


Packages

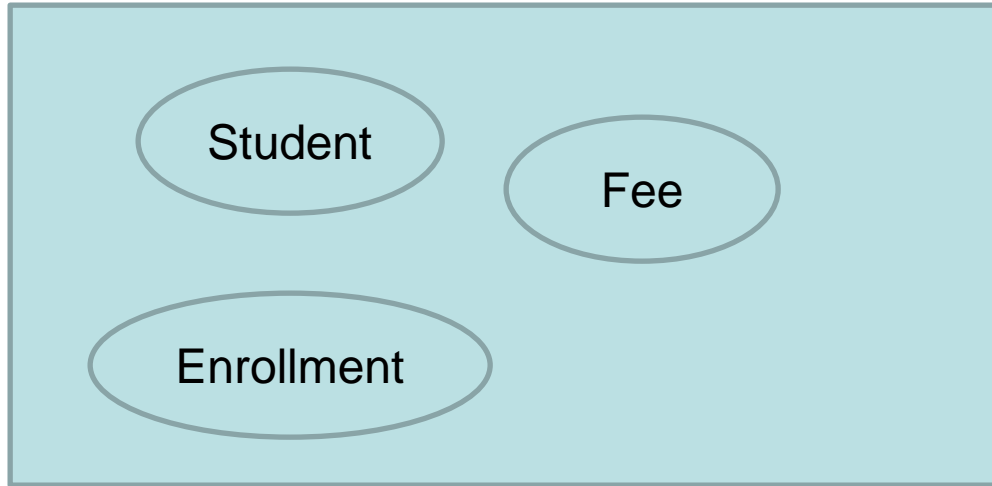


Definition

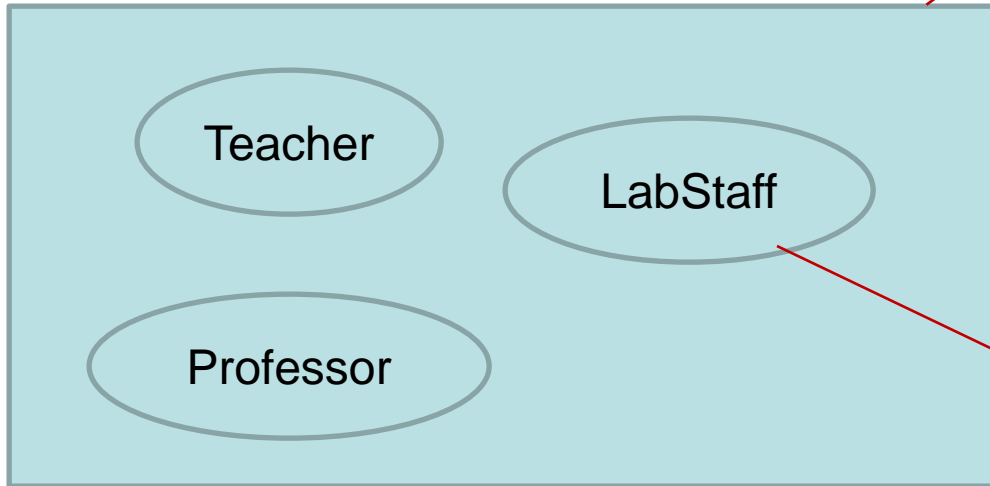
- Package is a grouping mechanism that holds a set of classes (or interfaces)  **later**
- Classes can be grouped under a package for reasons of similar / shared responsibilities, for reasons of organization and so on
- Packages are implemented using file system directories.
- Package must be the first statement in the java source file.
- Concept of packages also allow us to specify visibility of classes across packages.
- Syntax for creating packages:
`package package_name[.package_name] ;`
- So far classes that were created were in an unnamed package. This is fine for a small or temporary applications but not recommended for projects.

Example Scenario

studentManagement

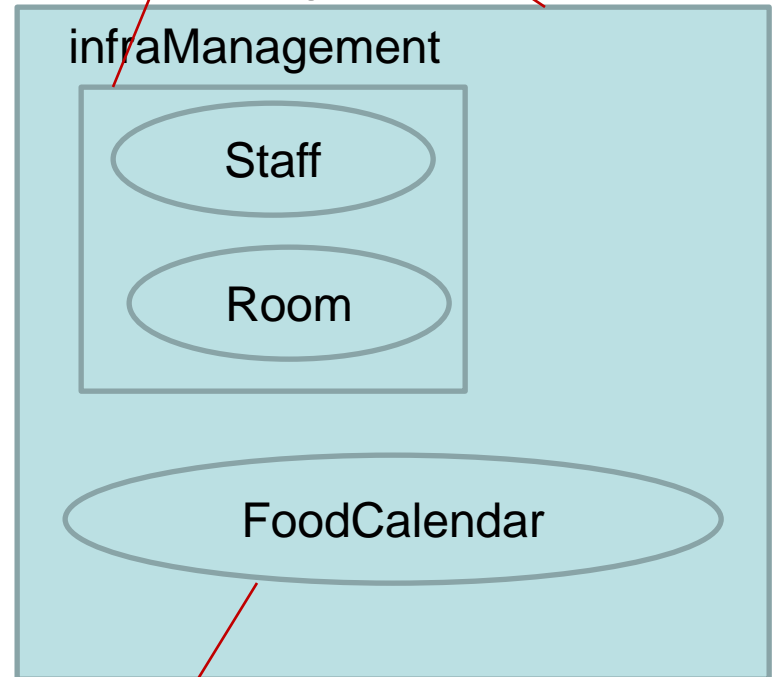


facultyManagement



packages

hostelManagement



Classes / interfaces

- A college application will consist of automating several aspects. For example features related to
 - Student management
 - Faculty management
 - Hostel management
 - Payroll
- It is easier to work with a set of classes that are grouped in accordance to shared / similar responsibilities. In other words, if we had a package called student, all the classes related to student, like Enrollment, Fee etc. can be placed in it.
- Using packages also assists in organization of the code. Rather than have dozens of classes all around, a higher level view of a smaller number of packages would be simpler.
- Work allocation could also be done in accordance with packages.
- As we develop our application to include all the above features we will find that we have lots of classes, and sometimes we may find that we have difficulty finding unique names and therefore we end up with the class names becoming too big and complicated.

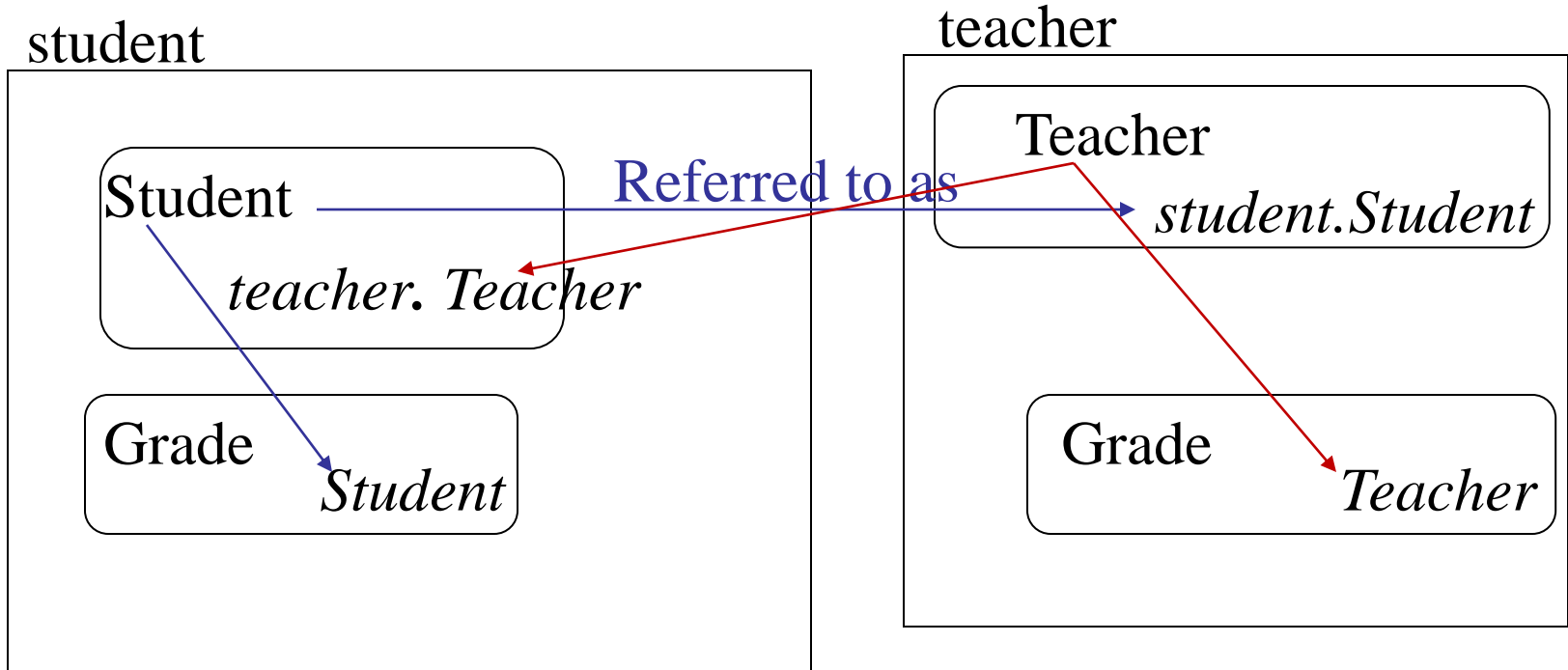
Benefits

- It is easier to work with a set of classes that are grouped in accordance to shared / similar responsibilities.
- Easy to locate the classes.
- Eliminates name collision
- You can restrict the access to your classes.
- Work allocation could also be done in accordance with packages.

Creating and accessing classes in a package

- We will create 2 packages with 2 classes each
 - **student**
 - **Student**
 - **Grade**
 - **teacher**
 - **Teacher**
 - **Grade**
- Packages are named like variables except that they are all in lower case and can contain dots (.)
- `Teacher` can access `Student` using its fully qualified name `student.Student`.
- `Grade` defined inside `student` package can access `Student` without using full qualified name.

Access



Outside the student package, the `Student` class is now to be referred to as `student.Student`.

Example 1 : code for student package

```
package student;  
  
public class Student{  
    // same code what we wrote in the last session  
}
```

Student.java

Complete code at the end of the slides

Since Student class is inside the same package, full-name of the class is not required.

```
package student;
public class Grade {
    private Student student;
    private int mark[];
    private final static String O="Outstanding";
    private final static String A="V.Good";
    private final static String B="Good";
    private final static String C="Average";
    private final static String F="Fail";

    public static void main(String str[]) {
        Grade g=new Grade(new Student("Raja"),new int[]{ 80,85,
        91,86, 82});
        System.out.println(g.getGrade()); }

    public Grade(Student s, int mark[]){
        this.student=s;
        this.mark=mark;    }
```

Grade.java

```

public String getGrade() {
int percent=getPercent();
if(percent>=90)    return O;
else  if(percent<90 && percent>=80)
return A;
else  if(percent<80 && percent>=60)
return B;
else  if(percent<60 && percent>=50)
return C;
else  return F;
}
int getPercent(){
        int tot=0;
        for(int i=0;i<mark.length;i++)
            tot=tot+mark[i];
    return (tot/mark.length);
}
}

```

Executing classes in a package -command line

- Compile the files as :
 - `D:\MyJava> javac *.java`
- Trying to execute
 - `D:\MyJava> java Grade`
- Results in run time error is generated -
 - Exception in thread "main"
`java.lang.NoClassDefFoundError: Grade (wrong name: student/Grade)`
- Let us try
 - `D:\MyJava> java student.Grade`
- Again run time error is generated.

Folder used to implement packages

- Java implements packages using the folder concept.
- The classes in a package must be inside the folder that is named after the package.
- The error in the previous slide was because JVM was trying to locate **Grade** class inside the folder **student**.
- So if we copy the Student.class and Grade.class inside student folder, the code will compile.
- There are good compilation options also that we can use instead of manually copying the class files..

Command to compile into package folders

- Shortcut during compilation:

```
javac -d <directory> *.java
```

```
D:\MyJava> javac -d . *.java
```

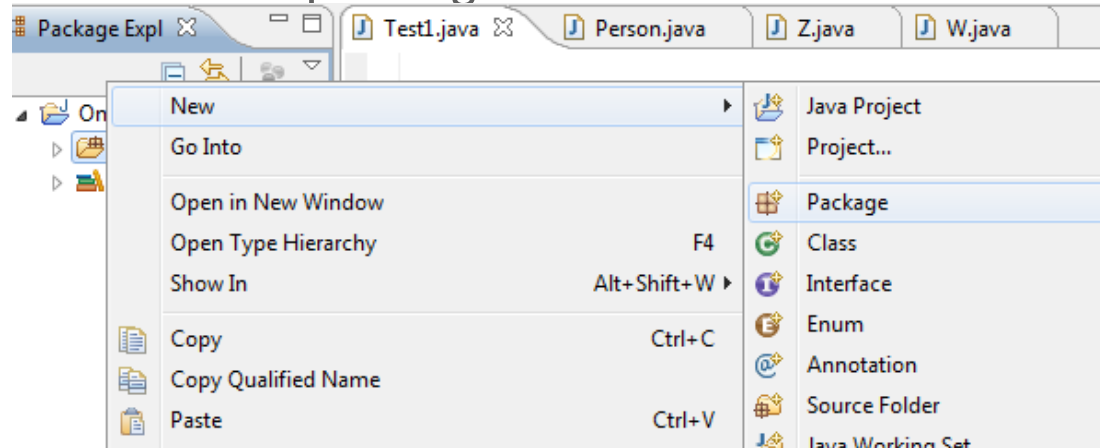


Creates a folder in the current directory and puts **Student.class** and **Grade.class** inside it.

- **D:\MyJava> java student.Grade**
- Executes the code successfully
- Please note that the execution does not happen from inside the **student** folder. It happens from one folder above the **student** folder. That is why the JVM is able to locate student folder and **Grade** class inside it.

Working on a package from eclipse

- Create a package
 - Right click on the “src” and click on New→Package.
 - Enter the package name and click finish



- Create a class inside the package
 - Right click on the newly created package and click on New→Class.
 - Rest is the same.



- The IDE will automatically find the packages in the same project. Compilation and execution is simple and straight forward from IDE.

Example 2 : code for teacher package

```
package teacher;

public class Teacher{
    private int factId;
    private String name;
    private static int id;

    public int getFactId(){return factId;}
    public String getName(){return name;}
    public void setName(String name){this.name=name;}
    private int generateId()
    {id++;
    return id;  }
    public static int getId(){return id;}

    public Teacher(String name){this.name=name;
    factId=generateId();
    }
}
```

Teacher.java

```
package teacher;
```

```
public class Grade{
```


```
    private Teacher faculty;
```

```
    private student.Student student;
```

```
    private String subjectCode;
```

```
    private String grade;
```

student who has graded the teacher-
fully qualified name



```
public final static String A="EXCELLIENT";
```

```
public final static String B="GOOD";
```

```
public final static String C="AVERAGE";
```

```
public final static String D="POOR";
```

```
public Grade(Teacher faculty, student.Student student,
```

```
String subjectCode, String grade){
```

```
    setFaculty(faculty);setStudent(student);
```

```
    setSubjectCode(subjectCode);
```

```
    setGrade(grade) ;
```

```
}
```

Grade.java


```
public Teacher getFaculty() {  
    return faculty;    }
```

```
public void setFaculty(Teacher fact) {  
    this.faculty=fact;    }
```

```
public void setStudent(student.Student  
student) {this.student=student;}  
public student.Student getStudent() {  
    return student;}  

```

```
public void setSubjectCode(String subjectCode) {  
    this.subjectCode=subjectCode; }  
public String getSubjectCode() {  
    return subjectCode;}  

```

```
public void setGrade(String grade) {  
    if (grade.equals(Grade.A) || grade.equals(Grade.B) ||  
        grade.equals(Grade.C) || grade.equals(Grade.D))  
        this.grade=grade;  
    else  
        this.grade=Grade.D;  
}
```

```
public String getGrade() {  
    return grade;  
}  
}
```

Testing the whole application

```
public class Tester{  
    public static void main(String str[]){  
  
        /*accessing student package */  
        student.Student s1= new student.Student("Malini");  
        System.out.println("Name of the student "+  
            s1.getName()+" ("+ s1.getRegNo()+"") );  
  
        student.Grade g=new student.Grade(s1,new int[]{  
            80,85, 91,86, 82});  
        System.out.println(g.getGrade());  
  
        /*accessing teacher package */  
        teacher.Teacher f=new teacher.Teacher("Tom");
```

```
teacher.Grade gf= new teacher.Grade(f,s1,"001",  
teacher.Grade.B) ;
```

```
System.out.println("Teacher Name "+  
gf.getFaculty().getName()+"-  
"+gf.getFaculty().getId());
```

```
System.out.println("Student who graded "+  
gf.getStudent().getName());
```

```
System.out.println("Grade "+ gf.getGrade());  
}  
}
```

Activity: Executing the Test class from command line

- Let us assume a directory structure as given below:
 - D:\MyJava : **Student.java** , **Grade.java**
 - D:\MyJava1 : **Teacher.java** , **Grade.java**
 - D:\Test : **Tester.java**
- Compilation of **student** package is done. So we have D:\MyJava\student folder with **Student.class** and **Grade.class**.
- Since **Grade** class of the **teacher** package refers to **Student** class, we need to tell JVM where to find it. We can do this in 2 ways
 - By setting **classpath**.
 - **set classpath=%classpath%;D:\MyJava**
 - By copying the student folder inside D:\MyJava1

- Compile **classes** in **teacher** package
 - `javac -d *.*.java`
- Similarly **Tester** class also refers to classes in both the packages. We either
 - Set the class path to
 - `set classpath=%classpath%;D:\MyJava1;.`
 - `set classpath=%classpath%;D:\MyJava1;D:\MyJava;. (in case we have opened a new command prompt session)`
 - Copy the student and teacher folder inside D:\Test
- Finally we compile and execute **Tester** class.

Tell me how?

- The previous exercise is fine if we are working on the same machine. In a project this is rarely the case. How can we distribute the classes in cases where a team is involved?
- Answer to this lies in trying to answer a very general question?
 - How will you normally distribute a set of files arranged in folders, say over the internet?
- If your answer is by creating a zipped file or Win RAR or by compressing it - it is right. And this is what Java has also provided for distribution.
- JAR or Java Archive.

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 also be used to extract a jar file.

jar command

- Creation of jar file

- `jar -cvf filename.jar *.*`

- Updation:

- `jar -uvf filename.jar newfilename.ext`

- Extraction :

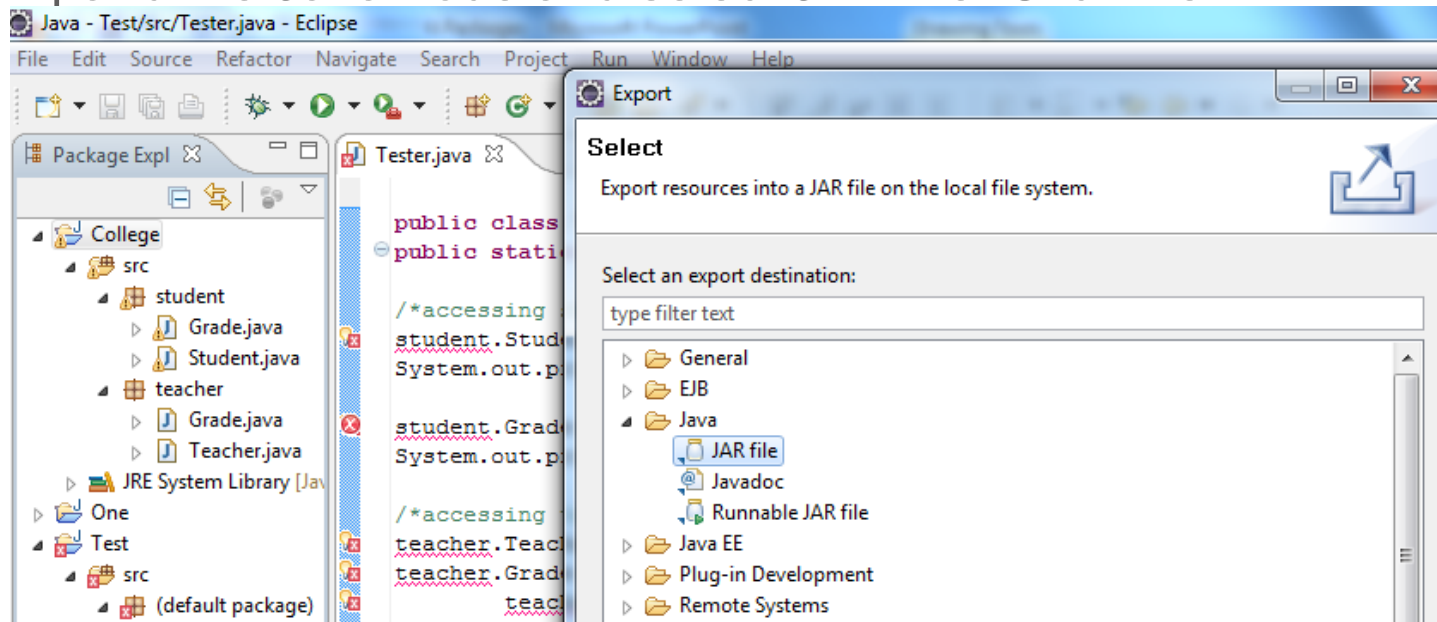
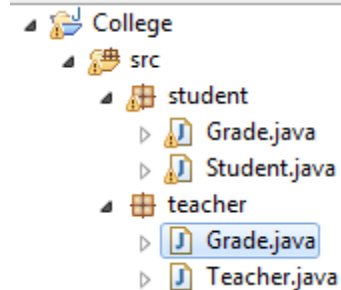
- `jar -xvf filename.jar`

Activity: Creating JAR file

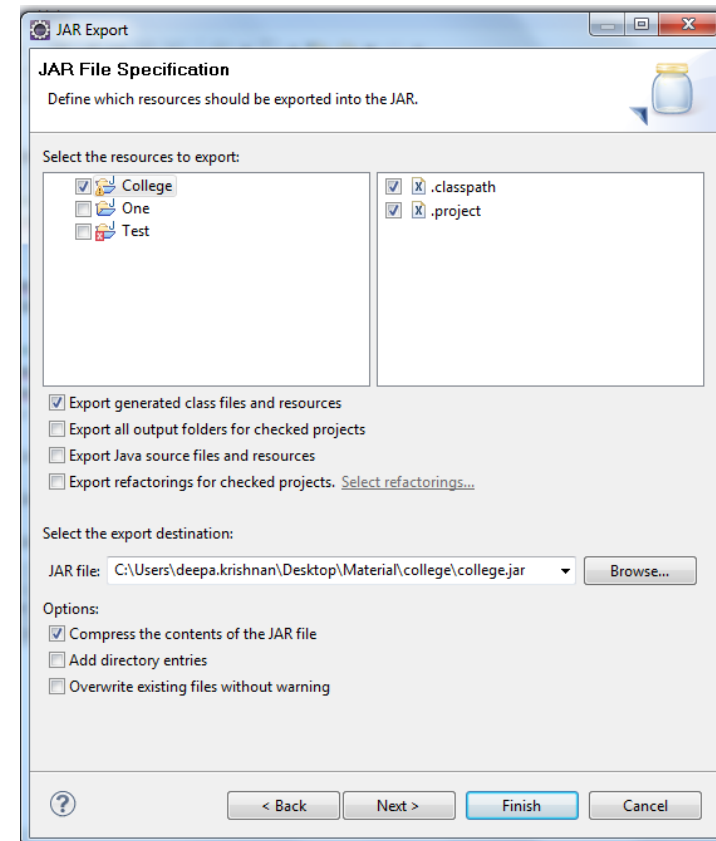
- Copy student and teacher package folders into D:\test.
- Run the following command
 - D:\packages >**jar -cvf college.jar *.class**
- This **jar** file can now be copied and placed anywhere. Copy this file to any of your favorite location.
- Close the command prompt session.
- Set the classpath to
 - **set classpath =%classpath%;yourpath\college.jar**
- Compile and Execute **Tester** class.

Activity: Creating jar in eclipse

- Set up:
 - Make sure that you have a java project called 'College' which has 2 packages student and teacher and the associated classes]
 - Create a new java project called Test and copy the Tester class into it. You should see a lot of errors.
- Right click on the College project and select Export.
- Expand the Java node and select JAR file. Click Next.

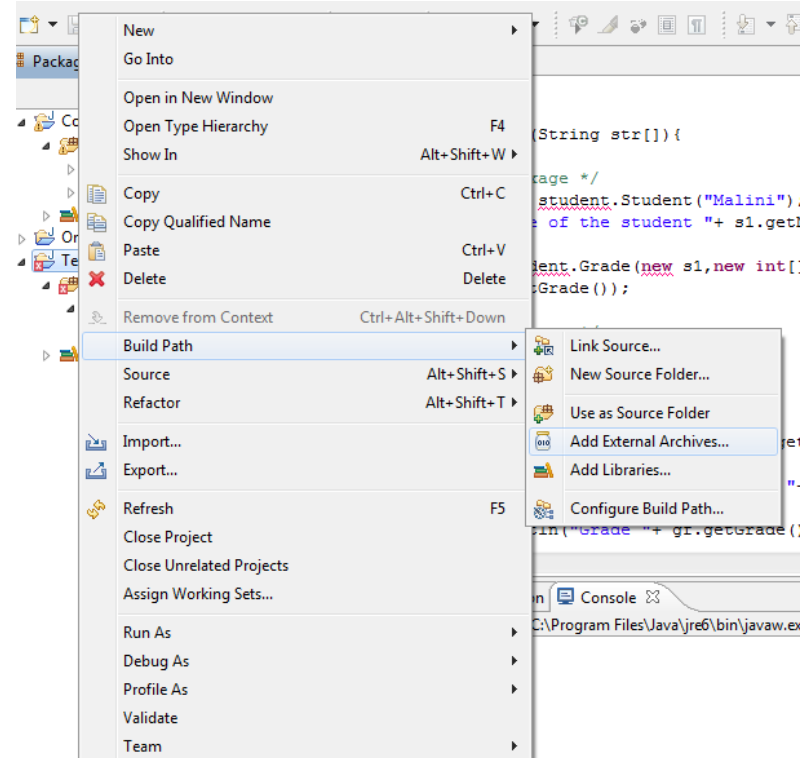


- In the JAR File Specification page, select College project.
- In the Select the export destination field, either type or click Browse to select a location for the JAR file.
- Click Finish to create the JAR file.
- Browse on the location that you specified to find the jar file



Activity: Setting classpath to jar in eclipse

- Right click on the Test project and select Build Path → Add External Libraries.
- Browse and Select college.jar file.
- All the errors must disappear.
- Execute the Tester class.



Imports

- Java provides an easy way to access the 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 is accessed using *ClassName.staticMemberName*. This name also sometimes could be very long. Instead imports could be used such that only static member name could 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.");
}
```

Example: importing classes

```
import student.*;  
import teacher.*;
```

```
public class Tester1{  
    public static void main(String str[]){
```

```
        /*accessing student package */
```

```
        Student s1= new Student("Malini");
```

```
        System.out.println("Name of the student "+  
s1.getName()+" (" + s1.getRegNo()+" )");
```

```
        Grade g=new Grade(s1,new int[]{ 80,85, 91,86,  
82});
```

```
        System.out.println(g.getGrade());
```



```
/*accessing teacher package */
```

```
Teacher f=new Teacher("Tom");
```

```
Grade gf= new Grade(f,s1,"001",Grade.B);
```

```
System.out.println("Teacher Name "+
```

```
gf.getFaculty().getName()+"-
```

```
"+gf.getFaculty().getId());
```

```
System.out.println("Student who graded "+
```

```
gf.getStudent().getName());
```

```
System.out.println("Grade "+ gf.getGrade());
```

```
}}
```

*The program does not compile. Can you figure out why ?
How will you solve it ?*

Static Imports

- Makes the static members of a class available to code 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. Recall “Dynamic Linking feature of java”.
- Only the classes whose members are invoked in some way are in the source code will be loaded.
- Hence both using `*` or explicitly specifying class, will not impact class loading.
- One advantage of using explicitly specifying class name would be for better code comprehension.

Default Access Specifier

- One of the advantages of using package is also access control of classes. This is done through default access specifier, sometimes also called package access specifier.
- Classes or members having package access specifier can be accessed only by the classes belonging to the same package.
- There is no keyword for default access specifier.
- If no access specifier is specified for a class or member then by default it is assumed to be package access specifier or default access specifier.

Accessing members with different access specifier

package student;

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

```
class StudentTest{  
    void test1()  
    public void test2()  
    private void test3()  
}
```

package teacher;

```
public class Teacher{  
    ..  
}
```

Recall

- Write down all modifiers (access specifiers and other keywords that you can put in front of a class or member like static) that we have covered so far.
- What modifiers are applicable a class?
- What modifiers are applicable a member?

Can you nest packages?

- For the need of better name and organization packages can be created such that classes can be placed inside multiple folders which are in hierarchy or nested.
- For instance, if all the classes related to college must be inside a college folder, then Student class has to be inside student folder which in turn should be inside college folder and then Teacher class has to be inside teacher folder which in turn should be inside college folder, then the package statement has to be

```
package college.student;
```

```
public class Student{...}
```

```
package college.teacher;
```

```
public class Teacher{...}
```


Packages - Not truly nested

- Although packages appear to be hierarchical or nested with respect to folder, they are not.

- This means that

```
import college.*;
```

does not import **student** and **teacher** package.

- It **imports** only the classes inside **college** folder/package (if there are any)
- To import both **student** and **teacher** package, 2 import statements are necessary
 - **import college.student.*;**
 - **import college.teacher.*;**

Packages - Not truly nested

```
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;}
```

- If class C is in one of the above packages then the import statement in to which the package belongs can be omitted.

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 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.

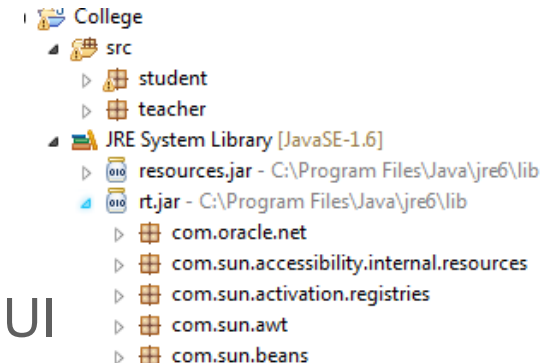
Test your understanding

Can you find out if there is there any problem with this code?

```
package a;  
  
class String{}  
  
public class A{  
  
    public static void main(String str[])  
  
    {  
  
        System.out.println(str[0]) ;  
  
    }  
  
}
```

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.



Which class in JSE have we looked at so far?

Let us look a 2 more classes in JSE–System and Scanner class

System class

- The **System** class contains has classes that allow working with standard input, standard output, and error output streams.
- It cannot be instantiated.
- All the members of the **System** class are **static**.
- Apart from console input and output streams, it also has methods to
 - access to externally defined properties and environment variables;
 - to load files and libraries
 - for quickly copying a portion of an array
 - request for garbage collector to run

System class members

- **static final InputStream in**

- The standard input stream.

We have already been using many of these.

- **static final PrintStream out**

- The standard output stream.

System.in will be dealt with in Scanner class

- **static final PrintStream err**

- The standard error output stream

- **public static void gc()**

- Request JVM to run garbage collector to recycle unused reference.

- **public static void exit(int status)**

- Terminates the currently running JVM. A nonzero status code indicates abnormal termination.

Example : Array copy (System)

- `static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)`
- Example:

```
public class ListFlowers{  
    public static void main (String[] args) {  
        String[] copyString = new String[3];  
        System.arraycopy(args, 1, copyString, 0, 3);  
        for(String s:copyString)  
            System.out.print(s+ " ");  
    }  
}
```

→ `java ListFlowers List Rose Lily Lotus`

→ Prints: Rose Lily Lotus

System class- loading library example

- `static void loadLibrary(String libname)`
- This method is used when you want to load native libraries.
- The example below loads HelloNative.dll (if run from Windows).
- Note that since libraries are loaded only once, the `loadLibrary` is in static block.
- `native` modifier denotes that the method is defined in native language (C or C++).

```
public class HelloNative{  
    public native static void greeting() ;  
    static{  
        System.loadLibrary("HelloNative") ;  
    }  
    public static void main(String str[]){  
        HelloNative.greeting() ;}}}
```

We will not be looking at JNI. This example is provided just to understand loadLibrary() method

System class- working with system properties

- `static String setProperty(String key, String value)`
- `static String getProperty(String key)`
- Property files are used to maintain system and application configuration information like user name, password etc.
Properties class has methods that allow working with property file.
- The Java platform itself uses a **Properties** object to maintain its own configuration.

Example

```
public class Test{  
    public static void main (String[] args) {  
        System.out.println( System.getProperty("java.home")) ;  
        System.out.println(System.getProperty("java.version")) ;  
        System.out.println(System.getProperty("java.class.path")) ;  
        System.out.println( System.getProperty("java.vendor")) ;  
        System.out.println( System.getProperty("os.name")) ;  
        System.out.println( System.getProperty("os.version")) ;  
    }  
}
```

Prints

```
C:\Program Files\Java\jre6  
1.6.0_24  
D:\workspace\One\bin  
Sun Microsystems Inc.  
Windows 7  
6.1
```

*Are you able make out what each
of these properties are?*

Scanner

- *We know how to print on the console but how to read inputs from the console?*
- *Before Java 5, reading console inputs was not very easy. From Java 5 onwards, Thanks to the Scanner class, reading input from console is simplified.*
- A **Scanner** class is used to parse text and primitive type from any input stream.
- It breaks its input into tokens based on delimiter(default is whitespace).
- The **Scanner** class is a class in `java.util` package.

Methods in Scanner

- `int nextInt()`
- `long nextLong()`
- `float nextFloat()`
- `double nextDouble()`
- `String next()`
- `String nextLine()`

Examples

- To read an **int** from console:
 - `Scanner sc = new Scanner(System.in);`
`int i = sc.nextInt();`
- To read a **string** from console:
 - `Scanner sc = new Scanner(System.in);`
`String = sc.next();`

Student class complete

```
package student;

public class Student{
    private String name;
    private int regNo;
    private String degreeName;
    private int currentSemester;
    private static int gRegNo ;
    private static int generateRegno() {
        gRegNo++;
        return gRegNo;
    }
    public static int getGRegNo() {
        return gRegNo;
    }
}
```

```

public Student(String nm, String d) {
    this(nm) ;
    setDegreeName(d) ;
    setCurrentSemester(1) ;}

public Student(String nm) {
    setName(nm) ;
    regNo=generateRegno() ;
    setCurrentSemester(1) ;}

public String getName(){return properCase(name) ;}
private String properCase(String nm) {
    nm=(nm.charAt(0)+" ").toUpperCase()+nm.substring(1) ;
    return nm;
}

```



```
public void setName(String nm){
    if(nm==null)
        System.out.println("Name cannot be null");
    else name=nm;}

public int getRegNo(){ return regNo; }
public String getDegreeName(){ return degreeName; }

public void setDegreeName(String dnm){
    if(dnm==null)
        System.out.println("Degree name cannot be null");
    else degreeName=dnm;}

public int getCurrentSemester(){
    return currentSemester;}

public void setCurrentSemester(int i){
    if((i<0) || (i<currentSemester ))
        System.out.println("Invalid value for current
semester");
    else currentSemester=i;}}
```