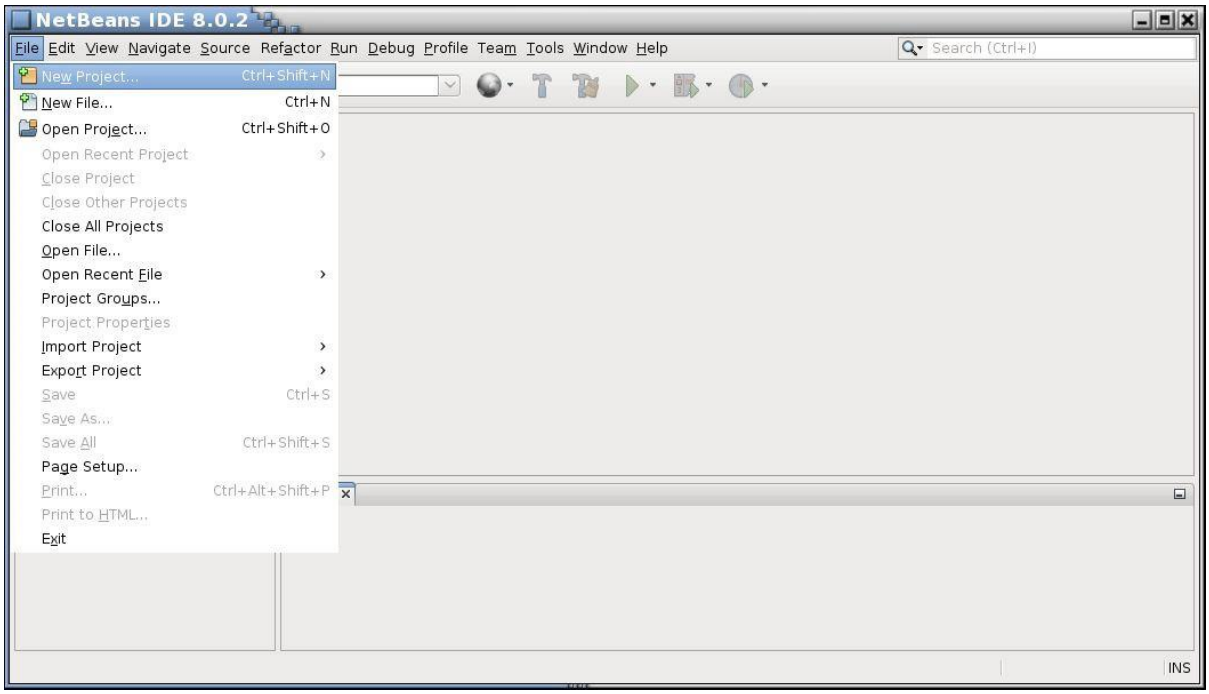
Pre-Lab Exercise: Lab Instruction

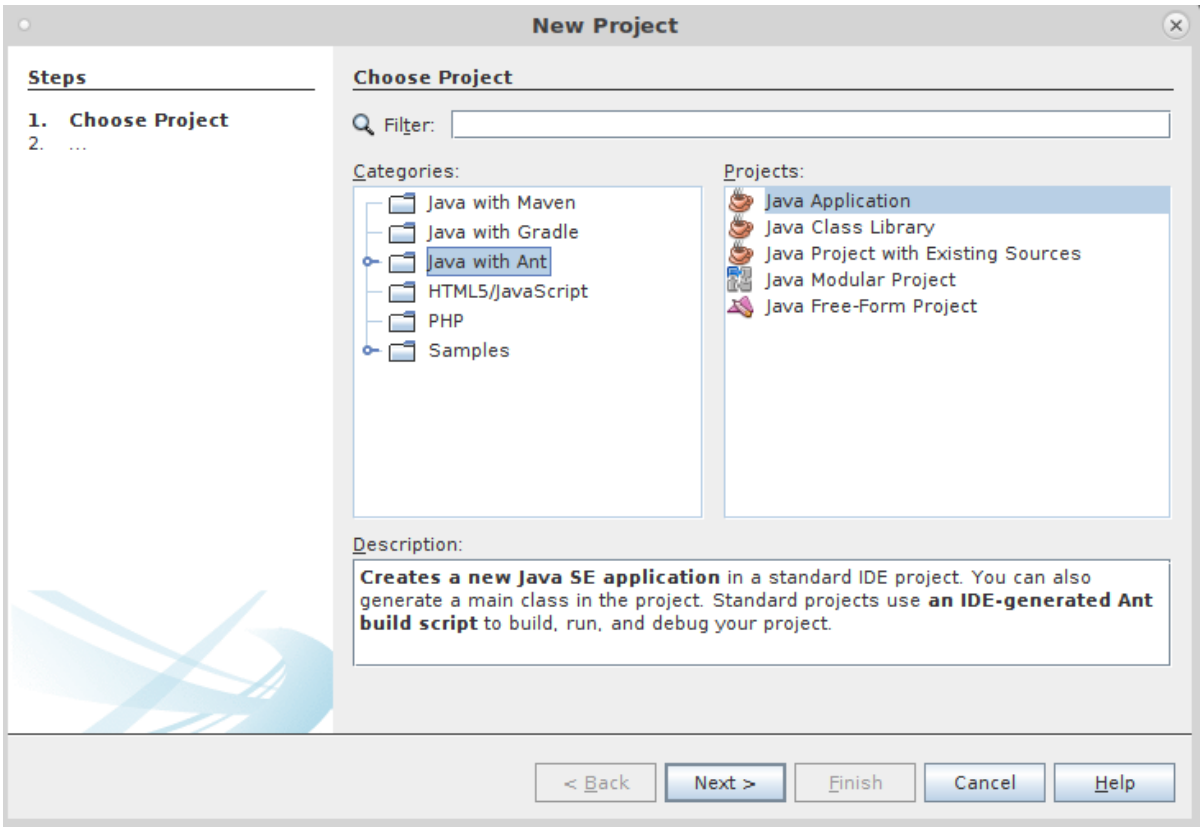
These instructions assume you are using a Linux workstation in UTS Building 11. Steps may differ slightly if you are using Windows or MacOS.

Importing code into NetBeans

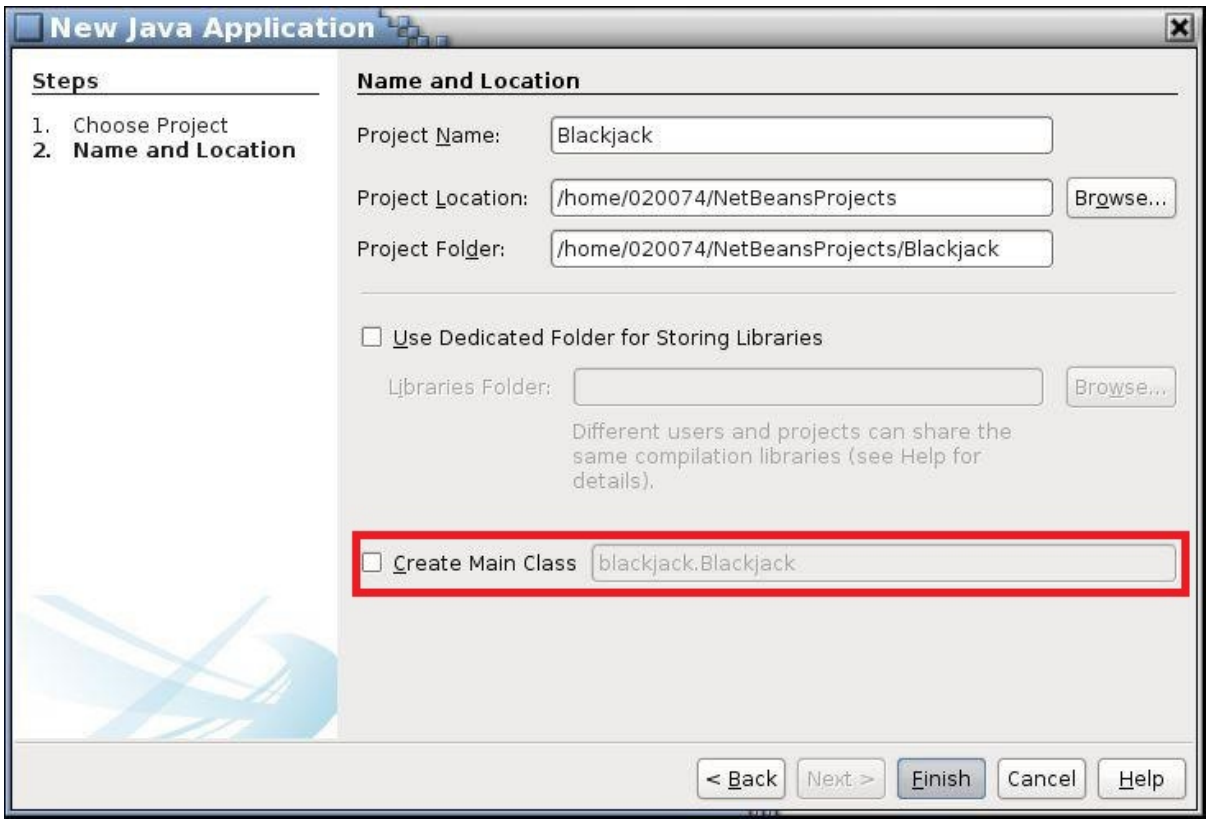
From within NetBeans, create a new project.



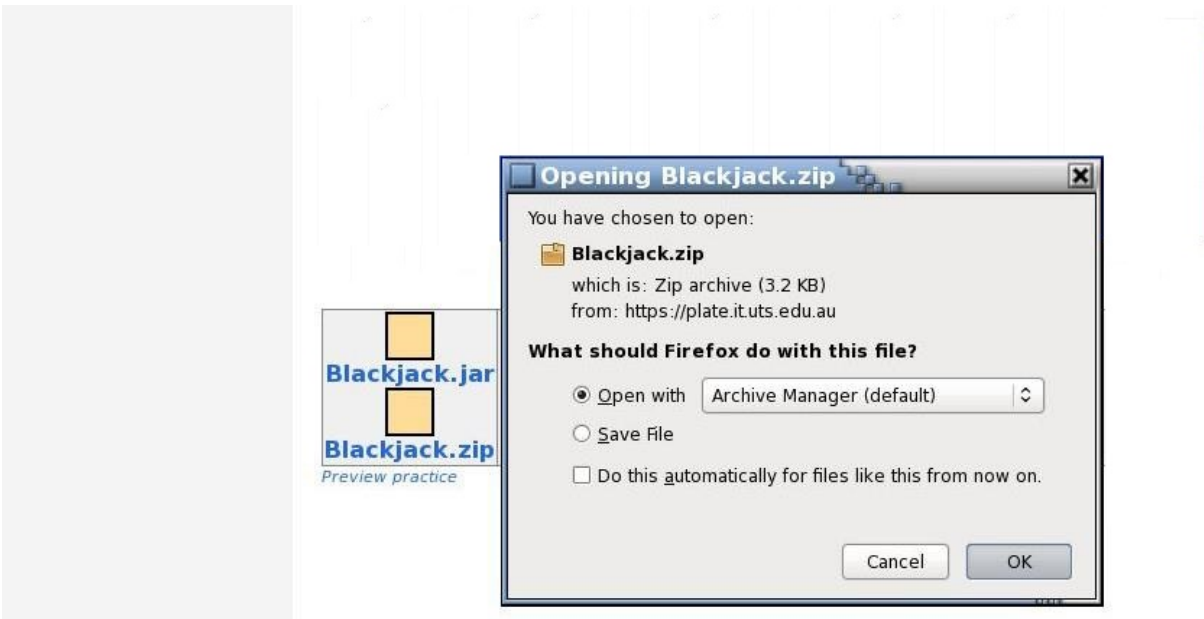
Choose “Java Application”



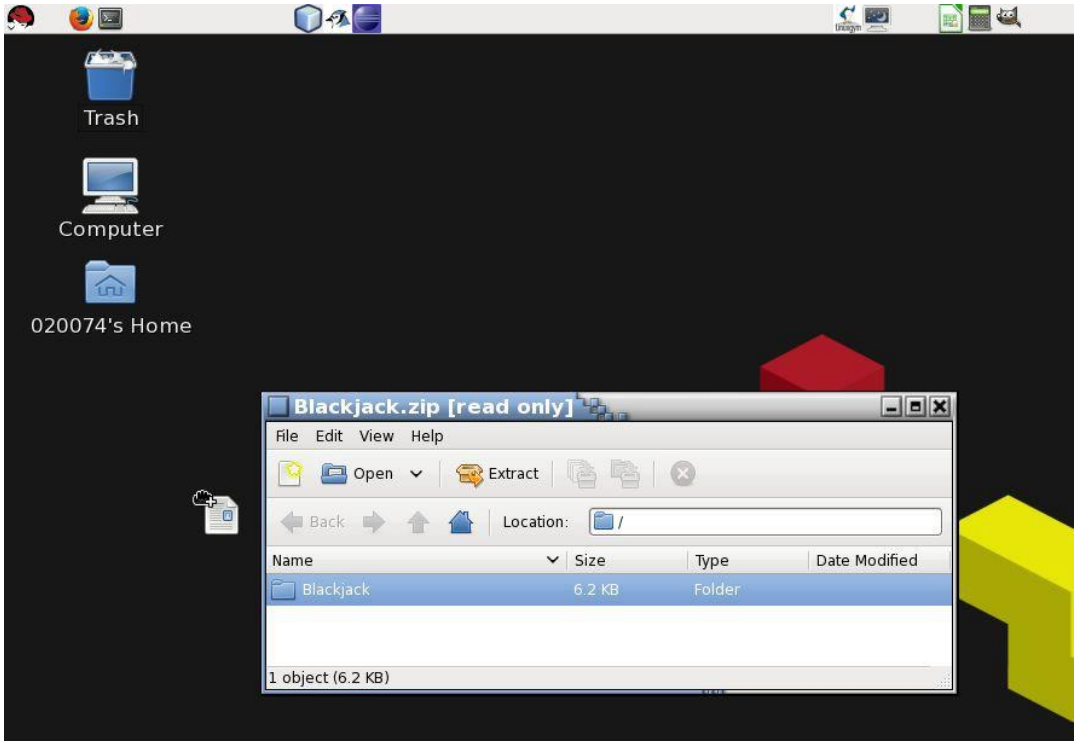
Enter the Project Name: Blackjack
Deselect “Create Main Class”. That is, do NOT create a Main Class
Don’t change anything else. Your project location/folder will be automatically chosen



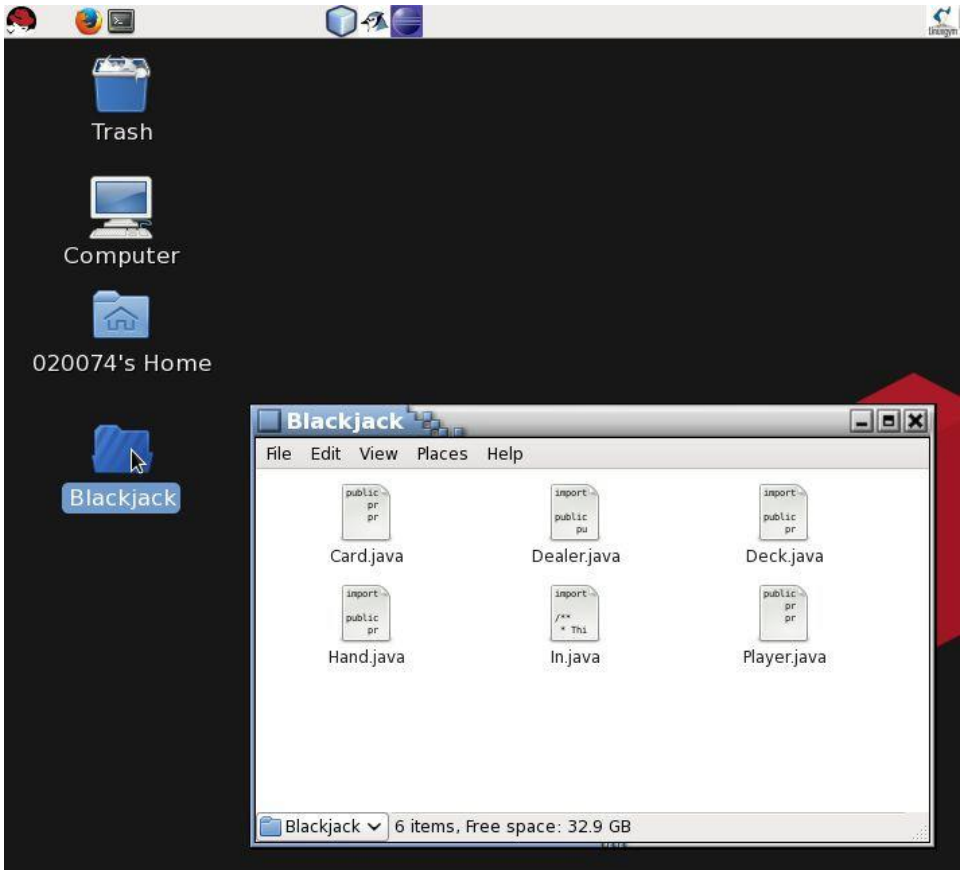
On desktop, click on Blackjack.zip and open it with the Archive Manager (or another preferred application for opening a ZIP file)



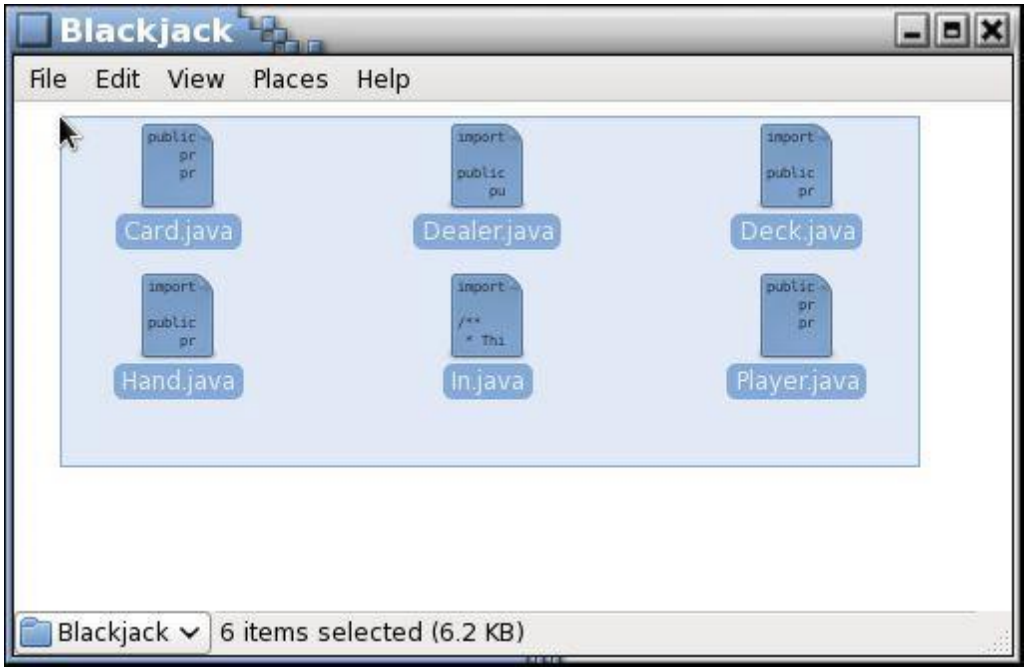
Drag the Blackjack folder from inside the ZIP file to the Desktop



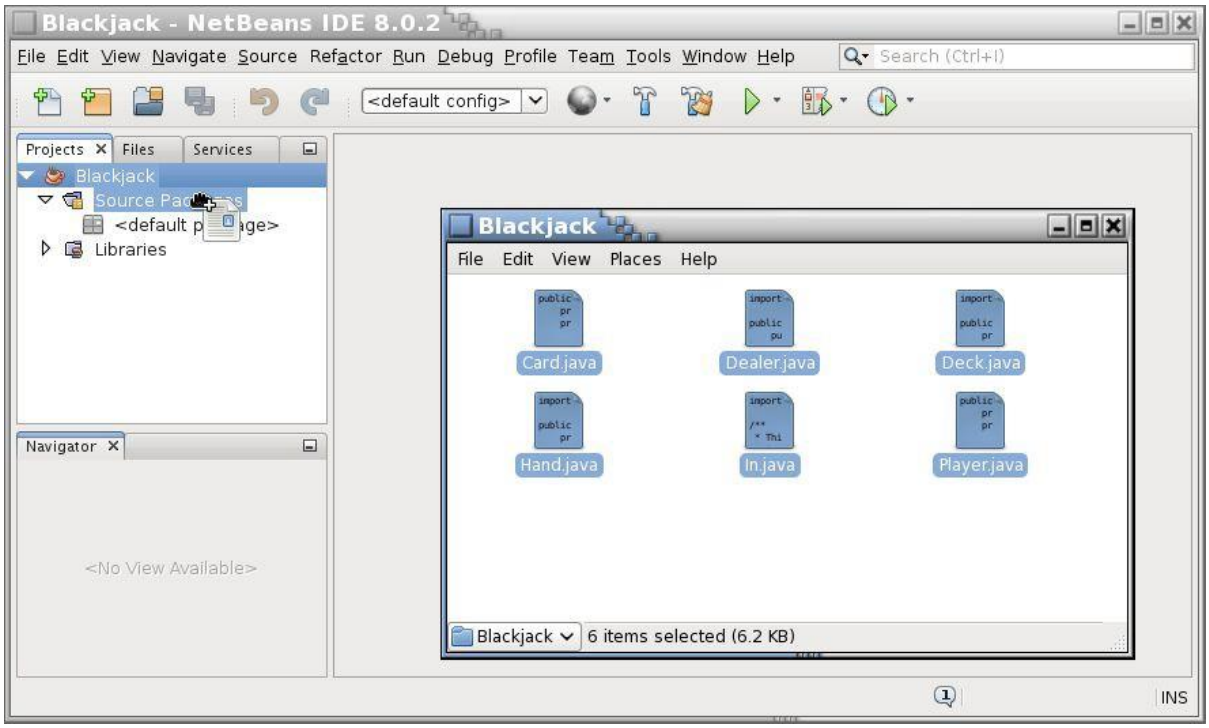
Double click on the Blackjack folder on the Desktop



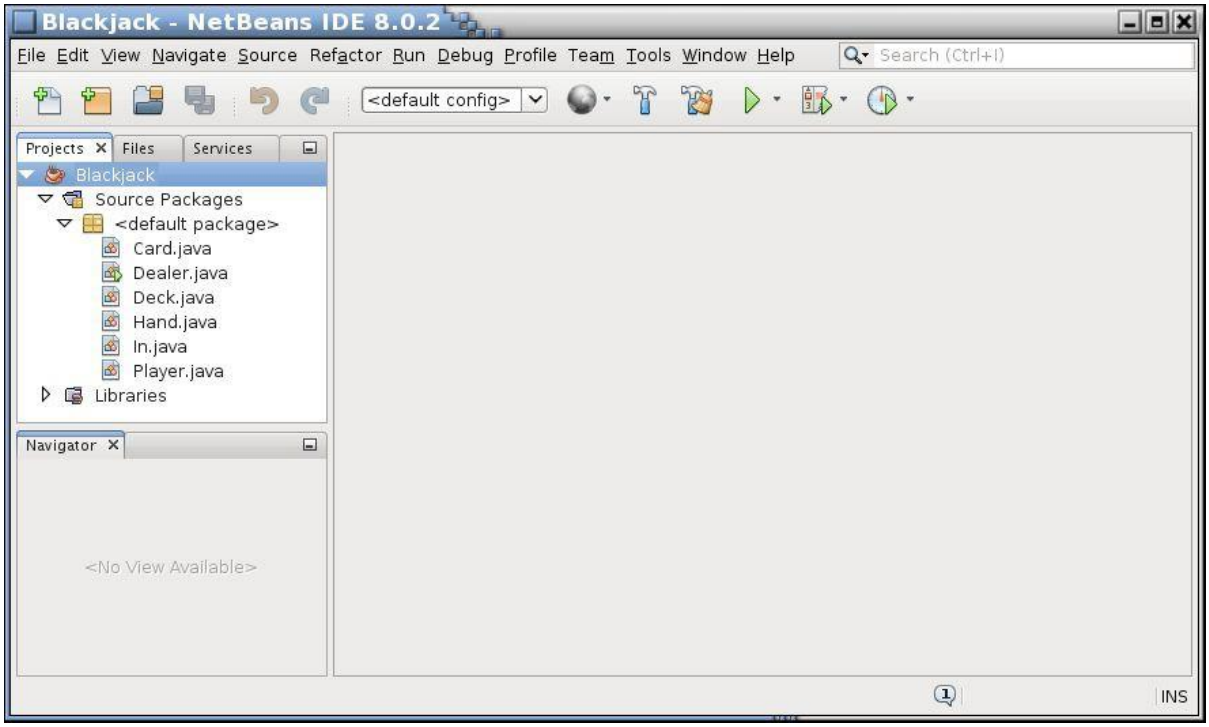
Select all Java files in the folder



Drag the Java files into the “Source Packages” folder inside your NetBeans project. NOTE: If you’re using MacOS, this doesn’t work. Instead, copy the files with ⌘C (Command-C). Then paste them with ⌘V (Command-V). Try pasting them into “Source Packages” first. If that doesn’t work, try pasting them into <default package>.



Expand to view the contents of the default package



You may double click on any Java file to view the code for that class within NetBeans.
Double-click on the Player.java file to open the definition of the Player class.
Click on the green PLAY button to run your program. Select Dealer as the main class.

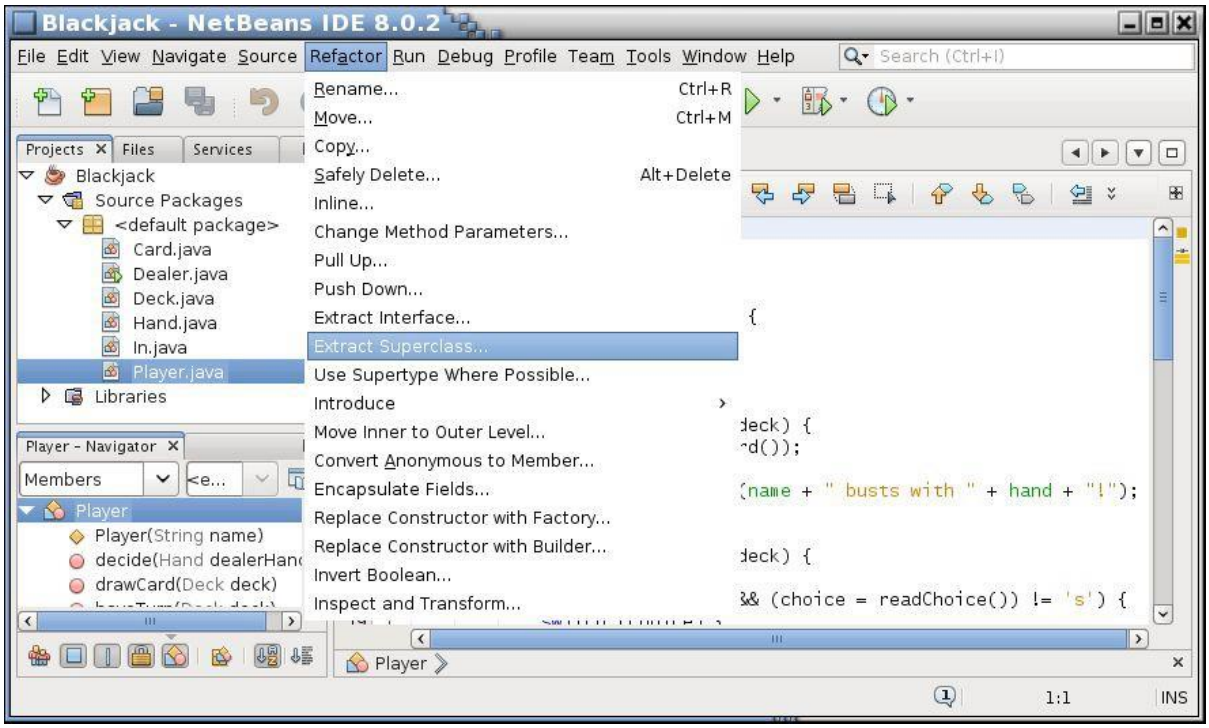
Before continuing, you may want to move the Blackjack folder from the Desktop to the Trash, since you have already copied the files into your NetBeans project.

Refactoring code

Next, you will experiment with NetBeans' refactoring features. As shown in Study 6, it is possible to share common code between related classes by **inheriting** shared code from a superclass. NetBeans provides a refactoring tool to help you do this.

If you make a mistake, take a look at the final version of the lecture code (with the Person superclass). The code can be downloaded as a ZIP file from the top of the Study 7 module.

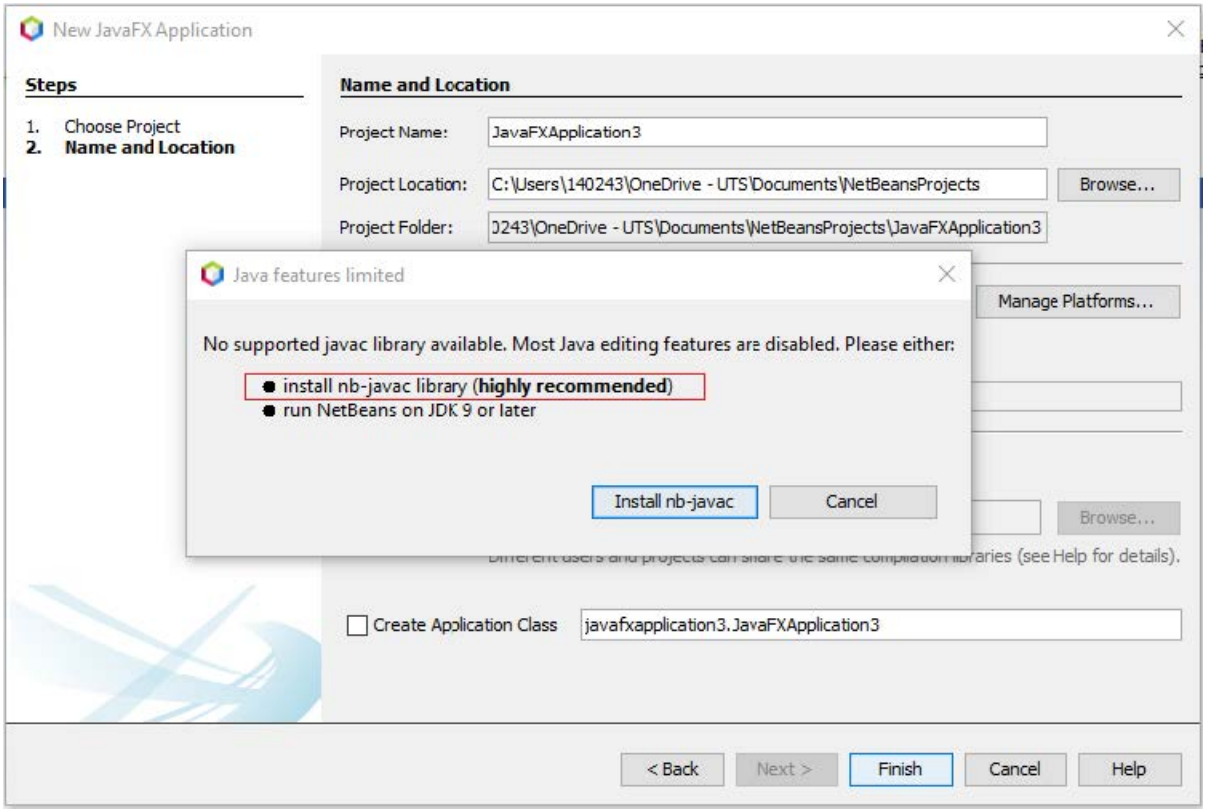
Open the Player.java file by double clicking on it. The Player class has a number of members that are in common with Dealer. We will extract those members into a superclass called Person. Select Extract Superclass… from the Refactor menu:

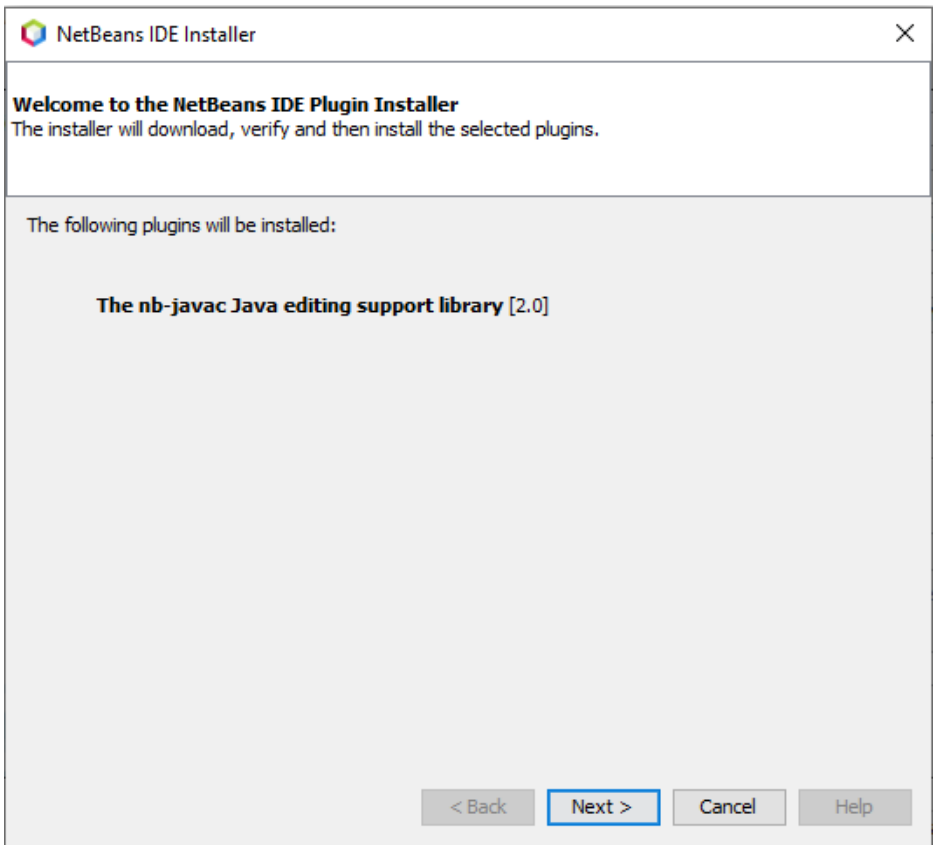


If you could not find the "Extract Superclss..." from the Refactor menu or it doesn't work, this is because the latest Java editing features haven't been involved in the default library in NetBeans. You need to download and install the "nb-javac Java editing support library" into your NetBeans. JavaFX project will also need it to activating most Java editing features.

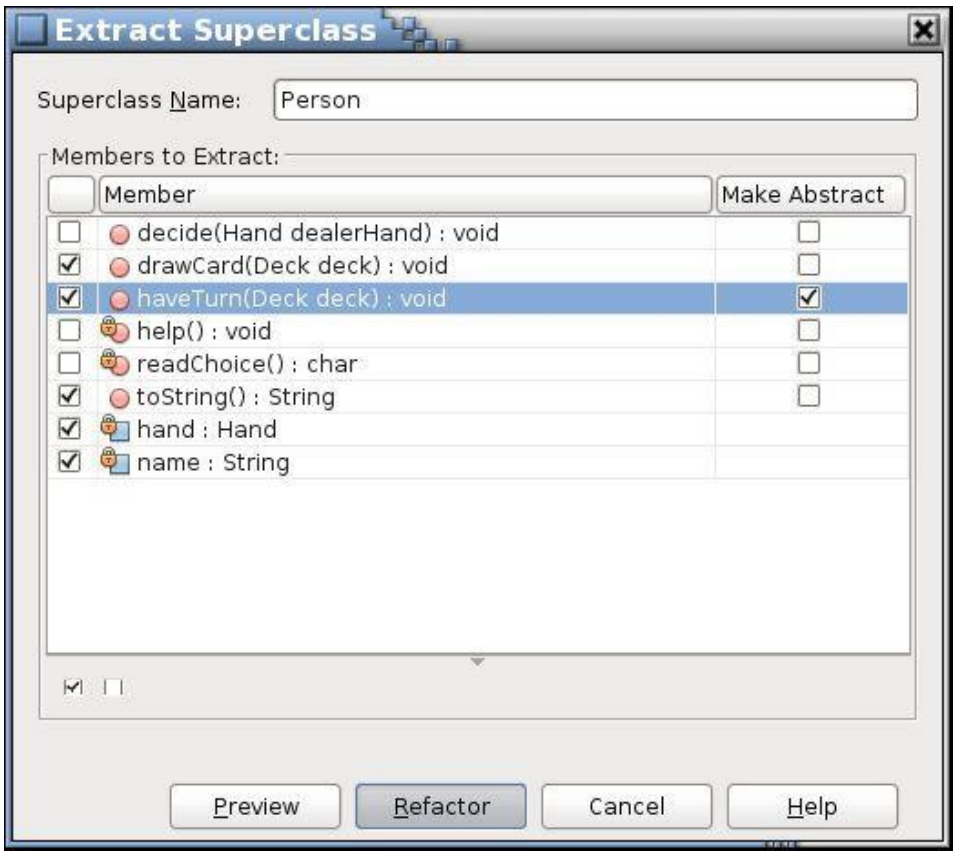
If you are using the lab pc on campus or the all-in-one NetBeans(8.2) + JDK(8u111) bundle, you don't need to install any extension to activate Java editing features.

When Java features limited window pops up, please select "Install nb-javac" instead of upgrading JDK to later version.





Name the superclass Person. You need to select the checkboxes below from **both** the LEFT column and the RIGHT column. If you miss a checkbox, you'll need to start from scratch!



What this means is that we want the following members to be shared by both Player and Dealer:

- drawCard
- *haveTurn*
- toString
- hand
- name

From the RIGHT column, we must also select the "Make Abstract" checkbox for *haveTurn* because Player and Dealer have to implement this method differently.

Have a look at the resulting code. Notice some members have lifted from class Player up to class Person.

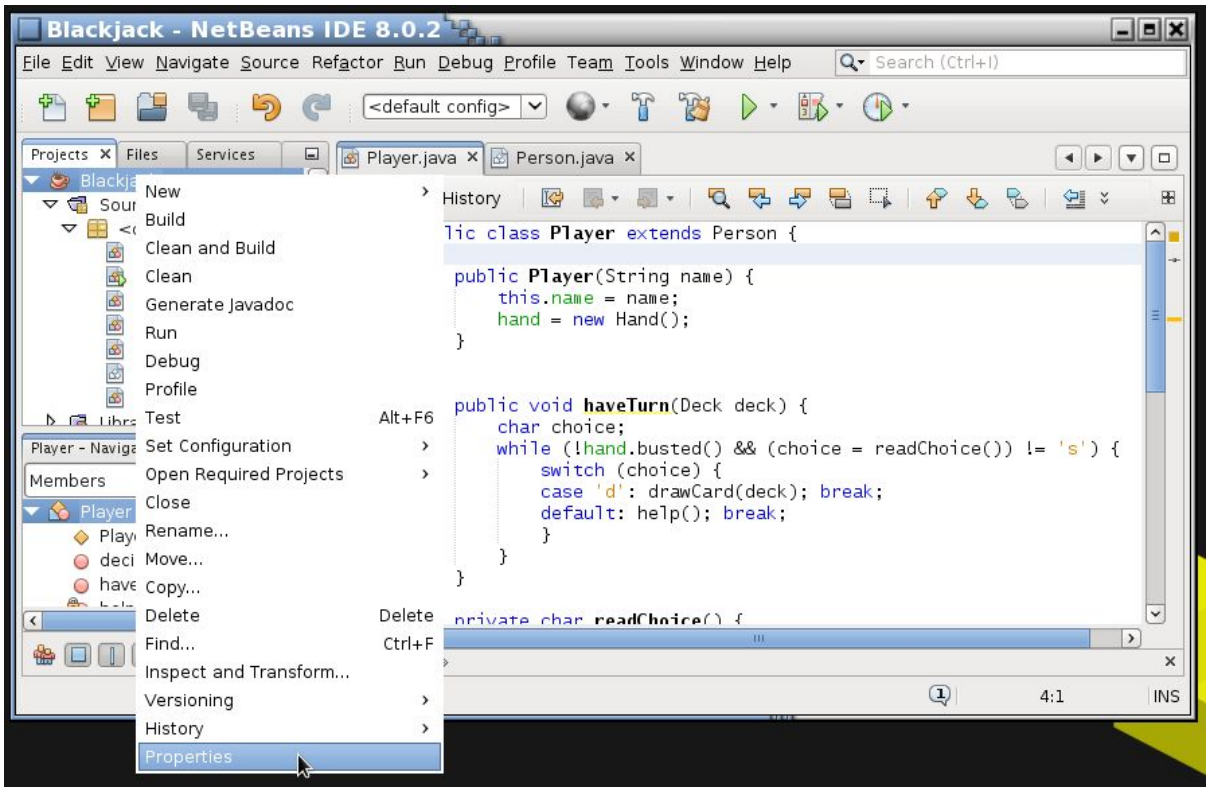
Note that the refactoring tool only made Player extend Person, but Dealer does not yet extend Person. You will need to do that one manually. Double-click on the Dealer.java file and modify the definition of class Dealer to extend Person. Delete from Dealer any members (both fields and methods) that are inherited. i.e. You can delete drawCard, toString, hand and name from Dealer since the Dealer class inherits these. The dealer **doesn't** inherit

haveTurn because the superclass didn't provide any implementation for it. So the Dealer has to keep its own implementation of haveTurn. You should add an @Overrides annotation above the Dealer's haveTurn method to indicate that it is defining its own version of this method. NetBeans might not have inserted an @Overrides annotation above the Player's haveTurn method either, so it would be good to add the annotation there, too.

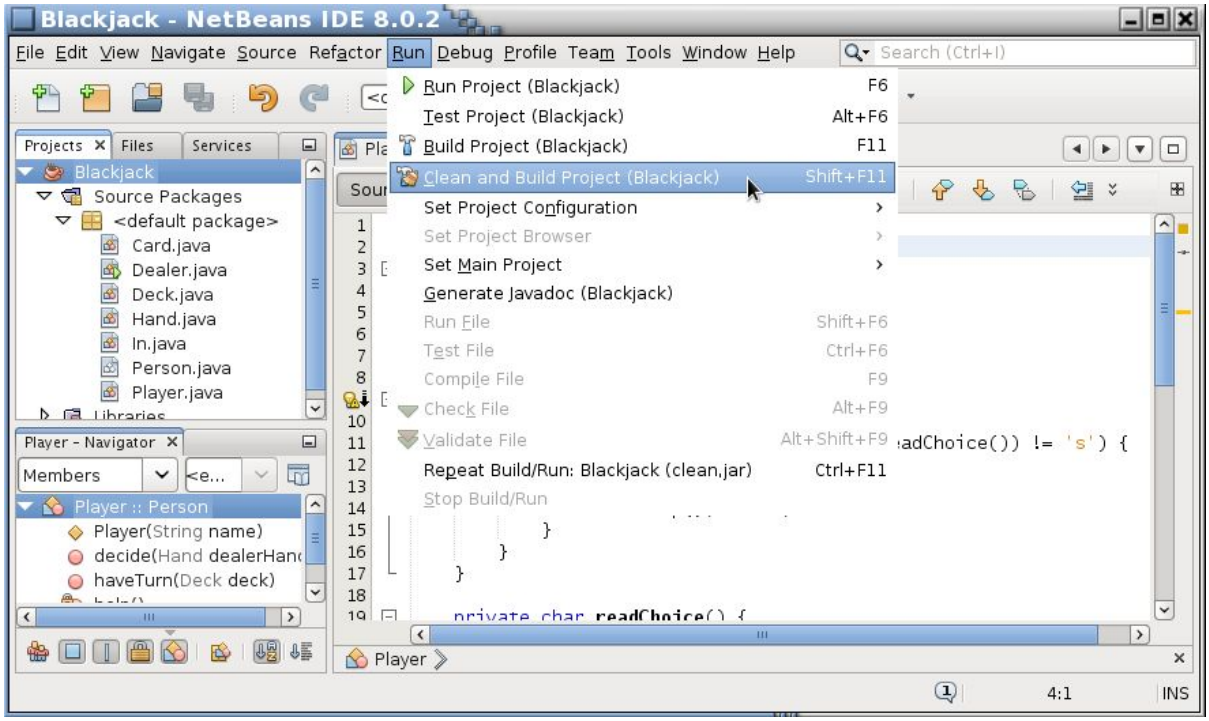
Optional: If you wish, you may improve it further by defining a constructor in the superclass Person to initialise the name and the Hand. Subclasses can invoke this by calling `super` as the first line of their constructors. The lecture demo ran out of time to demonstarte the **super** keyword, but you can see an example of it in the Lecture code ZIP file.

Submitting a JAR to Canvas

Finally, let's create a JAR file. NetBeans automatically compiles your code as you type code, and can also create a JAR file. By default, NetBeans will generate an executable JAR file named after the Project name under the file path: Project folder-->dist.



To create the JAR file, select the “Run” menu, and then select “Clean and Build Project”



Now submit this JAR file to Canvas. Remember it is in the “dist” directory of your project:

For this exercise, Canvas won’t test your code but ED does. The goal of this exercise is simply to learn how to carry out the entire development process in NetBeans, from importing code, to editing, to creating a JAR. What Canvas will check is that you managed to create your JAR file correctly from within NetBeans.

That’s it!

Remember the process so that you can submit future labs and Assignment 2 to Canvas.

If you forget the process, please come back to this study module and revisit this tutorial.