

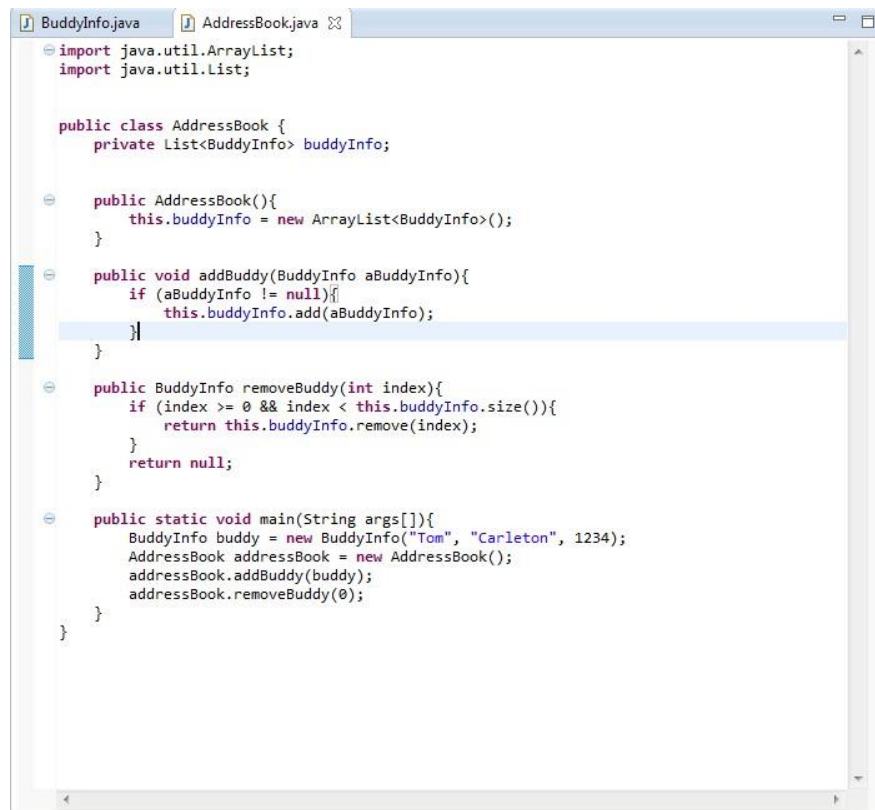
1.0 - Debugging

A good debugger can easily double your productivity. You know by now that Eclipse 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 package). 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.

With a debugger you can set breakpoints, which suspend the program when that line of code is reached. This allows you to look at the value of each variable in the program. At the same time, you can watch the value of variables and find out whether they are set correctly.

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



```
import java.util.ArrayList;
import java.util.List;

public class AddressBook {
    private List<BuddyInfo> buddyInfo;

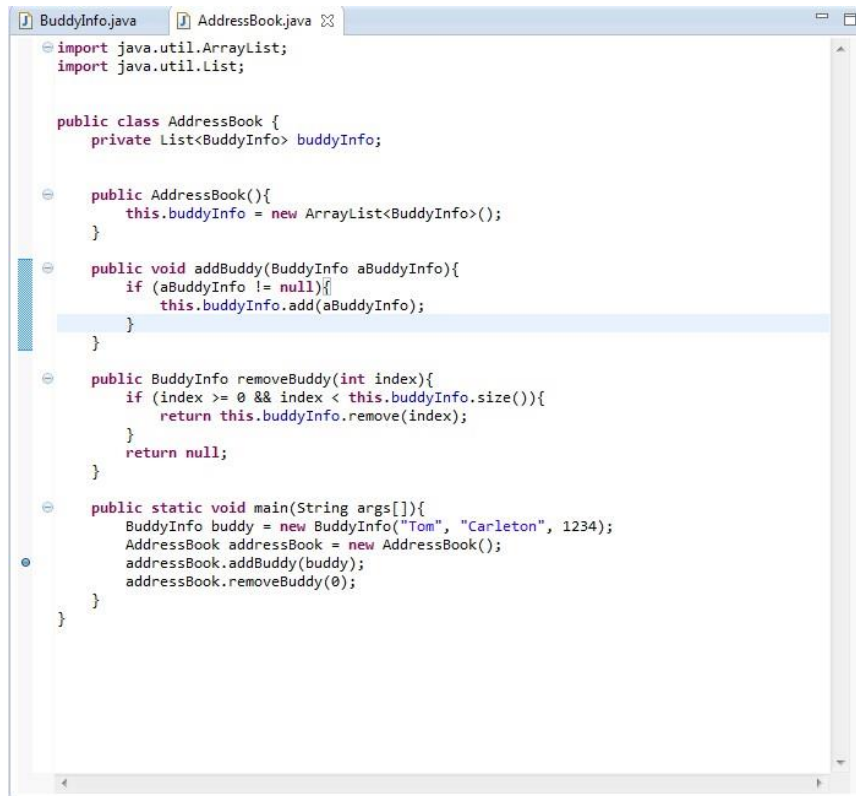
    public AddressBook(){
        this.buddyInfo = new ArrayList<BuddyInfo>();
    }

    public void addBuddy(BuddyInfo aBuddyInfo){
        if (aBuddyInfo != null){
            this.buddyInfo.add(aBuddyInfo);
        }
    }




    public BuddyInfo removeBuddy(int index){
        if (index >= 0 && index < this.buddyInfo.size()){
            return this.buddyInfo.remove(index);
        }
        return null;
    }

    public static void main(String args[]){
        BuddyInfo buddy = new BuddyInfo("Tom", "Carleton", 1234);
        AddressBook addressBook = new AddressBook();
        addressBook.addBuddy(buddy);
        addressBook.removeBuddy(0);
    }
}
```

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



Now let's launch the debugger (for example by clicking on the "bug" symbol on the toolbar; make sure you "Debug As" a Java Application). Eclipse will start the program, open the Debug Perspective, and suspend execution when the breakpoint is reached. This new Perspective can seem crowded at first, but mostly you will use the buttons of the Debug pane to step through the code, and the variables listed in the Variable pane.

3. Expand the `addressBook` variable. Expand the `buddyInfo` variable.
4. What is the value of the size variable?
5. Now click on the "step over" button ( or press F6) on the toolbar of the Debug pane. That will execute the call to the current method, return from that call and show the new state.
6. Has the value of the size variable changed? If so what is the new value?
7. There is much more to explore in here. Try "stepping into" () a method, for example. When you are done, you can "terminate" () the program. Finally, double-clicking on the blue dot will remove the breakpoint.

2.0 - Using an online repository – GitHub

In the last section, 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? We will use the website GitHub.

If you already have a GitHub account skip to step 4.

1. Go to <https://github.com/>
2. Click "Sign up for GitHub"
3. Create an account
4. Log on to newly created account
5. Click on the plus sign in the toolbar at the top of the website and select create 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, right-click the project and select Team->Remote->Push...
9. Enter your GitHub repository URL, your username and your password. Click Next.
 - a If you have two-factor authentication enabled on GitHub, this will not work. If this is the case, keep reading; else, go to step 10.
 - b Click on your profile picture in the upper right corner and go to settings.
 - c Click on "Personal access tokens" in the left hand menu.
 - d Click "Generate new token" in the upper right corner, enter a description for your token and make sure to check "public_repo" in the list of scopes. Click the Generate button at the bottom of the page.
 - e Copy your newly created personal access token and use it as your password in eclipse.

Push to Another Repository

Enter the location of the destination repository.

Location

URI: Local File...

Host:

Repository path:

Connection

Protocol:

Port:

Authentication

User:

Password:

Store in Secure Store ☐

? < Back Next > Finish Cancel

10. On the next page, click the “Add All Branches Spec” button, then “Finish”.

Push to: https://github.com/caleb-chan/SYSC-3303.git

Push Ref Specifications

Select refs to push.

Add create/update specification

Source ref: Destination ref: + Add Spec

Add delete ref specification

Remote ref to delete: + Add Spec

Add predefined specification

Add Configured Push Specs Add All Branches Spec Add All Tags Spec

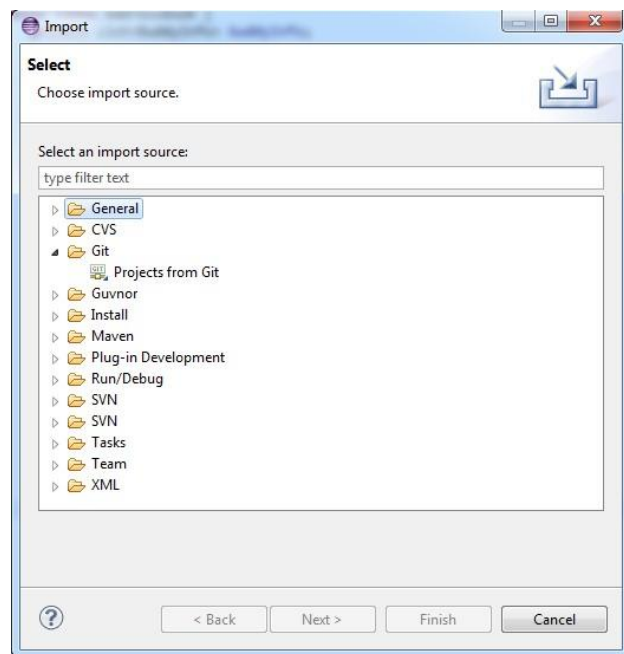
Specifications for push

Mode	Source Ref	Destination Ref	Force Update	Remove
Update	refs/heads/*	refs/heads/*	<input type="checkbox"/>	

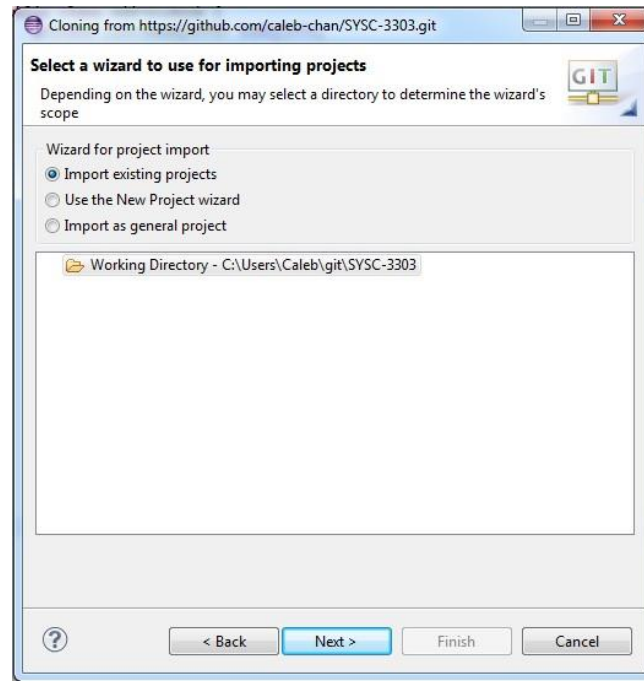
Force Update All Specs Remove All Specs

? < Back Next > Finish Cancel

11. In Eclipse, 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 Admin tab and then select Collaborators. You can then enter the information of your group members
13. If you want to share not just your code, but your entire Eclipse project (if all team members will be using Eclipse for development), you will need to also add your Eclipse project files to Eclipse. In the Project Explorer window, select the downward arrow icon in the top right corner. Click Customize View... and in the Filters tab uncheck the box `.* resources`.
14. You should now see `.classpath` and `.project` in your project. You can add those to your repository, commit them and push them to the online repository.
15. Make sure all of your files have been committed and pushed to your online repository. Now, right-click your project and delete it. Select the option that also deletes all of the files off the computer.
16. Normally, your code would now be gone. But since it is in the online repository we can simply import your project. Right-click in the Package Explorer window and click Import...



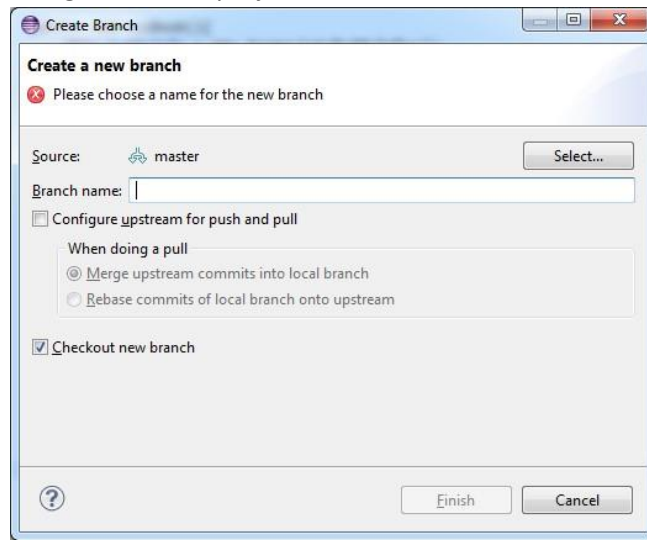
17. Select Git->Projects from Git and click Next.
18. Select Clone URI and click Next
19. Enter your online repository information, your GitHub username and password. Press Next until you get to the following screen.



20. Select Import existing projects and click Next.
21. Follow the screens and click Finish.
22. You should find your project is back in your workspace, just like before you deleted it.
23. Let's edit your source code outside of Eclipse. On the GitHub website, go to your repository and navigate to the HelloWorld.java file. Click the Edit button, add some text to the file and commit your changes.
24. These changes were made on the website so they will not show up in your Eclipse 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 Eclipse project, right-click on the project and select Team->Pull..
25. Your code in Eclipse should now contain the changes you made on the website.
26. 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.

3.0 Branching Code

1. 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.
2. To create a new branch, right-click the project and select Team->Switch To->New Branch...



3. Give a name to your new branch and click Finish
 - a. (Optional) The equivalent console command is : `git branch <branch name>`
 - b. (Optional) To switch branches in the console the command is : `git checkout <branch name>`
4. You can now make some changes to your code. Add a new method and commit your changes.
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 master branch, then right-click on the project and select Team->Merge... Select to merge your branch with the master branch.
 - a. (Optional) The equivalent console command is : `git merge <branch name>` (Must be done from master branch)