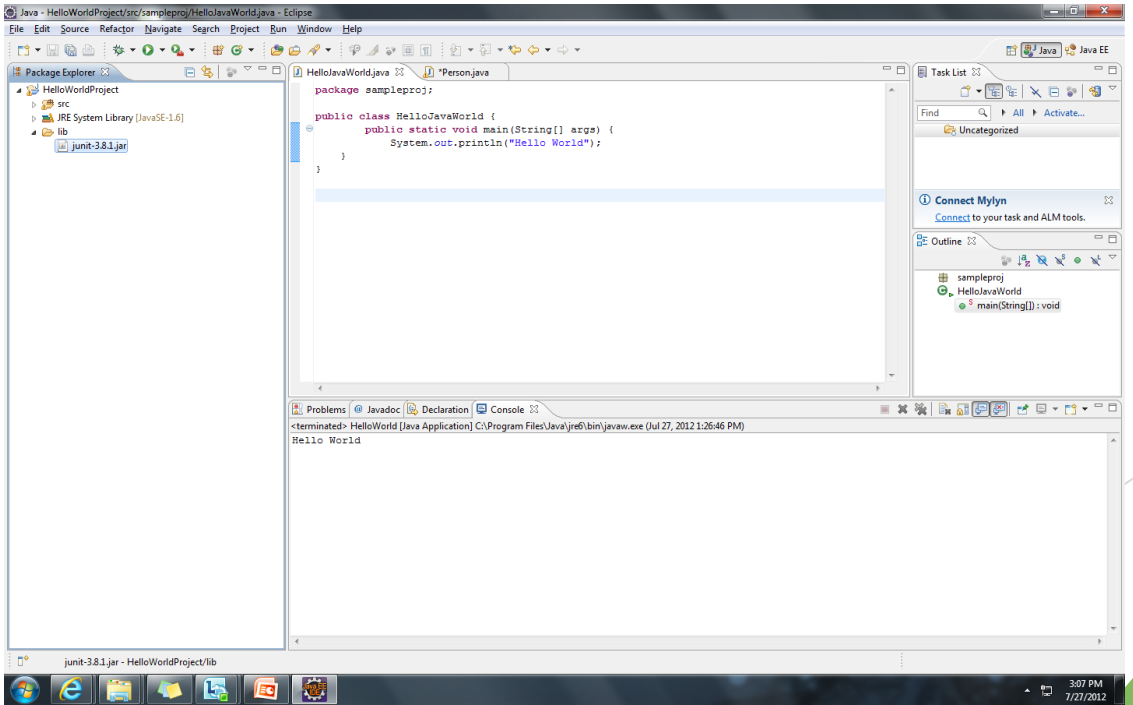
Step 2: Select JAR File from the Export window. Select Next



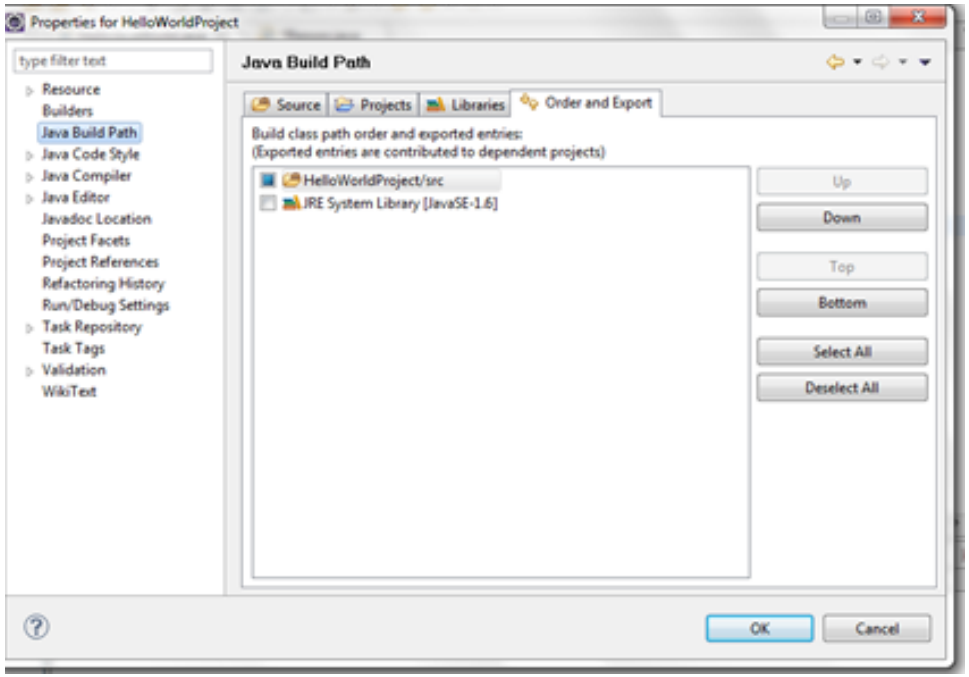
Select the project and specify the export destination and a name for the jar file.
Select Finish.
This jar file can be executed by including it in the classpath.

- The Steps to add a library (.jar) to the project are as follows.

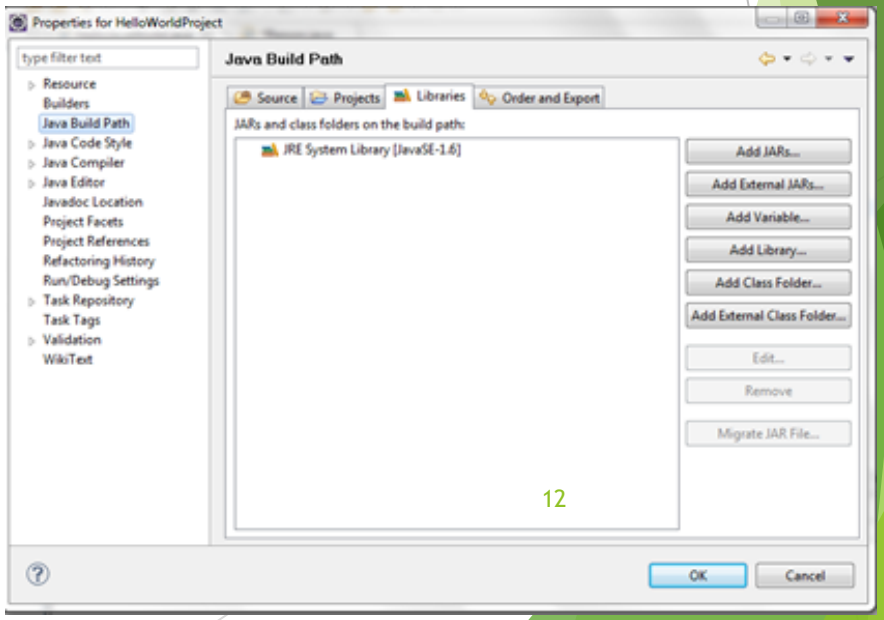
Step 1: Import the jar to the project by selecting File → Import → General → File System. Select the jar and select the folder lib as target. Alternatively, just copy and paste the jar file into the "lib" folder.



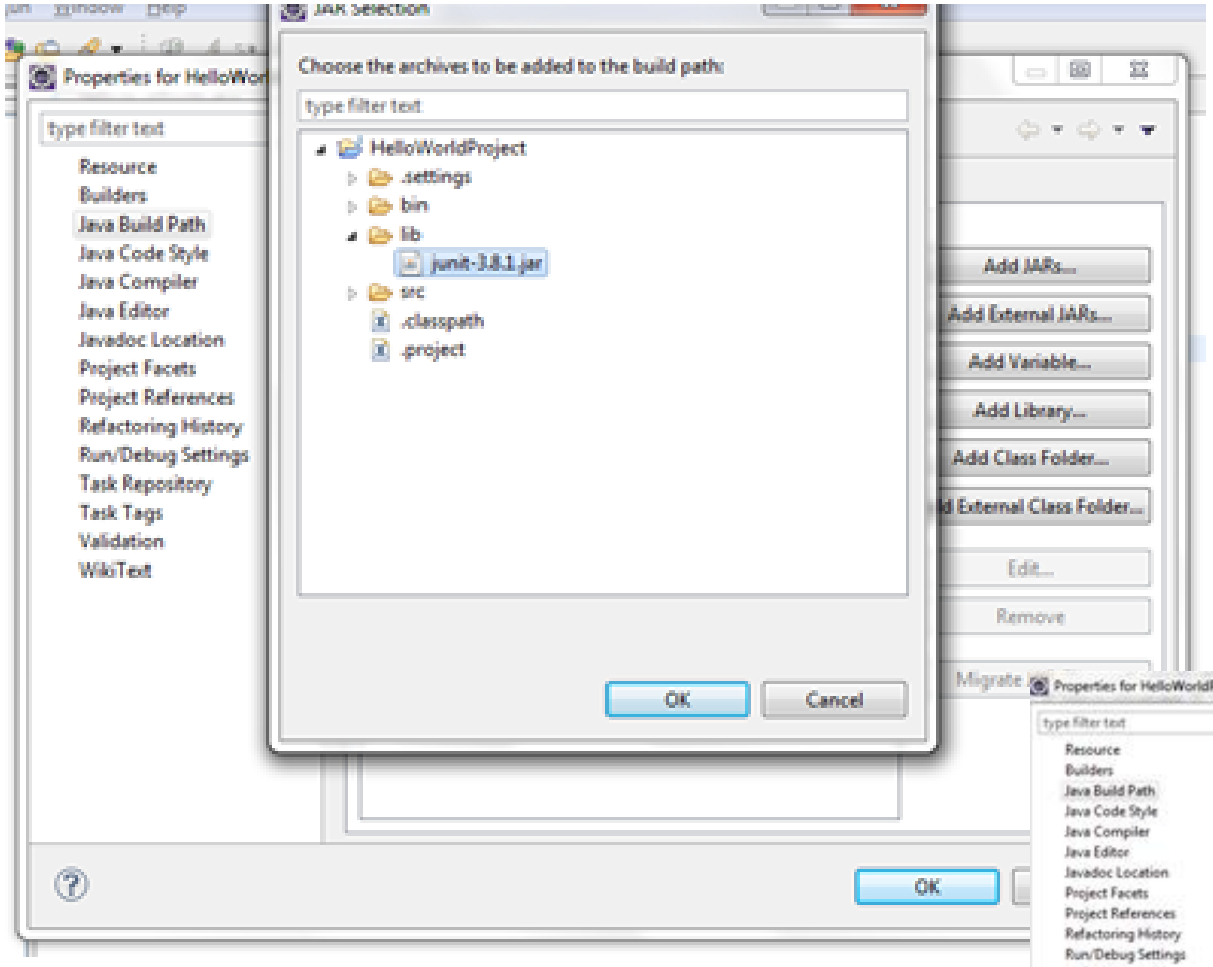
Step 2: Right click on the project and select Properties and select Java Build Path

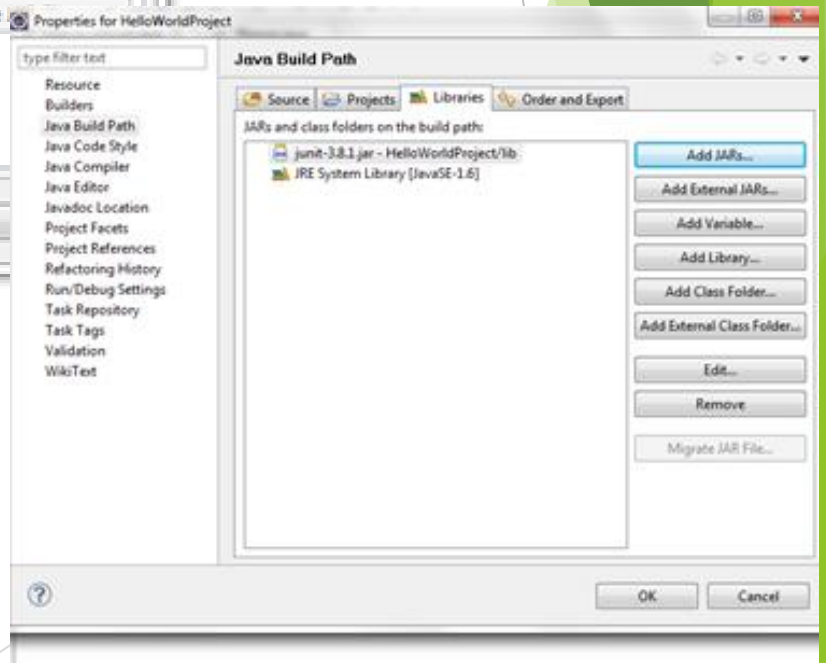


Step 3: Select “Add JARs” or “Add External JARs”



Step 4: Select the Jar to be added and click OK.





Imports

- Java provides an easy way to access classes in other packages instead of using long names. This is done by using **`import`** statement.
- The same is the case with static members of a class too. Static members of a class are accessed using *ClassName.staticMemberName*. This name also sometimes could be very long. Instead imports could be used such that only static member name can be used.
- Two forms of imports:
  - Class imports
  - Static imports

Class Imports

a) Importing only a particular class.

Example:

```
import student.Student;
public class Tester{
…
Student s= new          Student("John","M.C.A.");
}
```

b) Importing all the classes in a package.

Example:

```
import student.*;
public class Tester{
…
Student s= new              Student("John","M.C.A.");
}
```

Static Imports

- Makes the static members of a class available to code the directly without explicitly specifying the class name.
- Syntax:

```
import static packageName.ClassName.*;
```

(imports all the static members)

Or

```
import static packageName.ClassName.staticMember;
```

(imports only the particular ‘staticMember’)

Example: Static Imports

Static member of System

```
import static java.lang.System.out;
import static teacher.Grade.*;
import teacher.*;

class GradeTest{
public static void main(String str[]){
Teacher f=new teacher.Teacher("Tom");
Grade gf= new Grade(f,new
student.Student("Malini"),"001",B);
out.println("Grade "+ gf.getGrade());}
}
```

Instead of System.out.println

Instead of Grade.B

Tell me how?

- If importing using * means all classes are imported, is it not better to import the required class to avoid loading of all classes?

- Java loads the classes on demand. This is feature called Dynamic Linking.
- Only the classes whose members are invoked in some are loaded.
- Hence both using * or explicitly specifying class, will not impact class loading.
- One advantage of using an explicitly specified class name is for better code comprehension.

Default Access Specifier

- Classes or members having default or package access specifier can be accessed only by the classes belonging to the same package.
- There is no keyword for default access specifier.

```
public class Student{
private int regNo;
public  int getRegNo(){..}
int getRegNo(){..} // default access
}
```

Packages - Not truly nested

- To organize the name in meaningful manner packages can be created such that classes can be placed inside multiple folders which are in hierarchy or nested.
- Although packages appear to be hierarchical or nested with respect to folder, they are not.

```
package com;
  public class A{}
 package com.util;
  public class B{}
```

To use A and B in a class outside the package we must import both the packages separately.

```
import com.*;
import com.util.*;
public class C{
A a;
B b;}
```

Java source file rules

So far, we have been typing all the java classes in a separate file. And of course that is the recommended way. But if you still feel that it will save time if you put some of your related classes together in the same source file then here are the rules that the compiler insists on.

- A java source file can contain:
  1. a single package statement which will be the first statement.
  2. any number of import statements. They should be between package and class statement.
  3. any number of classes with default access specifier. In such cases, the file name could be 'anything'.java.
  4. only one public class. If there is a public class then the file must be named after the public class name.
  5. The package and the import statements apply to all the classes in the source file.

Standard packages in JSE

- Some of the important packages :

a) **`java.lang`** : All the classes that are fundamental to the Java programming language. **`String`** class is a part of **`java.lang`** package. This package is automatically imported by all java classes.

b) **`java.util`** : All the utility classes like **`Date`**, **`LinkedList`** etc. which are frequently used in applications.

c) **`java.io`** : All the classes related to IO.

d) **`java.sql`** : All the classes related to database.

e) **`java.awt`** : All the classes related to building GUI

- The packages in JDK begin with **`java`** or **`javax`**. So it is recommended that we don't use **`java`** or **`javax`**. when we define our package.
- Search for **`rt.jar`** in your system or in eclipse open **`rt.jar`** inside JRE system library and look for the above packages.

Exercise

- *Create a new project in which create a package named org.animals. In that create various classes like Lion, Monkey, Elephant. In each class create data members like color, weight and age. Create methods like isVegetarian, canClimb, getSound.*
- *Create another project and in that create a package called zoo and create a class called VandalurZoo and create objects for the animals that are existing in zoo and print the characteristic of each animal.*

(15 mins)