SYSC 3110 – Lab 3: Debugging and Git

Part 1: Debugging

A good debugger can easily double your productivity. You know by now that IntelliJ can help prevent java syntax errors with its code completion (for example by listing candidate methods to an object you just typed) and tips (the light-bulbs that often appear in the margin, offering for example to help you import the proper packages). But how about runtime errors? Runtime errors include null pointer exceptions, out of bound arrays, etc.

Typically, you first need to locate the bug, and then inspect the state of the program (i.e. the value of the variables, the methods on the call stack…) right before and after the bug appears. Using print() statements around suspicious zones of code is tempting at first, but it becomes very inefficient as soon as the bug proves hard to crack, as the print() statements proliferate and you can't keep track of them anymore.

This is where the debugger is useful. The debugger is a powerful tool, which lets you find bugs a lot faster by providing an insight into the internal operations of a program. This is possible by pausing the execution and analyzing the state of the program by thorough examination of variables and how they are changed line by line.

1. Your address book main method should look similar to below:

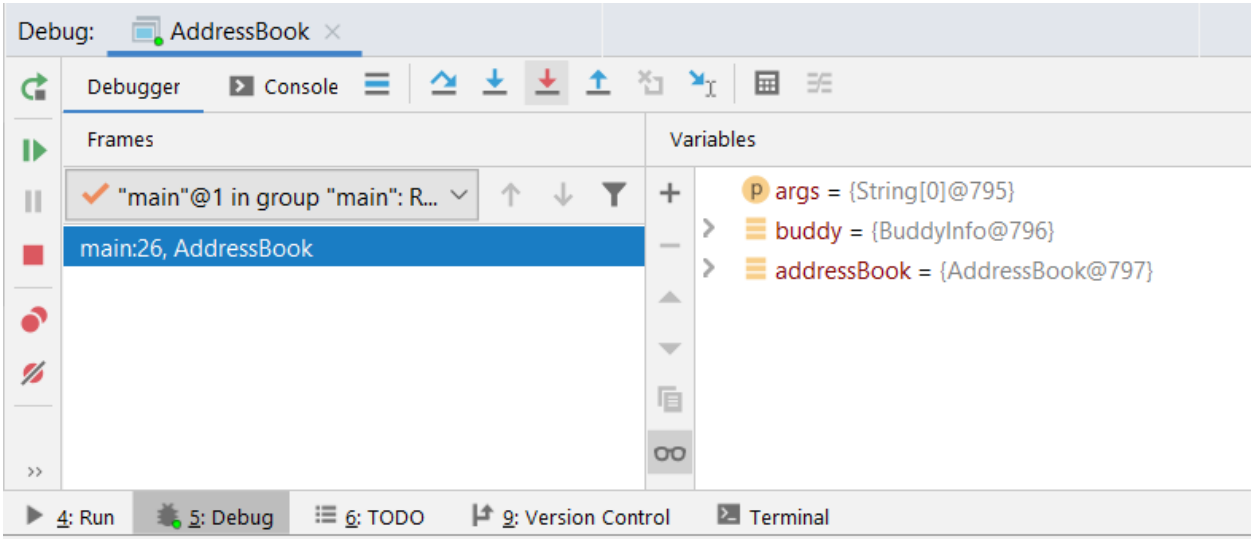
AddressBook.java | BuddyInfo.java

```
1  import java.util.ArrayList;
2
3  public class AddressBook {
4
5      private ArrayList<BuddyInfo> myBuddies;
6      public AddressBook(){
7          myBuddies = new ArrayList<>();
8      }
9
10     public void addBuddy(BuddyInfo aBuddy){
11         if(aBuddy != null) {
12             myBuddies.add(aBuddy);
13         }
14     }
15
16     public BuddyInfo removeBuddy(int index){
17         if(index >= 0 && index < myBuddies.size()){
18             return myBuddies.remove(index);
19         }
20         return null;
21     }
22
23     public static void main(String[] args) {
24         BuddyInfo buddy = new BuddyInfo( name: "Tom",  address: "Carleton",  phoneNumber: "613");
25         AddressBook addressBook = new AddressBook();
26         addressBook.addBuddy(buddy);
27         addressBook.removeBuddy( index: 0);
28     }
29 }
```

2. Set a break point next to addressBook.addBuddy(buddy) statement in the main() method of the AddressBook class. You can do so by clicking in the gray margin on the left side of the editor, next to the statement. A red dot should appear, as shown in the figure below.

```
23    public static void main(String[] args) {
24        BuddyInfo buddy = new BuddyInfo( name: "Tom",  address: "Carleton",  phoneNumber: "613");
25        AddressBook addressBook = new AddressBook();
26        addressBook.addBuddy(buddy);
27        addressBook.removeBuddy( index: 0);
28    }
29 }
```

3. Now let's launch the debugger by clicking on the "Debug" (or press Shift+F9) symbol on the tool bar. IntelliJ will start the program, open the debug tool window, and suspend execution when the program reaches its first breakpoint. The debug tool window should like the figure below:



4. Expand the addressBook variable. Expand the buddyInfo variable.
5. What is the value of the size variable?
6. Now click on the "Step Over" button (or press F8) on the toolbar of the debug tool window. That will execute the call to the current method, return from that call and show the new state.
7. Has the value of the size value changed? If so what is the new value?
8. There is much more to explore in here. Try "Step Into" (or press F7) a method, for example. When you are done, you can "Terminate" ( ) the program. Clicking on the red dot will remove the breakpoint.

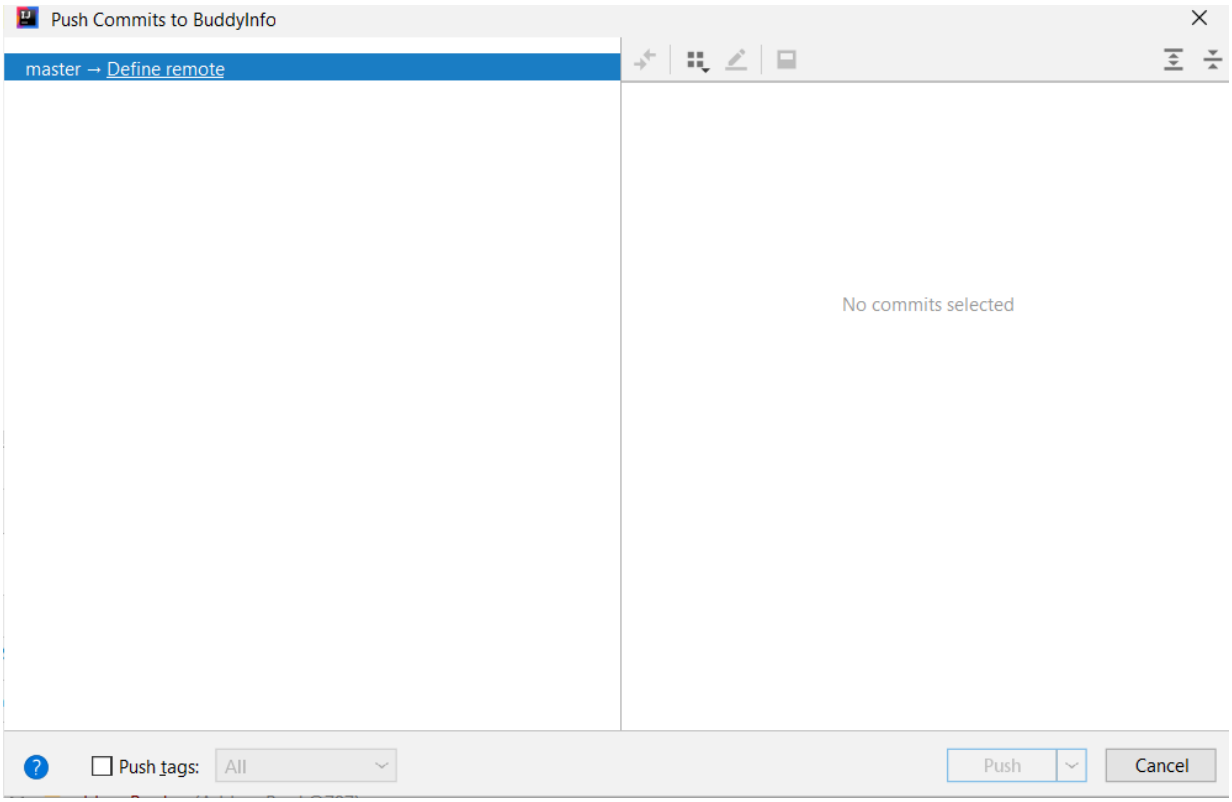
Part 2: Using an online repository – GitHub

In the last lab, you created a local repository. But what if we want to have our repository online so that several of us (or one person on several computers) can collaborate? In this lab we will show you how to use an online repository on GitHub to save versions of your code online and collaborate with other developers.

If you already have an account skip to step 4.

1. Go to https://github.com/
2. Click “Sign up for GitHub"
3. Create your account.
4. Logon to your account.
5. Click on the “+” in the toolbar at the top of the website and select “New repository”.
6. Enter a name for the repository and click “Create Repository”
7. The website will now provide you with the URL of your new repository. Make sure to copy this information because we will need it later.
8. Let’s share your BuddyInfo project online by adding the project to your online repository. To do this, select Git -> Push[1], or press Ctrl+Shift+K. The window below should popup:

[1] In older versions it might be “VCS -> Git -> Push”.



9. Select “Define remote”[2] and enter your GitHub repository URL and press “Ok”.
10. Now the local git repository you made in the previous lab can be pushed to your online GitHub repository. Press “Push”. If you pushed successfully you should now be able to see your src files in the GitHub repository online.
11. In IntelliJ, try making another change to your code and commit the changes. Are those changes immediately reflected online? What if you push your changes after committing?
12. At this point, only you are able to add code to your online repository (but anyone can view it). You will likely want several people to be able to commit code when working on your group project. To do this, on the GitHub website, go to your repository, click the setting tab and then select “Manage Access”. From here you can invite collaborators to your repository using their username or email address.
13. Make sure all of your files (including IntelliJ project files) have been committed and pushed to your online repository. Now, delete your project from the computer.

[2] Depending on your version of IntelII, you may be prompted after defining your remote to either authorize your JetBrains account through GitHub or to generate a token on GitHub and paste it into a pop-up.

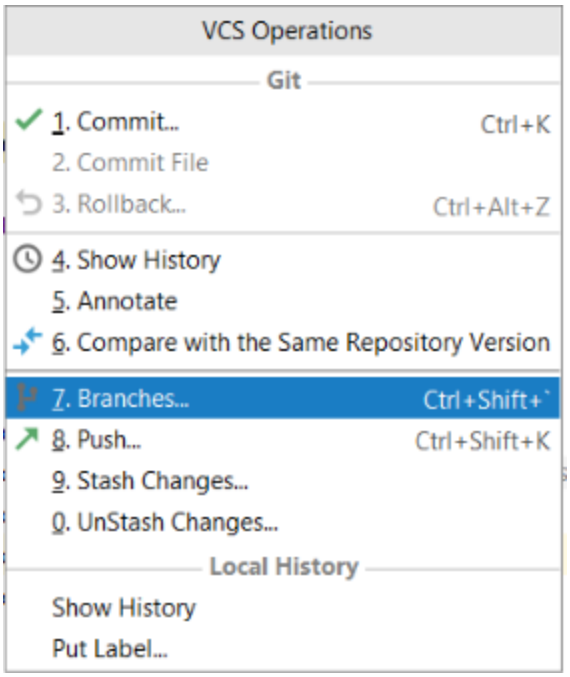
14. Normally, all of your code would now be gone. But since it is in the online repository, we can simply import your project. Select “Git -> Clone”[3]. Then enter the URL of your repository and select Clone.
15. IntelliJ will then clone the repository into your workspace and ask if you would like to open it. Select Yes. If IntelliJ asks which window to open the project select This Window.
16. You should find your project is back in your workspace, just like before you deleted it.
17. Let's edit your source code outside of IntelliJ. On the GitHub website, go to your repository and navigate to the AddressBook.java file. Click the Edit button (), add some text to the file and commit your changes.
18. These changes were made on the website so they will not show up in your IntelliJ project. This is similar to if one of your group members made changes to the code and committed them. To bring the changes into your IntelliJ project, select VCS -> Git -> Pull ( from toolbar or press Ctrl+T).
19. Your code in IntelliJ should now contain the changes you made on the website.
20. You now know how to set up your online repository, push changes to the online repository, import an existing project and pull changes from the repository.

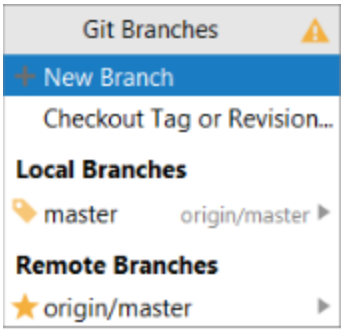
Part 3.0: Branching Code

At times, you want to make changes to the project that are experimental or take the project in another direction. Instead of making the changes to the main project, that other people might be using and working on, you can create a branch of the code. In IntelliJ, We will use the VCS Operations Popup to create a branch (you could also select VCS -> Git -> Branches or use the shortcut Ctrl+Shift+`).

1. To bring up the popup press Alt+` then select branches.

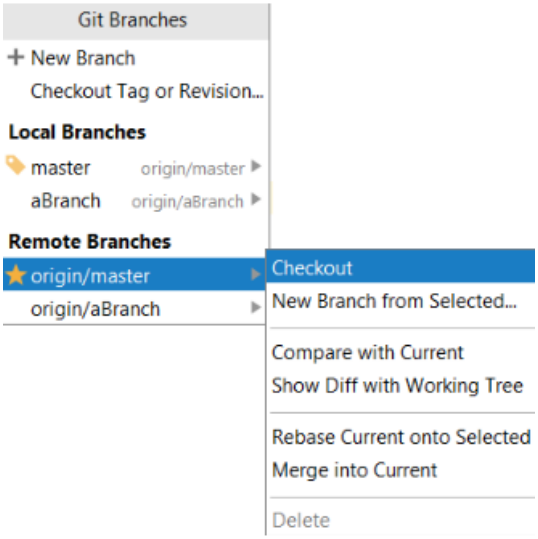
[3] Or “VCS -> Get from Version Control” depending on your version of IntelliJ.



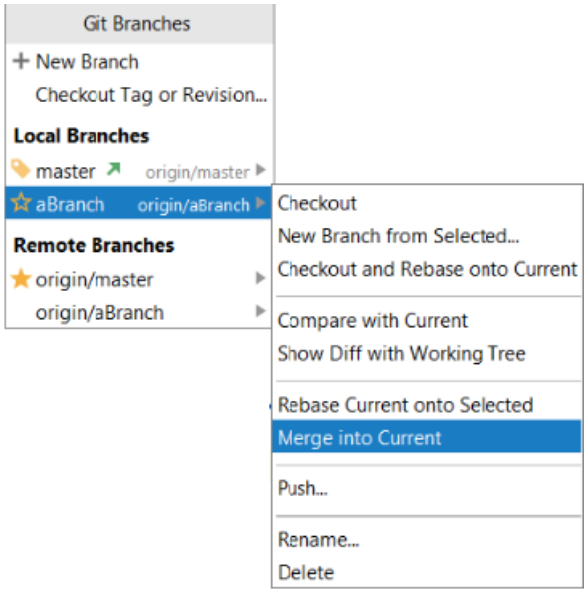


2. Select New Branch from the Git Branches menu.
3. Name your new branch and select Checkout[4].
4. You can now make some changes to your code. Add a new method and commit your changes. Now push it to the remote repository.
5. Remember, since this is a new branch it will not influence the main branch of the code. Your branch started from the same code as the main branch but can now go off in its own direction. Any of your teammates who are working on the main branch of the code will not pull the changes you made to this branch.
6. Maybe, after working on your experimental new version of the code, you want to add your changes back to the main branch (that your teammates are working on). To do this switch back to the main branch by bringing up the VCS Operations Popup, select branches again. Then checkout the master remote branch.

[4] In newer version of IntelliJ, the Checkout branch is now a checkbox (selected by default) and the button you need to click is labelled Create.



7. Finally bring up the VCS Operations Popup and select branches again. Now go to your branch in the local branches and select merge into current (merge into master branch).



8. Now the changes you have made in your branch should have merged into the master branch and you should see your changes have been added. Push the resulting merge to the online repository.

All done? What to show the TA:

- Debugging portion: show what happens to the size variable when you “step over” in steps 6-7.
- GitHub/branching portion: show your GitHub repo with your two branches and the various commits you made following the steps of the tutorial.