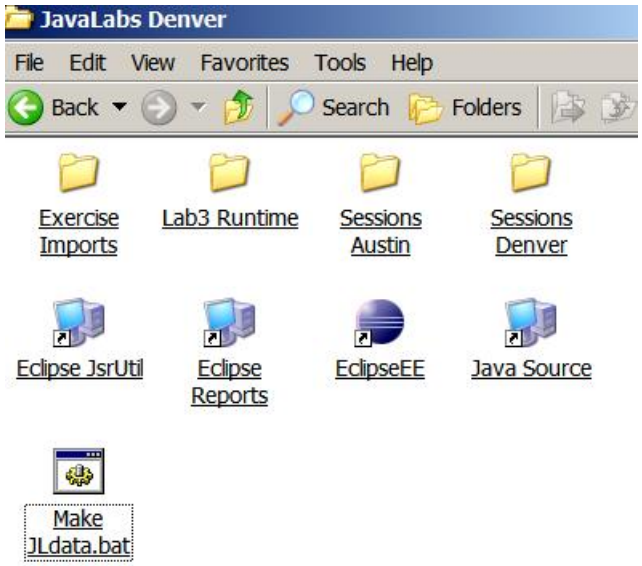
Agenda

- Inheritance and relationships
- Lab Exercises (see examples of polymorphic overloading, encapsulation, and inheritance)
- Make changes to programs to illustrate benefits of Inheritance and encapsulation.
- Abstraction and interfaces
- Lab Exercises

Steps I took at home to create Eclipse Projects from plain source
(C:\jsr\Java\source).

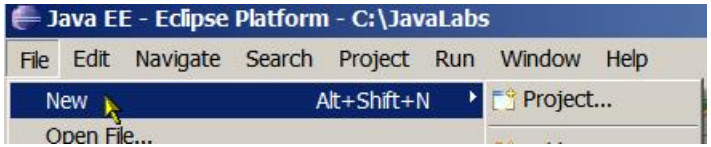
1) Create shortcut to old java source directory
2) Build JsrUtil Project and create shortcut to it's src.
3) Copy Programs into JsrUtil\src that you want in this project.
4) Refresh, if missing any dependencies, then copy it.
5-7) Repeat 1-3 but with Reports as project name.
8) Add JsrUtil Project to build path.

Now:

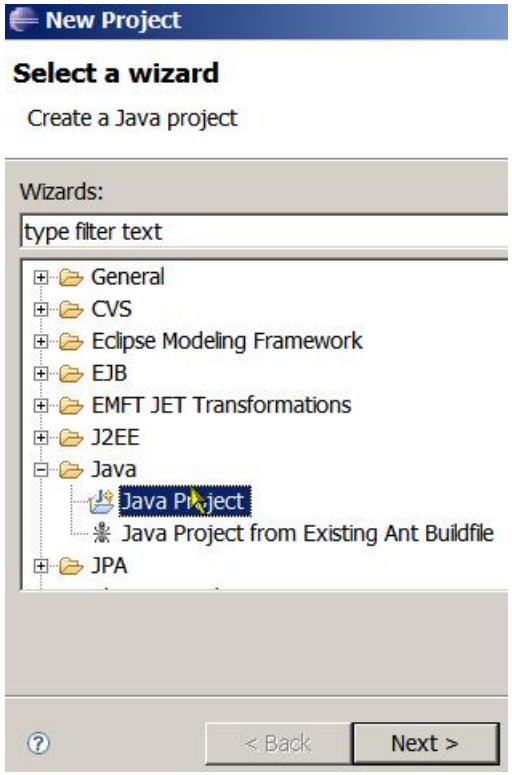


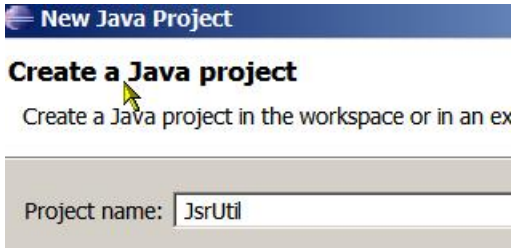
Open JavaLabs Denver
MakeJLdata.bat (creates C:\JavaLabs empty folder).
Click on Eclipse Icon.

X (close) Welcome screen



Create New Project


New Project
Select a wizard
Create a Java project
Wizards:
type filter text
General
CVS
Eclipse Modeling Framework
EJB
EMFT JET Transformations
J2EE
Java
Java Project
Java Project from Existing Ant Buildfile
JPA
< Back
Next >

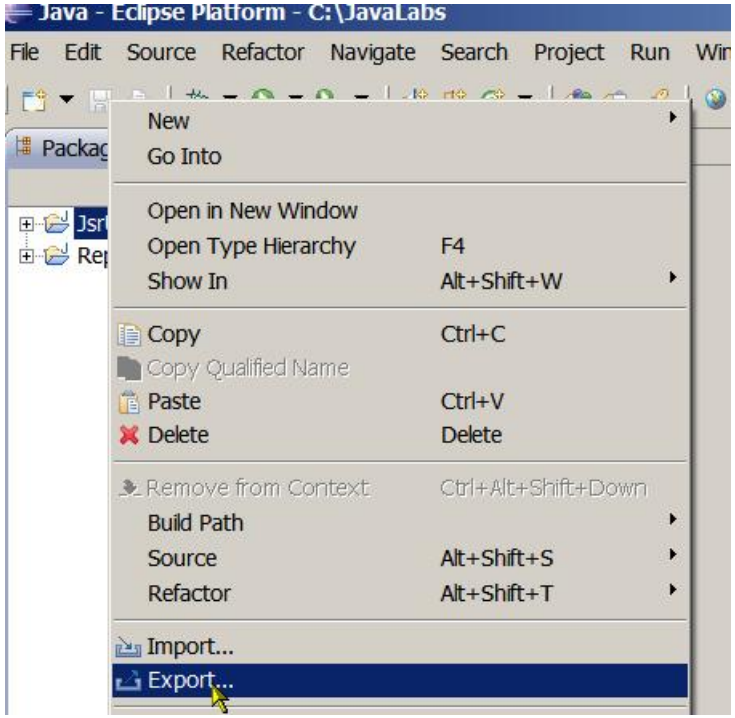

New Java Project
Create a Java project
Create a Java project in the workspace or in an ex
Project name:
JsrUtil

And click Finish

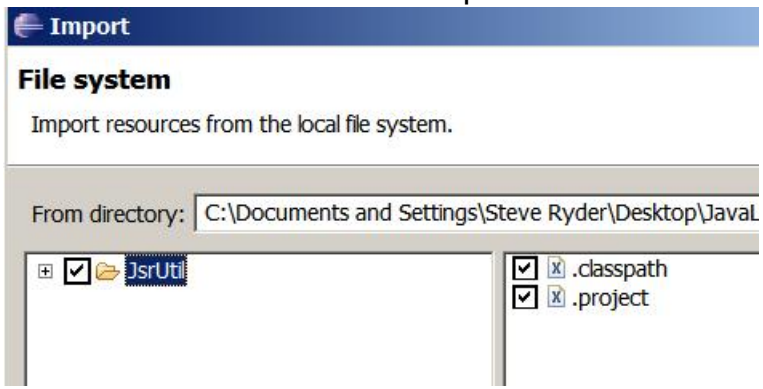
Check and click Yes.

Repeat Above for “Reports” Project

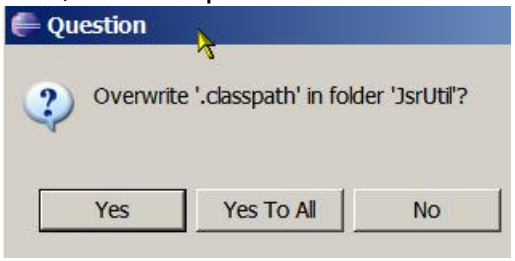
Now Import JsrUtil from File System



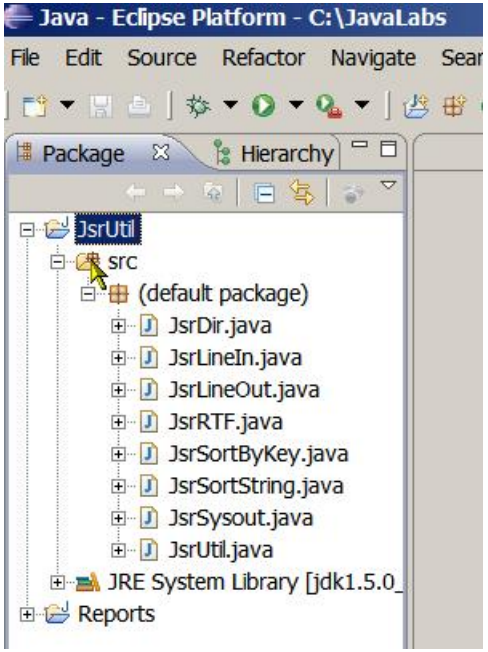
Navigating from Desktop down to JavaLabs Denver to Exercise Imports to JsrUtil



Select Resouce (check the box infrom of JsrUtil, note Export location is filled in as JsrUtil) and click Finish.

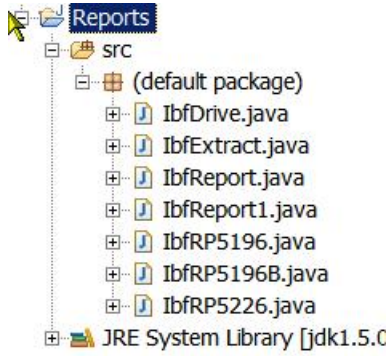


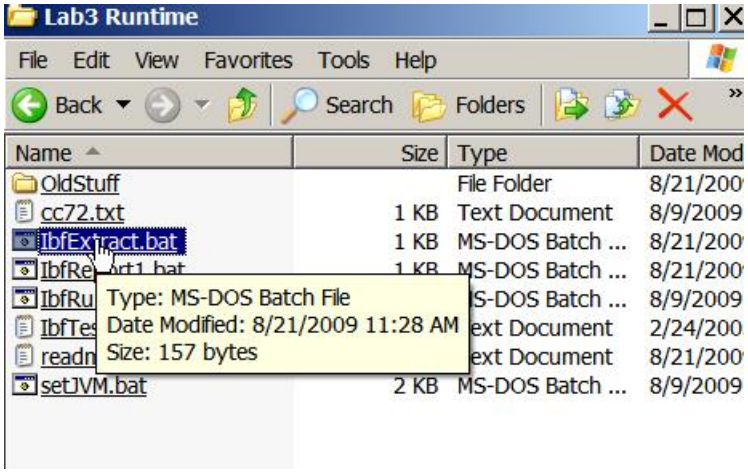
Click Yes To All



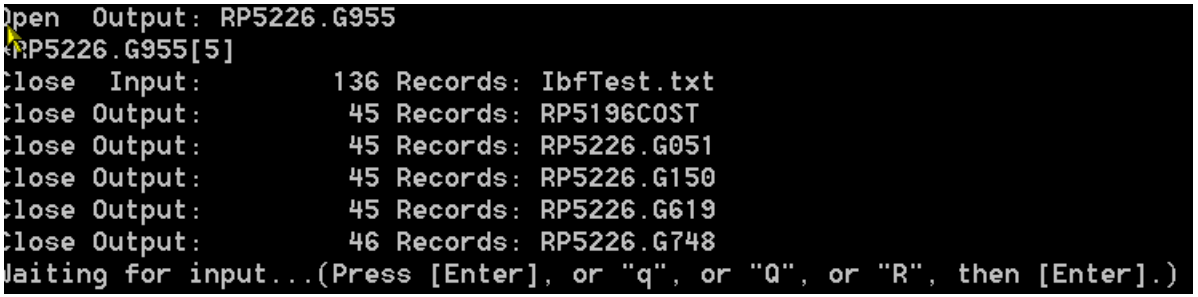
Expand JsrUtil

Repeat above for Reports...

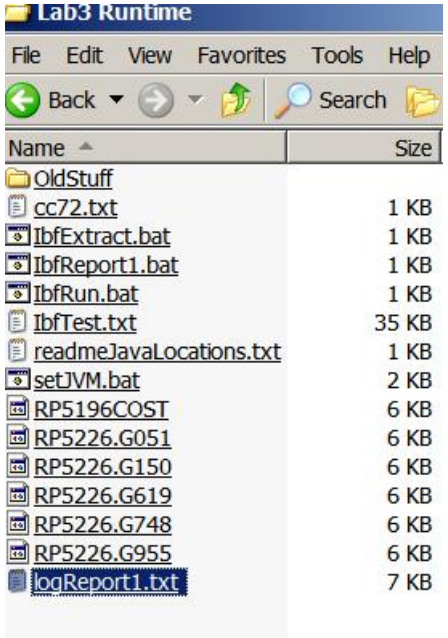




Navigate to Lab3 Runtime and click on IbfExtract.bat



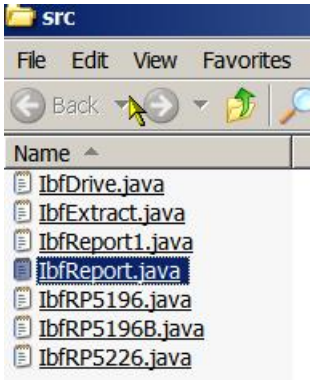
Press Enter and notice extra files in directory...
Click IPFReport1.bat, pressing enter at the pause.



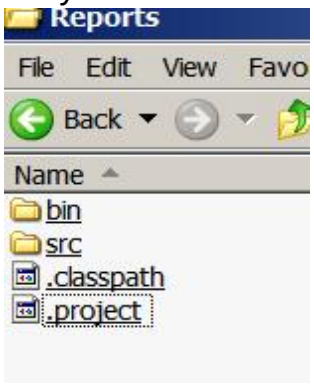
Click logReport1.txt This is a Xerox format file.
Lets look at how we were able to run Eclipse classes, while developing, without having to Export to a JAR file, which you would want to do for production.

The setJVM.bat file sets the JVM variable. The set classloc sets the location inside the Eclipse workspace where the class files are compiled. This is not visible to you from Eclipse.

If you navigate to JavaLabs Denver, and click the Eclipse Reports icon you will open an Explorer window into the src directory for the Reports project.



Click the up Arrow and you will see the actual directory structure:



Click the bin folder and you will see the classes.

The above code is from a project that I did several years ago (convert a COBOL extract and report job stream running on zOS to run on an NT box). I had convinced my client up front that it would cost him less than the cost of a COBOL compiler for me to just do it in Java. I actually developed on a Windows 98 box and it ran without a hitch on the NT box AND it ran faster on the NT box. The speed advantage was because COBOL did not have the ability to dynamically create a variable number of output files based on user input. The data files consisted of discrete sub-sets of data (report type and region). The COBOL application just sorted the whole file one time. Then each report (about 20 different executions of 3 different reports in the production environment) read the WHOLE file! I wrote a simple extract that simply split the file into 20 separate files (a simple trick of creating a new instance of JsrLineOut for each report-type, region combination). Then each report sorted one file (1/20th of the original sort so it was sorted in memory), and "printed" the report.

- Examine JsrUtil for examples of overloading (a form of polymorphism) and useful ways to do things in Java to mimic COBOL behavior.

- Examine JsrSysout for examples of class and instance variables and how they might be useful.

- Examine JsrLineIn and JsrLineOut as a useful example of encapsulating I/O logic (especially the exception handling).

- Examine IbfExtract and file IbfTest.txt as an example of allocating multiple output files depending on contents of input file. Run IbfExtract.bat. Examine console (benefits of JsrSysout). Open logIbfExtract.txt

- Examine IbfReport (example of Super Class)

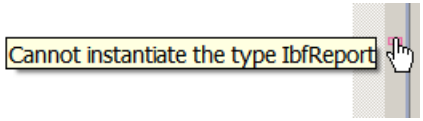
- Examine IbfReport1 (example of a sub-class)

- Run IbfReport1 using IPFReport1.bat

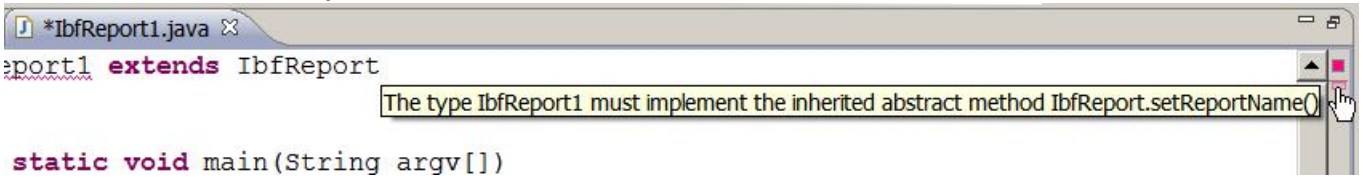
Your assignment:

- Modify IbfReport. This illustrates power of Abstract classes to force sub-classes to implement required methods. Make all the methods under // it is expected... abstract. Note they all get errors. Hint: move the keys = line up to the initial definition of keys...

- Now add abstract in front of class



- Oops, that is OK we will do that in IbfReport* class
- Remove the main method and there are no errors.
- There are no errors in IbfReport1 because all the methods are invoked. Remove one of the methods.



- Paste the removed method back.
- Run IbfReport1 again.
- Copy IbfReport and rename copy IbfReportRTF
- Change IbfReportRTF to use JsrRTF instead of JsrLineOut.

```
JsrRTF output    = new JsrRTF();
```

Also remove the xerox header line after output.setName:

```
output.setName(parm[2]);
//put133(nullChar+input.getLine().substring(1));
```

Hint: in newPage method, invoke setPage() method.

```
void newPage()
{
    pageNum++;
    if (pageNum > 1) output.setPage();
    out133(tl1);
```

Hint2: Look in the JavaDoc section of ExerciseImports in JavaLabs Denver and click on the index.html tab for methods in JsrRTF.

- Change IbfReport1 to extend IbfReportRTF and run again.

There is lots more you could do to “clean-up” IbfReportRTF. In the real world you would probably want a class ReportRTF that has the logic not specific to the Ibf Reports. Then IbfReport would extend ReportRTF and IbfReport1 would extend IbfReport as it does now...

Questions?