Lab3 - Website Chat

Reminder: You must first have your completed lab checked off by your TA (either in-person or by e-mail). Afterwards, you must submit your lab via Brightspace. Failure to do either will result in a grade of 0.

Goals

In this Lab you will:

1. Help design and implement multiple classes using Object-Oriented Programming principles we have covered in lecture, complete with getters and setters.
2. Deal with instantiating (using the `new` keyword) multiple instances of those classes to build an extremely simplified chat engine.

Resources

You will want to refer to each and all of the following to complete this Lab:

1. **Lab Overview** (this document).
2. **Lab3.zip**, the Starter Code for this Lab.
3. **Sample Exchange**, showing a sample input and output exchange between the program and a user running it on the command line, once the program is correctly written (attached).

Background

This Lab begins a multi-Lab extended project; each one will build upon the last, while exploring topics from lecture. This means that code you develop now will continue to be used in an ongoing fashion - so it's important that you understand your code, and write it in a way that you won't get confused over the next few weeks as you continue to add to it.

You are the C.T.O. of AmazBay, a new tech startup. Your co-founder, the C.E.O., has asked you to build a simple chat engine to allow two website users to privately send and receive text messages between them. A true chat engine running on the internet is outside of the scope of this Lab (and this course), instead you will be building something that approximates a chat engine, vastly simplified. There are three main Java classes that constitute our "server":

`class Person` - represents a single user on the website. Each `Person` should have at a minimum:

1. A *First Name*, also known as a "Given Name"
2. A *Last Name*, also known as a "Family Name"
3. A *UID*, or "Unique IDentifier", similar in nature to a Social Security Number or PUID. No two `Person` objects should have the same `uid`.
4. A method *`sendMessageTo()`* , to abstractly mirror the action of this `Person` sending a message to another person on the website. This will be achieved by creating a `new Message` instance, and calling `addMessage()` on some `Website` instance to record it.

`class Message` - represents a single text message from from one user to another on the site. Each

`Message` should have at a minimum:

1. A *sender*, of type `Person`.
2. A *receiver*, of type `Person`.
3. A *message*, the actual text content.
4. A *whenSent*, timestamp, that represents the date and time the message was sent.

`class Website` - represents the chat engine server and its data. A normal webserver would receive requests and transmit responses using something like `HTTP`; ours will merely provide some methods other code can call, and rely on `return` to send the response back. The website should have at a minimum:

1. *people*, several instances of type `Person`.
2. *messages*, several instances of type `Message`.
3. `addMessage()`, a way to register a new Message as having been sent on the site.
4. Some way to find someone on the site by their UID.
5. Some way to find all `Messages` sent to or by a `Person` on the site.

What To Do, And How

The most straight-forward explanation is that you can take the **Starter Code**, and find all of the `//TODO` comments in it and address them, until the program has a substantially similar appearance and operation to the **Sample Exchange** when run.

A great deal of **Starter Code** has already been written for you, including handling some of the requirements in **Background**. For anything missing, you are required to add attributes, getters, and setters, to implement all of the above; and if you would prefer you can even redesign the existing code too. You're also free to add any convenience or additional methods or attributes to your objects, beyond the ones required. It's up to you to pick the best types, names, and write the code how you prefer - while obeying the best practices we have covered in class.

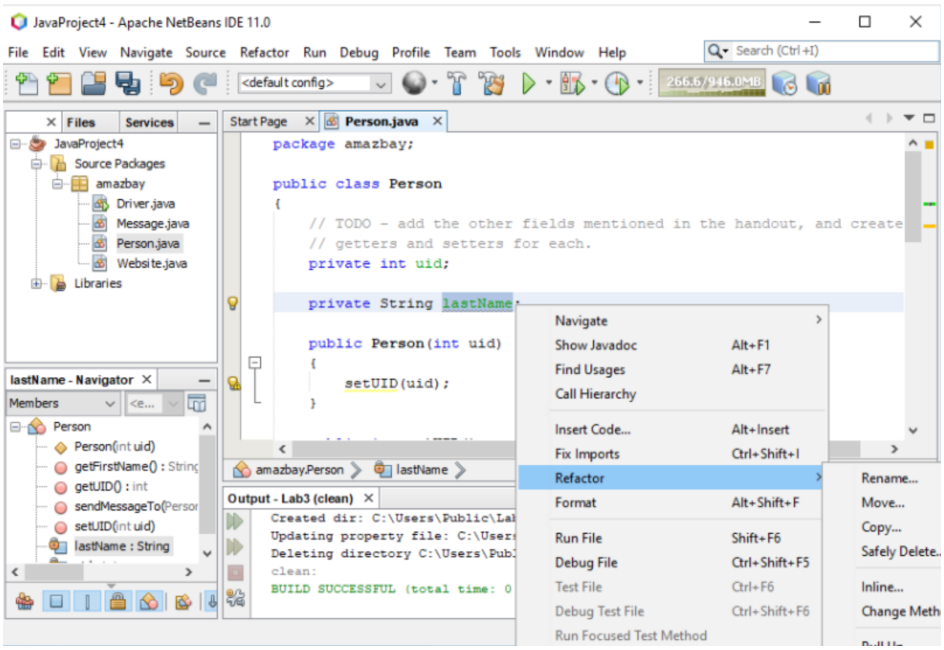
In addition to the `.java` files for `Person`, `Message`, and `Website`, there is a fourth file, `Driver.java`, which you and your TAs will use to confirm that your code is working and complete. `Driver.java` simulates creating a `Website` with several `Person` objects; and then, using a `Scanner` loop, creates several messages between site users. If you find using the Scanner loop tedious, especially as you keep re-running your program to test it, we recommend hard-coding some of the `Message` instances on the `Website` by calling `Person.sendMessageTo()` after you've finished creating the people on your website. After the messages are all specified, `Driver` also lets you view two users' chat history. Most of `Driver.java` has been written for you - but you will need to uncomment some code, and probably change the invocation of the various `Person` constructor calls.

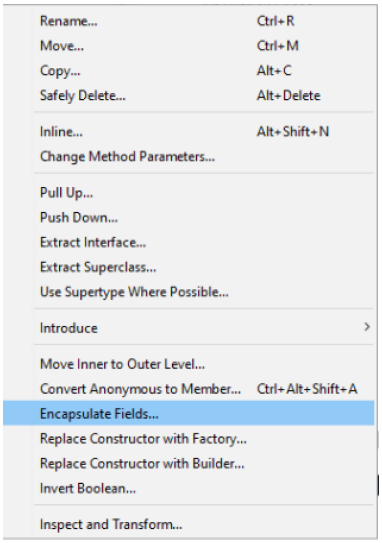
Because you will be specifying all of the fictional `Person` objects on your `Website`, and because there is substantial freedom in exactly how you represent and print `Message` objects, the output from your program is likely to differ from the **Sample Exchange** somewhat.

Letting `NetBeans` write your getters and setters for you

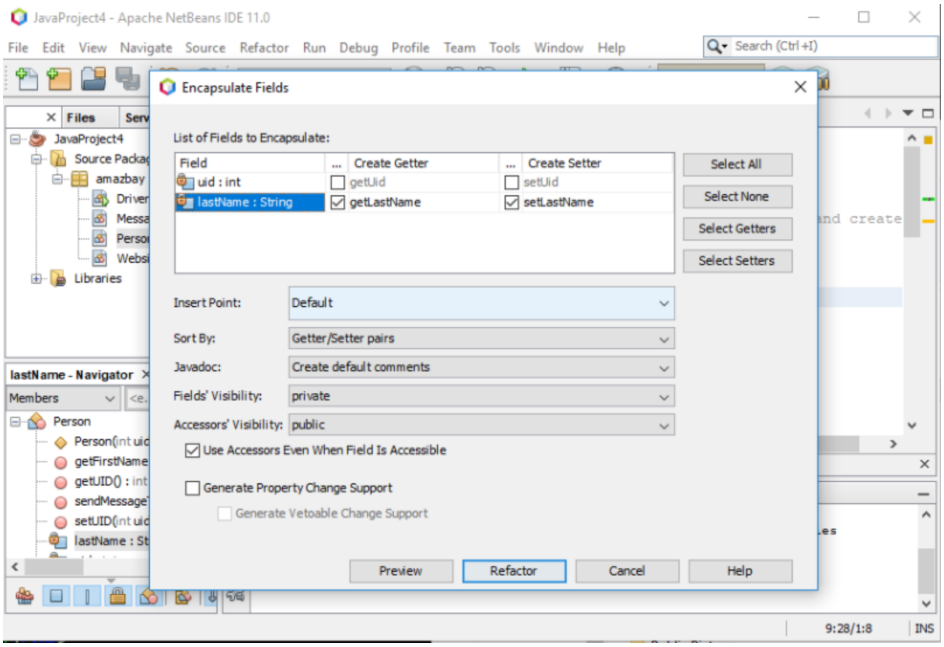
A lot of this lab involves writing getters and setters for new or existing attributes in a `.java` file. You can always do this by hand in any plain text editor, but some of you will prefer to use `NetBeans` to automatically add some of this "boiler plate" code for you.

If you have an attribute you would like to inject a getter or setter for (or both), simply select it with your mouse and then right click. From the popup menu, select "`Refactor`", then "`Encapsulate Fields`".





In the subsequent window, click the checkbox next to the code you want added. Here we say we would like `getLastName()` and `setLastName()` created for us, and specify what should be `public` and what should be `private`. You can also play with a few other settings here. Note that you could also do this later to **undo** or **update** this, and either remove or rewrite a getter or a setter. The names for the added methods will be automatically determined based on the variable name.



You can select multiple variables at the same time, and check all of the relevant getters and setters all at once. This can be *very* handy for adding a great deal of the code you'll need to add for this Lab, in just a few mouse clicks.

Regarding Timestamps

As currently written, `Driver` assumes that a `Message` is sent at the exact moment in time that the `new` keyword is used to create it, and achieves this using `System.currentTimeMillis()`, which returns the number of milliseconds that have elapsed since January 1, 1970 (so called "Unix Time", or "Epoch Time"). You can feel free to institute a different regime for simulating when `Messages` are sent (for instance, using `System.out.*` and `Scanner` to ask the user to specify this, or just hard coding it in your .java files) or how that moment in time is represented.

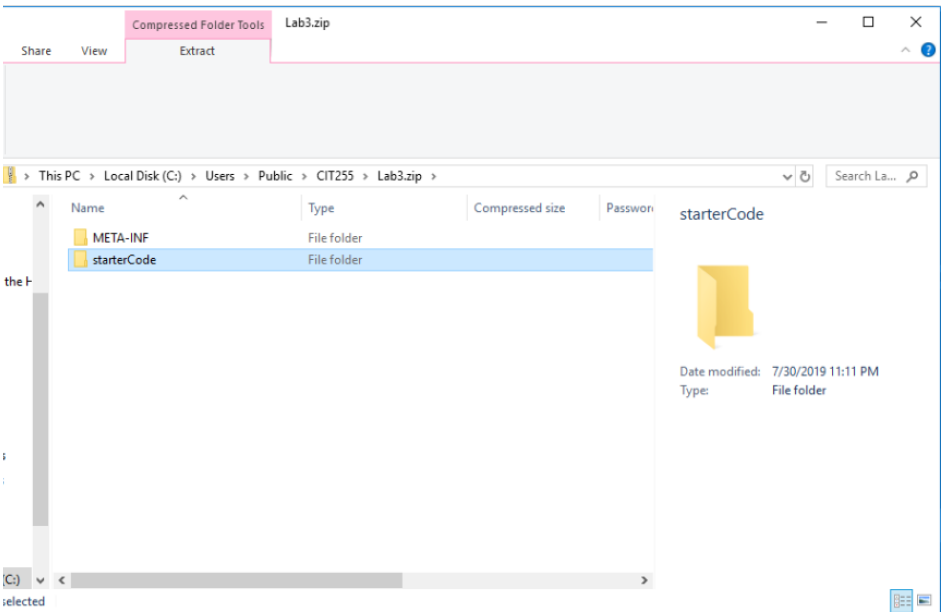
Unfortunately, printing gigantic integer numbers of milliseconds is a little obtuse when trying to report when something happened. We use the following magic formula to convert Unix Time to a nice readable form, like "07/21/2019 00:13:40":

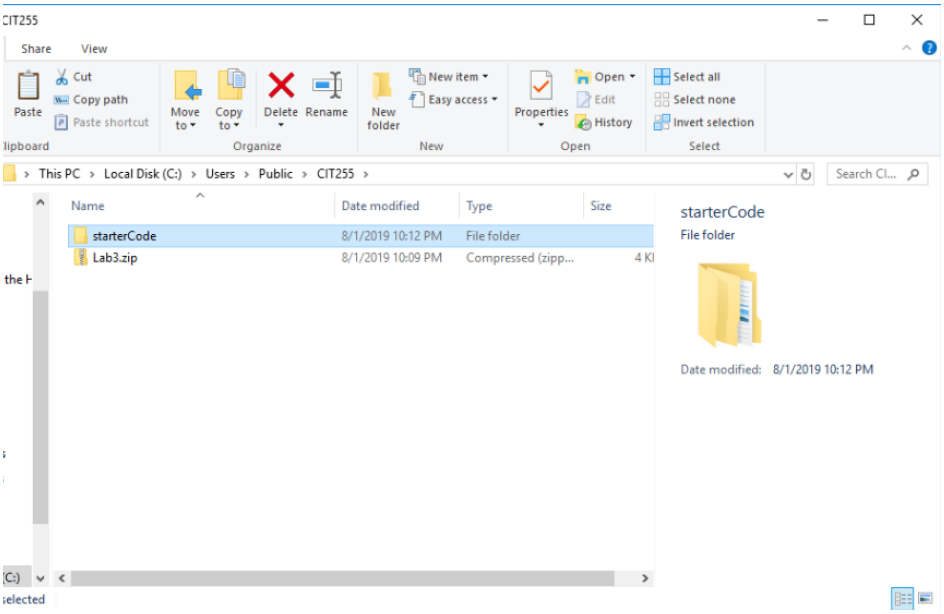
```
long milliseconds = System.currentTimeMillis();
String str = java.time.LocalDateTime.ofEpochSecond(
    milliseconds/1000L, 0, java.time.ZoneOffset.UTC
).format(java.time.format.DateTimeFormatter
             .ofPattern("MM/dd/yyyy HH:mm:ss"));
```

The part that reads `ZoneOffset.UTC` tells the computer to assume the timezone is the international "Greenwich Mean Time" of London, which will likely be several hours off from your local time. It is left as an optional exercise to figure out how to specify a different time zone instead.

Unlike in previous Labs, the **Starter Code** for this Lab has too many files to attach here. Instead you will find an appropriate source folder tree inside `Lab3.zip`, which your TA should provide to you.

Inside the `.zip` you will find a folder called `starterCode` - go ahead and pull this out of the `.zip` and put it somewhere on your computer you can find it.



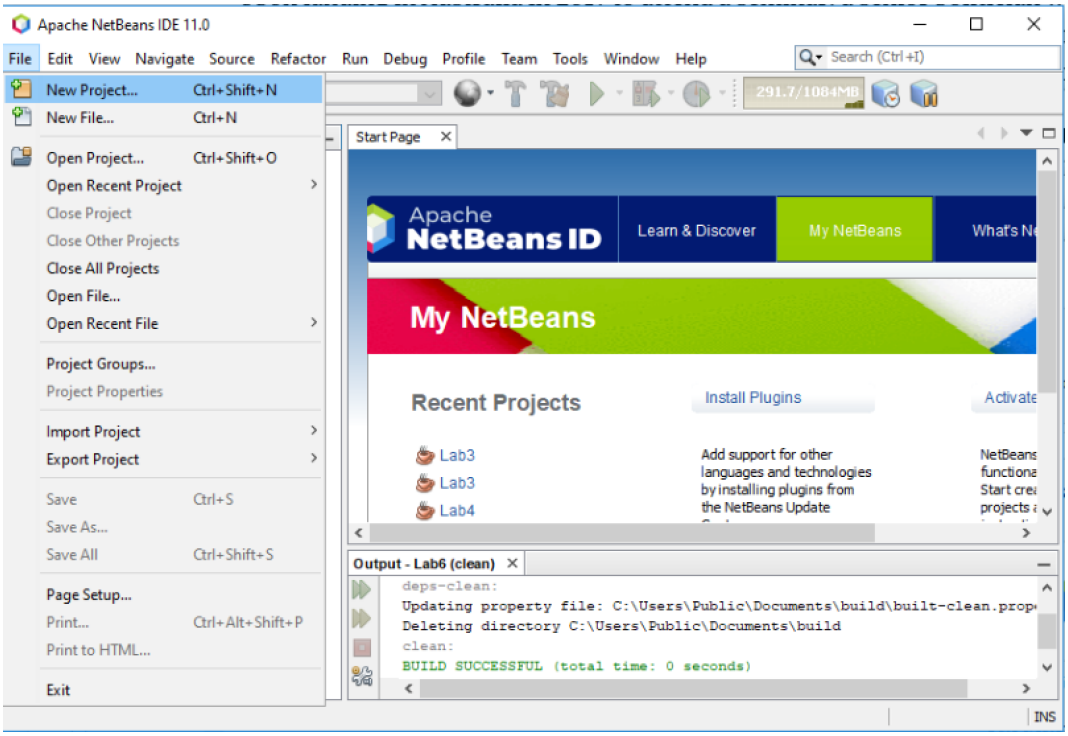


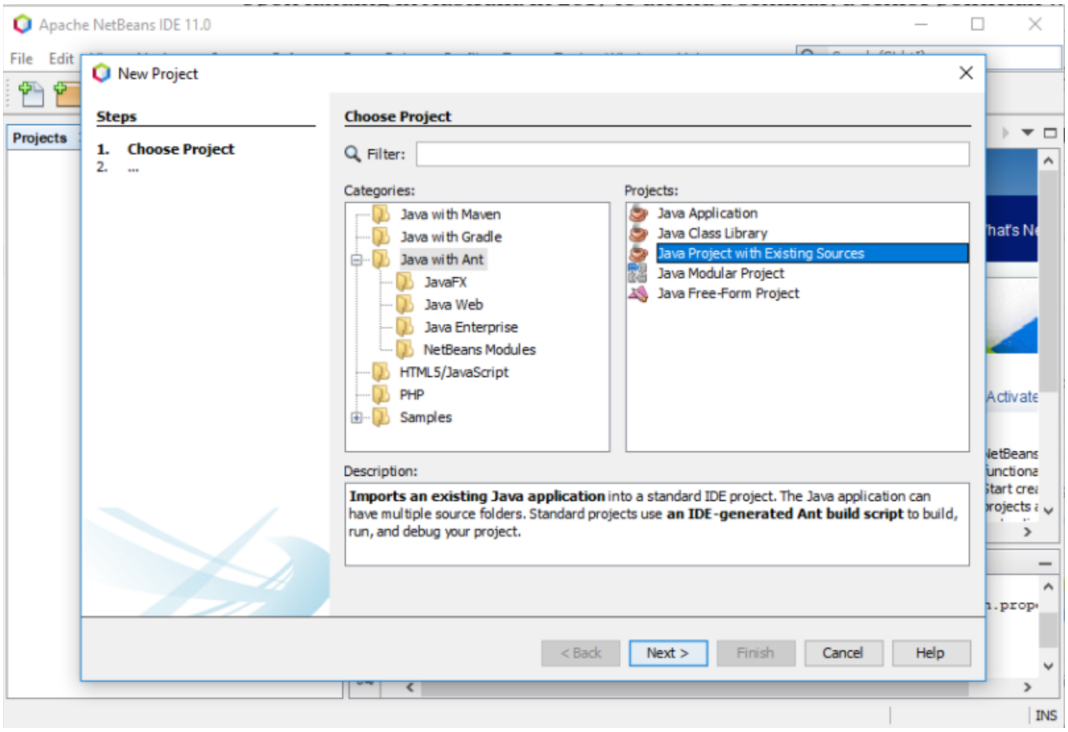
Next, we show how you can import this source code directly into a new `NetBeans` project. You

probably want to rename the folder to something besides `starterCode` - when you import into `NetBeans` like this, `NetBeans` will edit these files directly in place, so this is going to be YOUR code going forward.

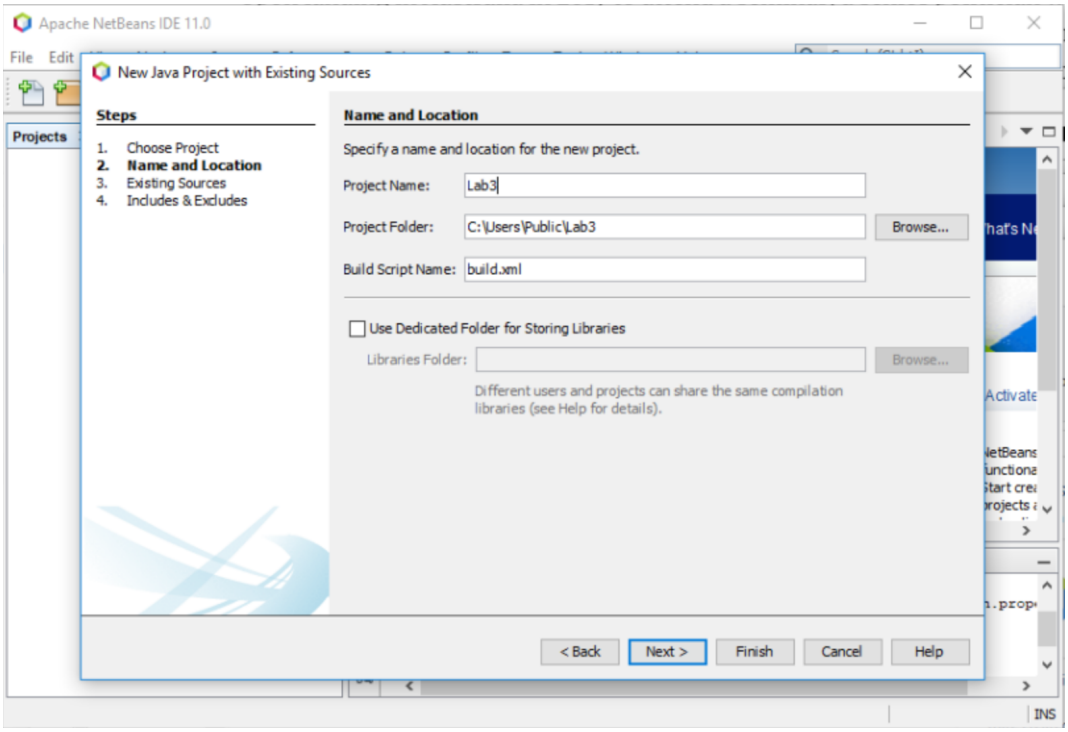
Note that all four files inside `starterCode` are in the package `amazbay` - this should remain true as you import these files into `NetBeans` and as you complete the Lab.

Start `NetBeans` as usual, and begin to create a `New Project` as usual, but this time choose `Java with Ant` → `Java Project with Existing Sources`:

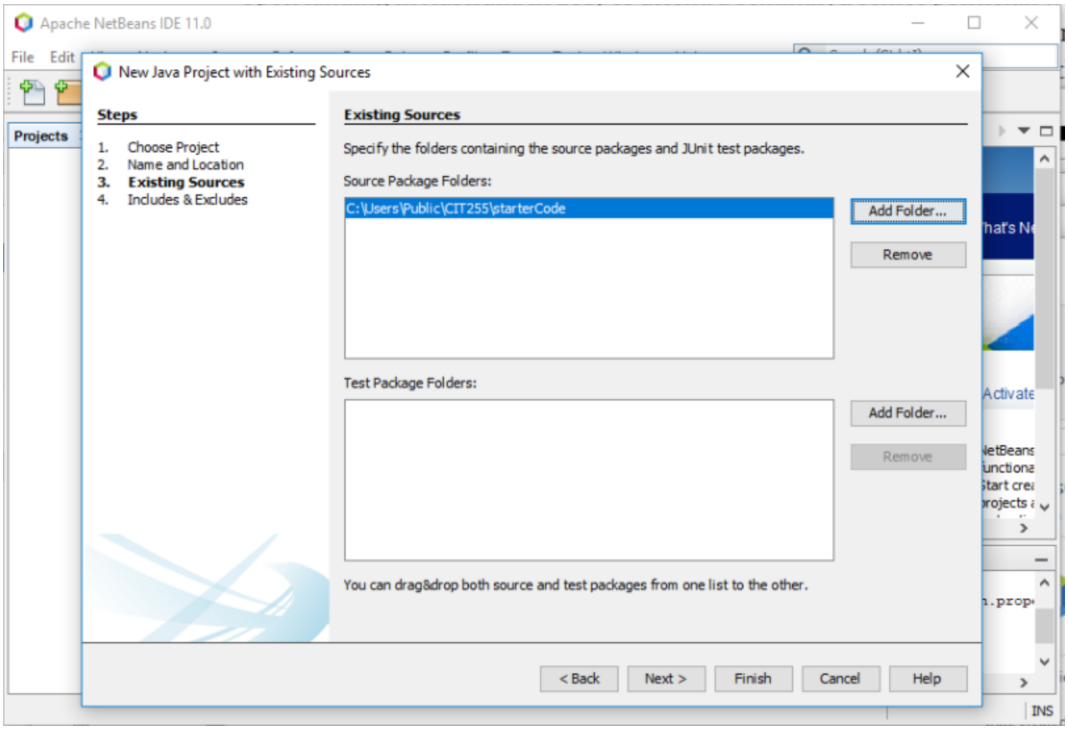




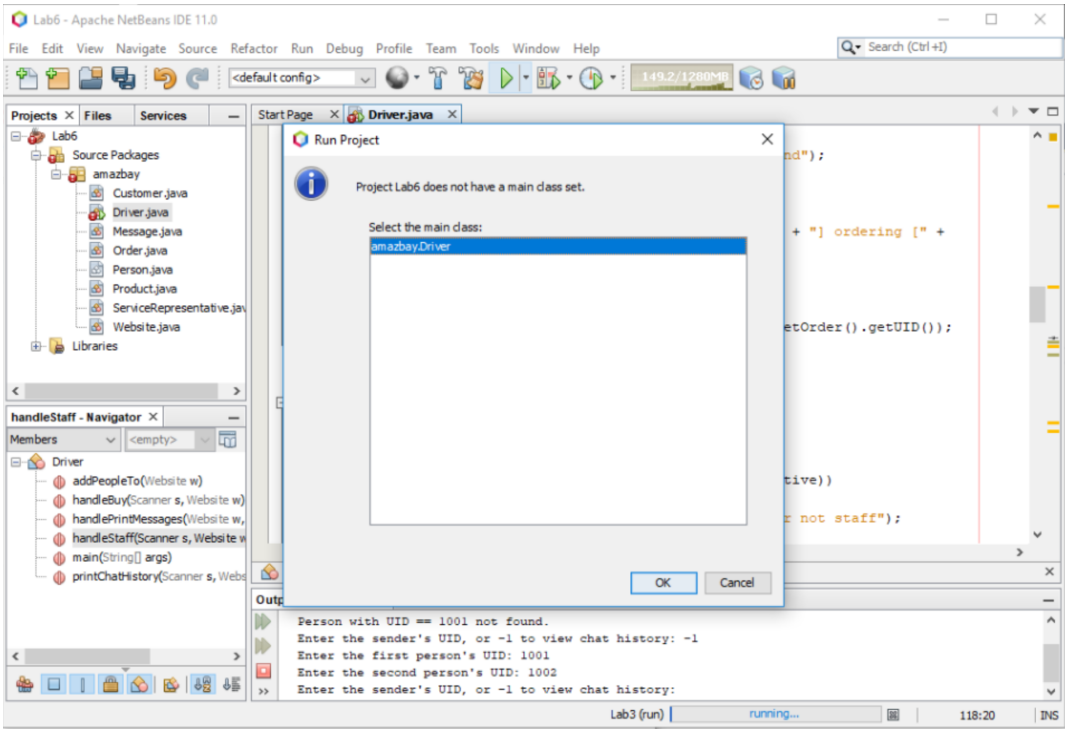
Name the Project like you normally would - `NetBeans` is going to create project meta files like it always does. But don't get confused later - your source code is not going to be where it normally is:



Instead, click `Add Folder..` and navigate to the folder you pulled out of the `.zip` file (hopefully with a name now other than `starterCode`):



You can then click `Finish`. The project should operate like normal, but the first time you go to run the project it may ask you to specify a `main class`. For all of the "AmazBay" Labs, choose `amazbay.Driver`:


Lab6 - Apache NetBeans IDE 11.0
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
Projects Files Services
Lab6
Source Packages
amazbay
Customer.java
Driver.java
Message.java
Order.java
Person.java
Product.java
ServiceRepresentative.jav
Website.java
Libraries
handleStaff - Navigator
Members
<empty>
Driver
addPeopleTo(Website w)
handleBuy(Scanner s, Website w)
handlePrintMessages(Website w,
handleStaff(Scanner s, Website w
main(String[] args)
printChatHistory(Scanner s, Webs
Start Page
Driver.java
Run Project
Project Lab6 does not have a main class set.
Select the main class:
amazbay.Driver
OK
Cancel
+ "] ordering [" +
etOrder().getUID());
tive))
r not staff");
Person with UID == 1001 not found.
Enter the sender's UID, or -1 to view chat history: -1
Enter the first person's UID: 1001
Enter the second person's UID: 1002
Enter the sender's UID, or -1 to view chat history:
Lab3 (run)
running...
118:20
INS

This is an example of running the program, including the text that prints out when using `System.out.print` and `System.out.println` and the actual text the user typed into the program (formatted **`like this`** below):

```
Type 'quit' to exit, anything else to keep going:  continue
Enter the sender's UID, or -1 to view chat history:  1001
Enter the receiver's UID: 1002
Enter the message text being sent:  Hi
Type 'quit' to exit, anything else to keep going:  continue
Enter the sender's UID, or -1 to view chat history:  1002
Enter the receiver's UID: 1001
Enter the message text being sent:  Who is this?
Type 'quit' to exit, anything else to keep going:  continue
Enter the sender's UID, or -1 to view chat history:  1001
Enter the receiver's UID: 1002
Enter the message text being sent:  It's your 255 TA
Type 'quit' to exit, anything else to keep going:  continue
Enter the sender's UID, or -1 to view chat history:  1002
Enter the receiver's UID: 1003
Enter the message text being sent:  Why is the 255 TA messaging me?!
Type 'quit' to exit, anything else to keep going:  continue
Enter the sender's UID, or -1 to view chat history:  1001
Enter the receiver's UID: 1002
Enter the message text being sent:  You forgot to send me your homework yesterday.
Type 'quit' to exit, anything else to keep going:  continue
Enter the sender's UID, or -1 to view chat history:  1002
Enter the receiver's UID: 1003
Enter the message text being sent:  False Alarm
Type 'quit' to exit, anything else to keep going:  continue
Enter the sender's UID, or -1 to view chat history:  1002
Enter the receiver's UID: 1001
Enter the message text being sent:  Whoops - I'll do that right now.
Type 'quit' to exit, anything else to keep going:  continue
Enter the sender's UID, or -1 to view chat history:  1001
Enter the receiver's UID: 1002
Enter the message text being sent:  Thanks.  See you in class.
Type 'quit' to exit, anything else to keep going:  continue
Enter the sender's UID, or -1 to view chat history:  -1
Enter the first person's UID: 1002
Enter the second person's UID: 1001
Showing chat history between [Jane Smith] and [255 TA]
At (07/21/2019 00:13:32), 255 TA said:  "Hi"
At (07/21/2019 00:13:36), Jane Smith said:  "Who is this?"
```

```
At (07/21/2019 00:13:40), 255 TA said:  "It's your 255 TA"
At (07/21/2019 00:13:46), 255 TA said:  "You forgot to send me your homework yesterday."
At (07/21/2019 00:13:52), Jane Smith said:  "Whoops - I'll do that right now."
At (07/21/2019 00:13:55), 255 TA said:  "Thanks.  See you in class."
Type 'quit' to exit, anything else to keep going:  continue
Enter the sender's UID, or -1 to view chat history:  -1
Enter the first person's UID: 1003
Enter the second person's UID: 1002
Showing chat history between [James Miller] and [Jane Smith]
At (07/21/2019 00:13:43), Jane Smith said:  "Why is the 255 TA messaging me?!"
At (07/21/2019 00:13:48), Jane Smith said:  "False Alarm"
Type 'quit' to exit, anything else to keep going:  quit
```