JAR Files

Handout by Eric Roberts, Mehran Sahami, and Brandon Burr

Now that you've written all these wonderful programs, wouldn't it be great if you could package them up and send them to your mom, dad, friends, and pets so that she could see what you've done? Because we here in CS106A feel that no parent should be spared the joy of sitting through a simplified version of a game he or she has undoubtedly played a million times before, here's a short guide to making an executable JAR file in Eclipse!

If Your Program Uses No External JAR files

Eclipse makes it easy to package your code up into a JAR file that you can double-click to run, especially for those programs that aren't using other JAR files. In other words, if your program is using the ACM libraries (or some other non-standard Java libraries), you'll have to do the slightly more complicated method that uses a manifest file. Both examples are described below.

Using the ACM libraries, Step 1

Our programs that have used the ACM libraries have started running via the `public void run()` method. We changed Eclipse to allow this, to make things easier for you. But in reality, a Java program needs to start at a particular method in a class, the `public static void main(String[] args)` method. If your program uses the ACM libraries, you'll need to edit your code to have a `main()`. You can do this easily by simply adding the following code to the class that has the `public void run()`, substituting the name of that class for `'NameSurfer'` below.

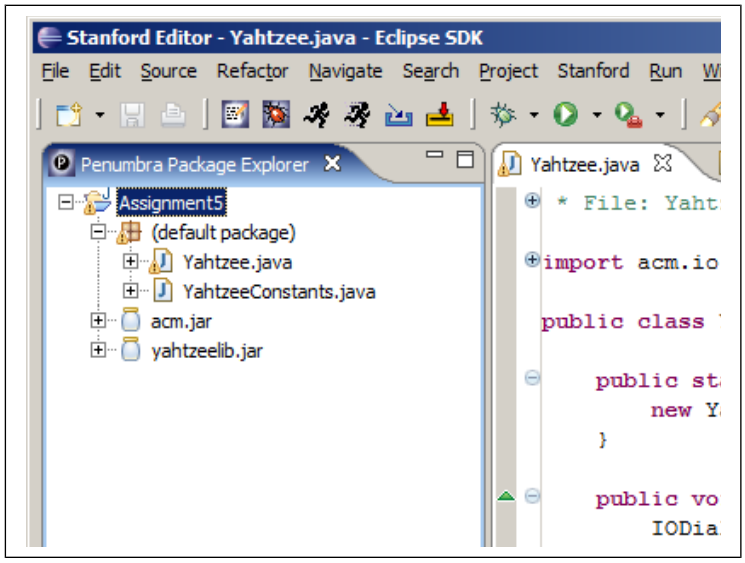
```
public static void main(String[] args) {
      new NameSurfer().start(args);
}
```

If you remember at the beginning of the quarter, we said that you needed the special Stanford version of Eclipse to run your programs, this is because of the edit (mentioned above) that we made to Eclipse. But if you add this `main()` method your program should run fine in any Java compiler.

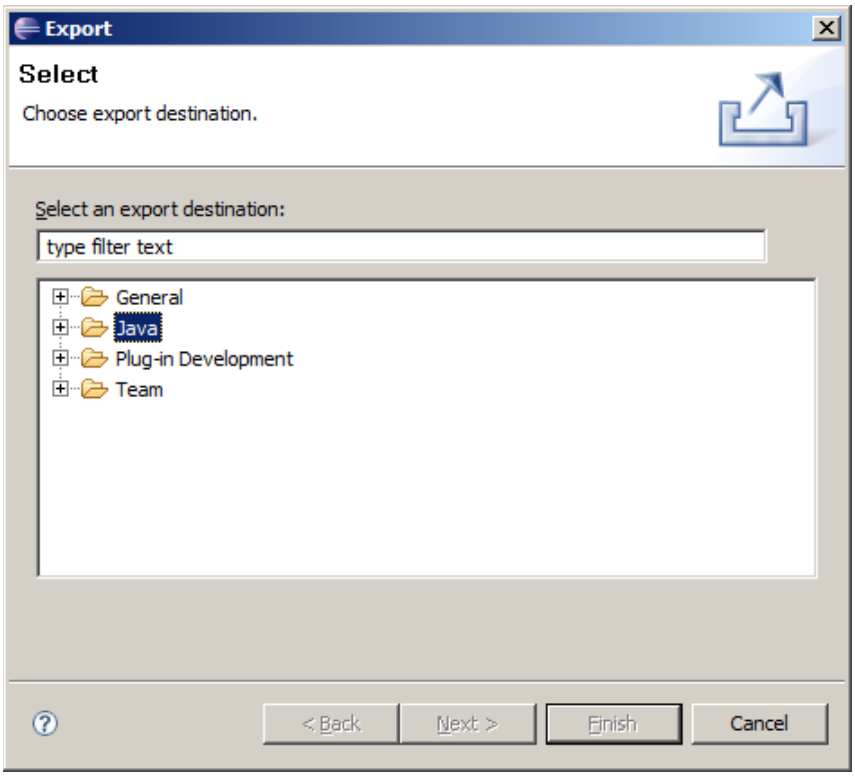
Using the ACM Libraries (Or Using Any External JAR Files), Step 2

Now that we have a normal running Java program, let's package it up into a JAR file. A JAR file is simple a *J*ava *AR*chive – a file that contains a set of Java class files as well as potentially other files that will be used by the program. One important issue to point out here is if we need to specify other libraries that our program will need to reference. The easiest way do this in Eclipse is by using a *manifest* file when creating your JAR file. The manifest file allows you to specify things like which is the `main` class to run when the JAR file is double-clicked, or what the external libraries are, or even security information about the JAR file. If you aren't using other JAR files, you don't need to use the manifest file. Eclipse provides a straightforward way of exporting your program. In either case, you can create a JAR file as follows.

Go through the process of *Exporting* your project to a JAR file as shown below. In Eclipse, highlight (click on the name of) the project you wish to create a JAR file from:

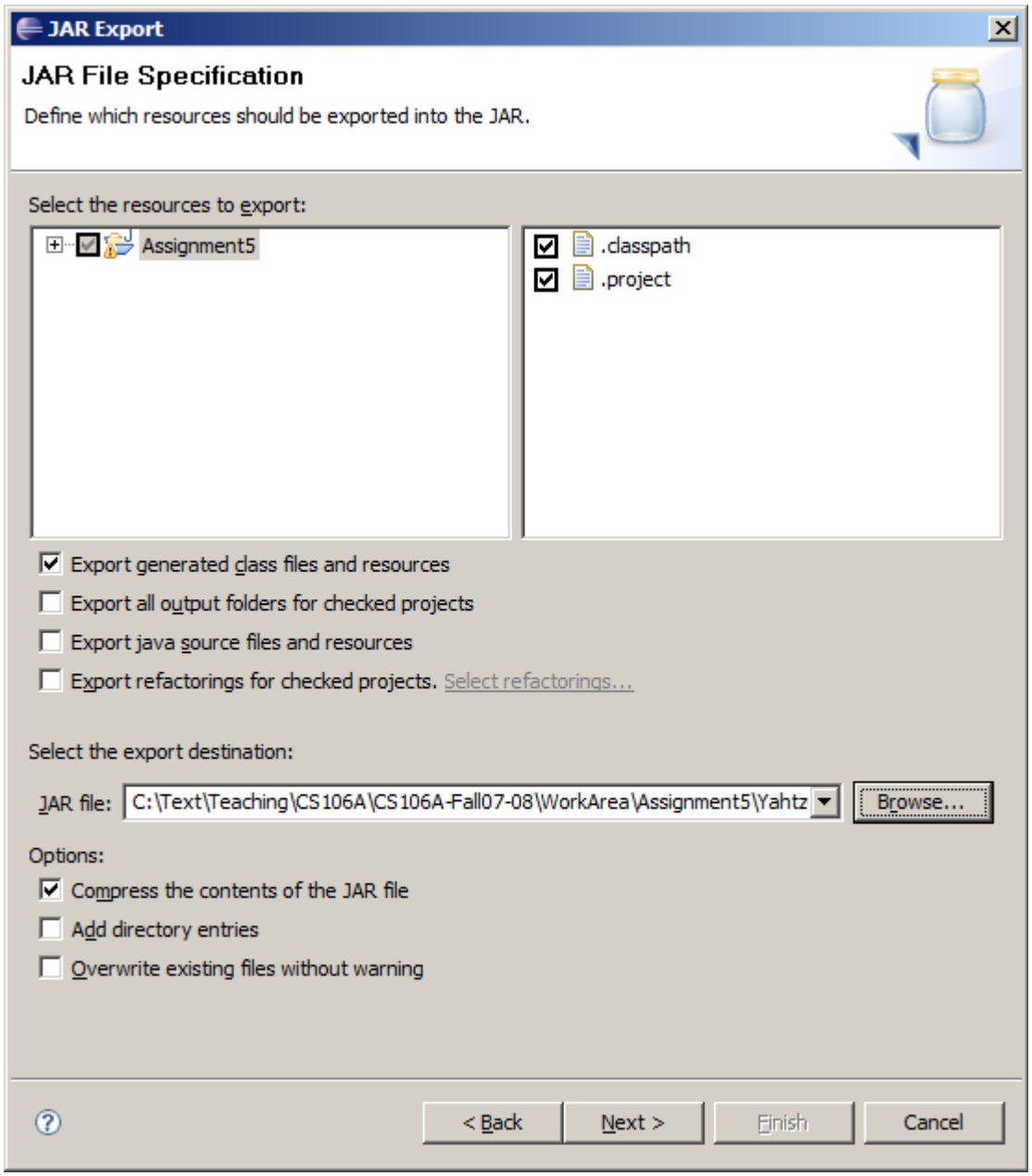


Go to the `File` menu and select the `Export`... command. This should give you something that looks like the following window:



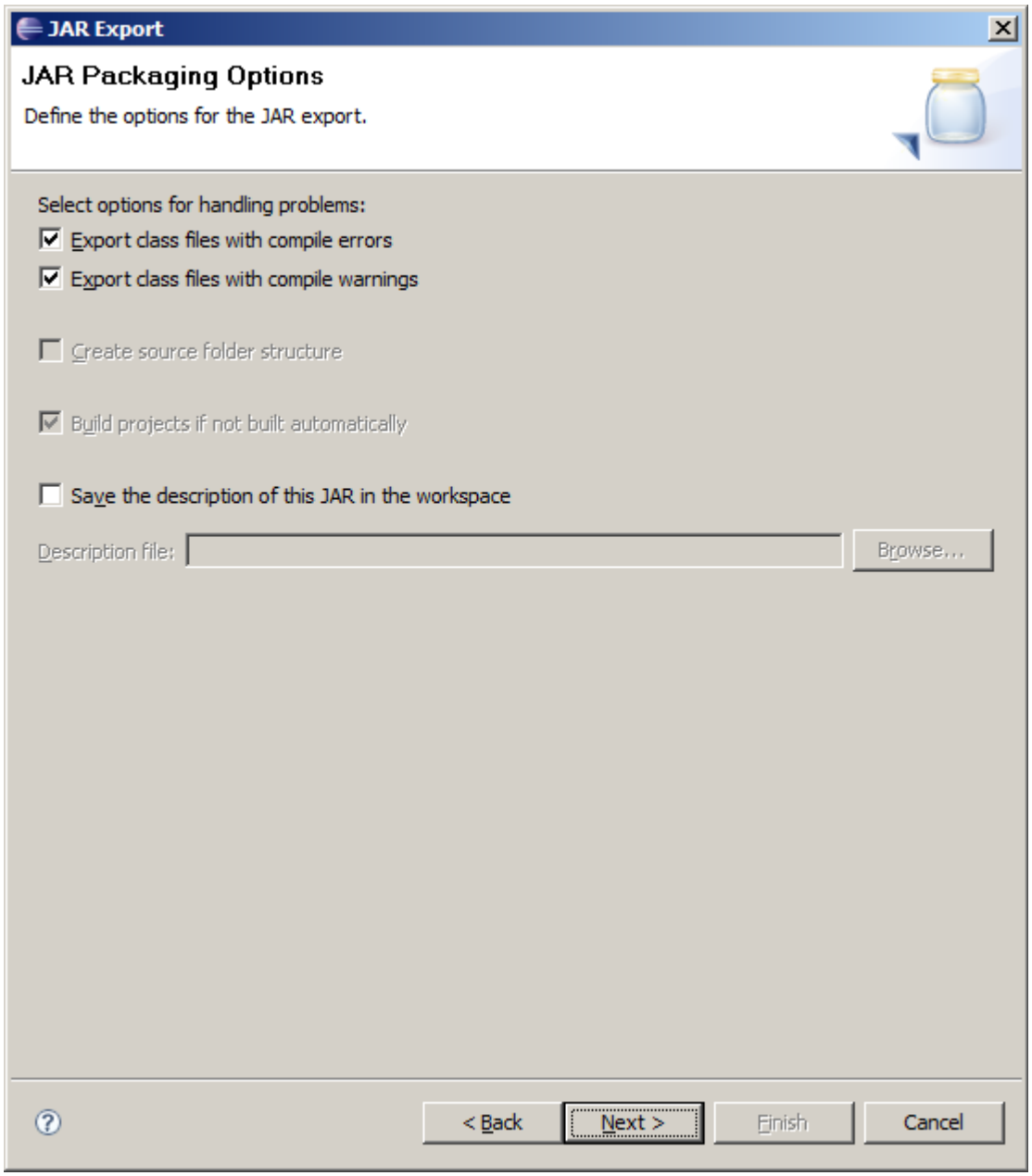
Double click on `Java` to expand the list of options, and then click on the option `JAR file` to highlight it. Then, hit the `'Next >'` button in the window.

You will see the JAR Export window:



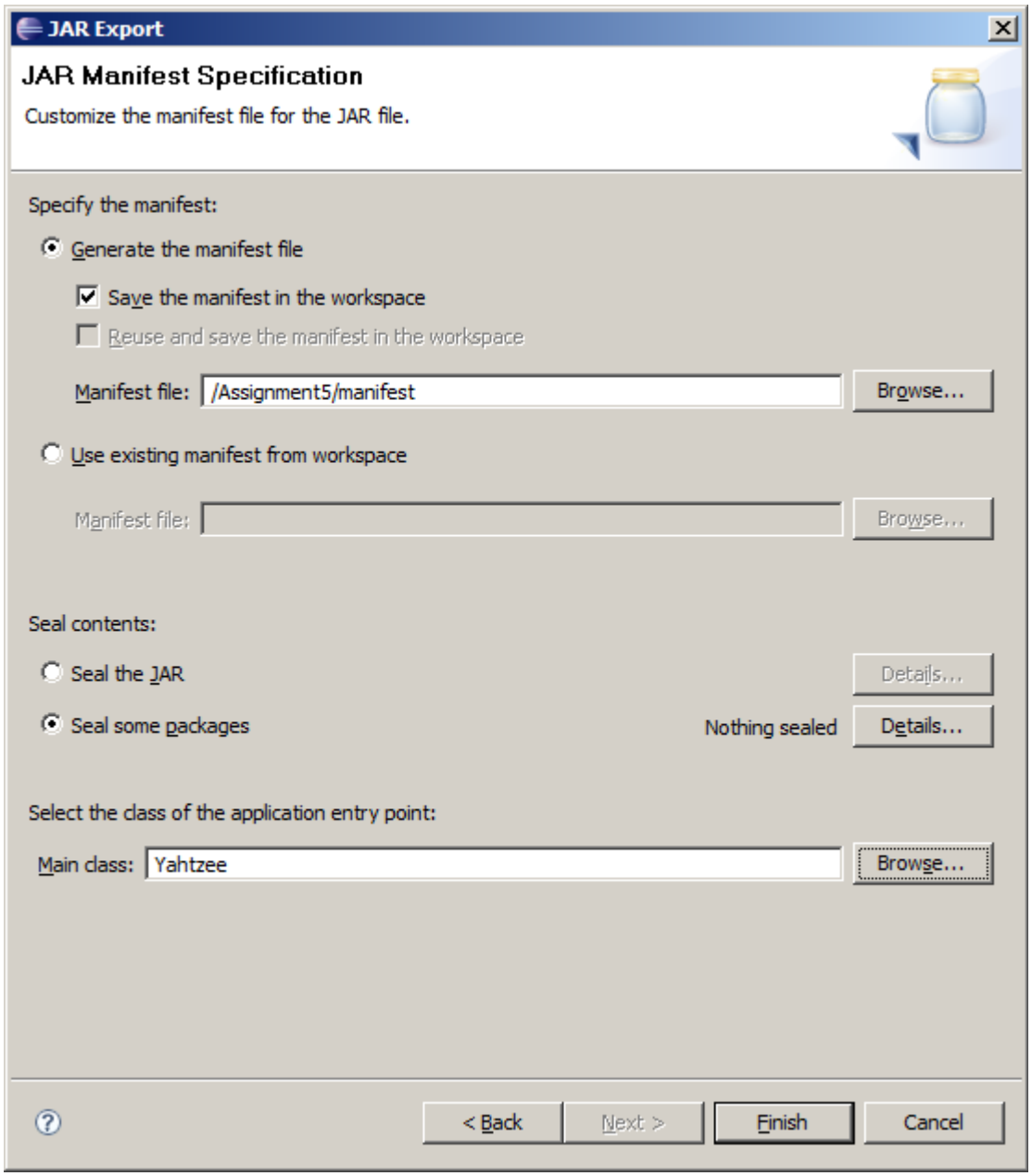
Click on the name of your project (**`Assignment5`**, in this case) to make sure the **`(default package)`** is selected, and select the destination of the JAR file using the **`'Browse..."`** button. Then hit **`'Next'`**.

You will then see the following window:



This screen isn't too important for our purposes. Just hit the `'Next >'` button again.

You will then see the following window:



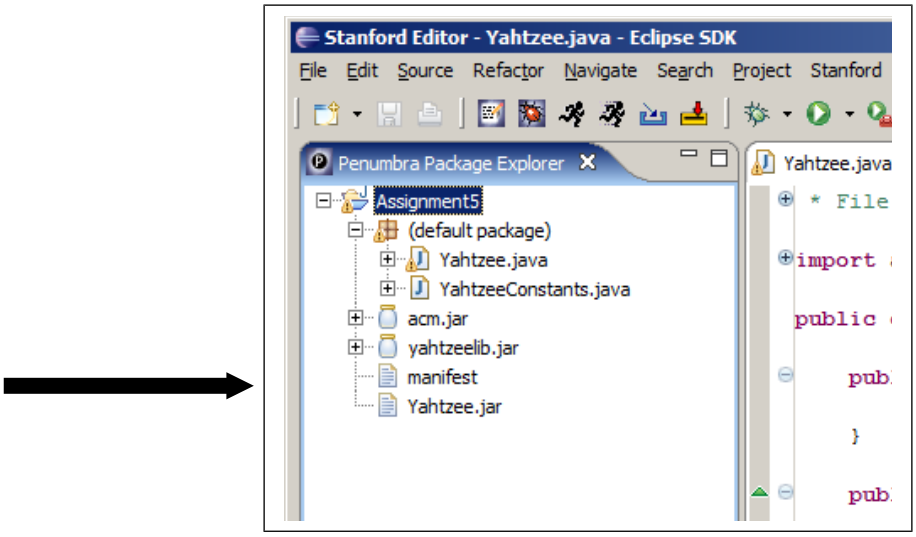
Okay, now here's where the important stuff happens. First, near the bottom of the window, select the **`Main class`** using the **`'Browse...'`** button. The main class (**`Yahtzee`**, in this case) should show up in the list if you correctly added the **`main()`** method described above.

If your program doesn't reference other JAR files (i.e., it does not use the ACM libraries or any other libraries), that's it. You're done! You don't need to worry about the manifest file stuff. Just hit the **`'Finish'`** button, and go double-click the JAR file you just created. Make sure you see the last section of this handout on data files and distribution, though.

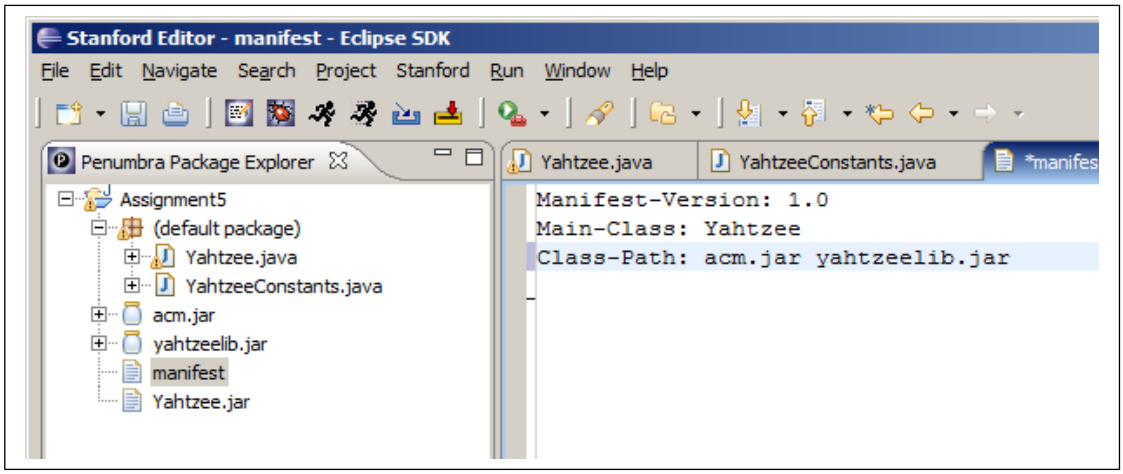
If you do need to reference other JAR files (i.e., like the ACM libraries), then you need to create a manifest file. To do this, we will go through this exporting process twice. The first time is to generate a manifest file, and the second time is to use that file when exporting our program. So, from here, make

sure **'Generate the manifest file'** radio button near the top of the window is selected, and that check box for **'Save the manifest in the workspace'** is checked. Use the **'Browse…'** button associated with the **'Manifest file'** text box to select the destination of the manifest file. Click the **Assignment5** folder (or whatever your project is named), and then in the box type in the name **"manifest"**. This screen should look something like the image above when you're done (i.e., the Manifest file will likely be **'/Assignment5/manifest'**). Now hit the **'Finish'** button.

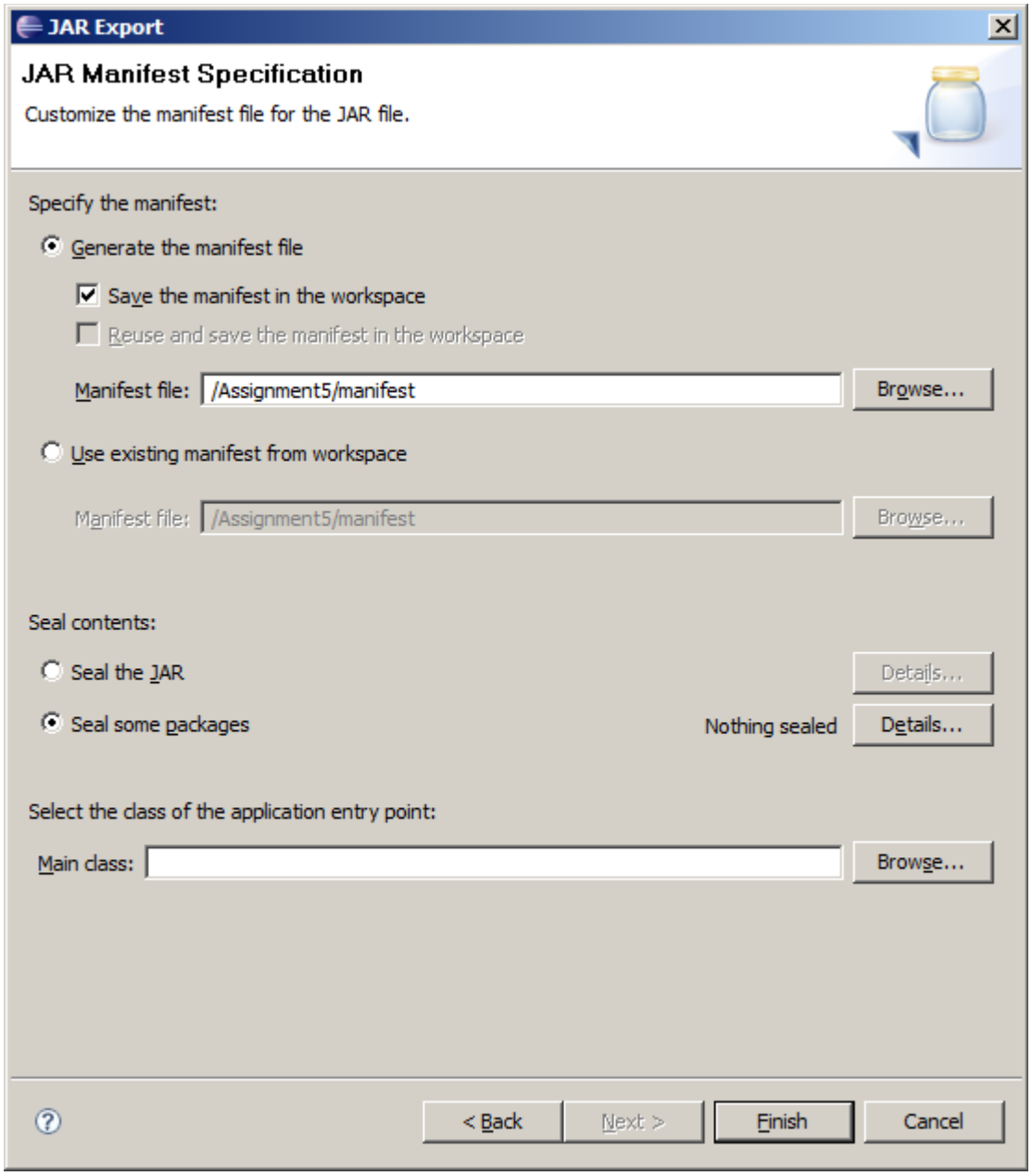
You should see the manifest file show up in the package explorer:



If you double click on the **manifest** file, you should see its contents. You need to edit the manifest file to add the line **"Class-Path: "** followed by the names of all the JAR files that this program uses, separated by spaces. In this case, that would include **acm.jar** and **yahtzeelib.jar**. When you're done the manifest file will look something like this:

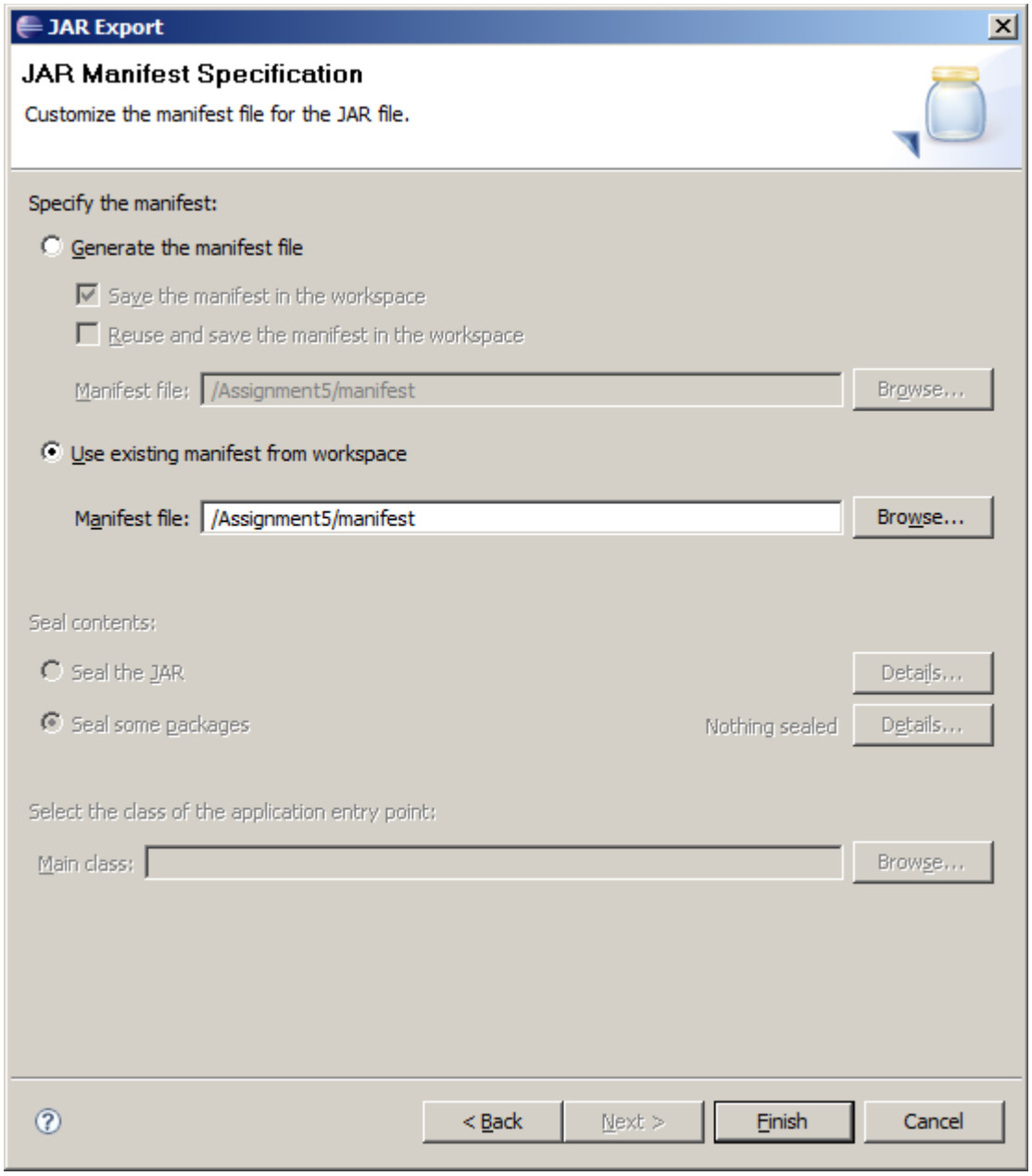


Make sure to save the updated manifest file. Now that we have this manifest file, repeat the entire above process of exporting a JAR file (i.e., click on your project name, pick **Export...** from the file menu, select the JAR file option for exporting, etc.). However, this time you will do something different when you get to the last window (shown below):



When you get here, make sure to click the radio button for **"Use existing manifest from workspace".**

You should then have a screen that looks like this:



Now, hit **"Finish"** button. Eclipse will use the manifest file we just created previously to make our **yahtzee.jar**. If it asks you to overwrite the old **yahtzee.jar**, just say "Yes".

We're almost there!

Distributing Your Program

Now you have your `yahtzee.jar` file, containing your Yahtzee code, but you can't simply send the `yahtzee.jar` to your friends. This jar doesn't contain the code in the other two JAR files (`acm.jar` and `yahtzeelib.jar`), nor does it contain any data files your program might use (text files with data, or even sounds or images). What you'll want to do is create a new folder, place your `yahtzee.jar` file in it, along with any other JAR files your program uses (like `acm.jar` and any other the ones you added to the `manifest` file) and data files you use. Once you have all of these files in a single folder, you should be able to just double-click your `yahtzee.jar` file, and have it run your application. You will need to distribute this *entire* folder to anyone that you want to share your program with. Usually the easiest way is to just zip the folder up into a single file, and then email that – just make sure that your friends know how to unzip the file!

A Final Note Regarding the Java Runtime Environment (JRE)

Just one final note about distributing your JAR files to friends: anyone who has your JAR files will need to have the Java Runtime Environment (JRE) installed on their computer in order to be able to run Java applications. In fact, you may have installed the JRE on your own computer at the beginning of CS106A so that you could work with Java. Some, but not all, computers come pre-installed with the JRE. We just wanted to point this out in case you pass along the JAR files for your program to someone who may not have the JRE installed and was having problems trying to run your program as a result.