**Live-Debugging Java Applications**

**Introduction**

Since we have the JAR file, we can use [Visual Studio Code](https://code.visualstudio.com/) or [Eclipse IDE](https://www.eclipse.org/) to remotely debug the application and see how our input is handled in real-time.

**Remote Debugging with Visual Studio Code**

The first thing that we want to do, is install the [Extension Pack for Java](https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-java-pack) for VSCode.

Once that's been installed, we're going to decompile BlueBird (if you haven't already) using Fernflower. As a reminder, this is what the commands look like:

Live-Debugging Java Applications

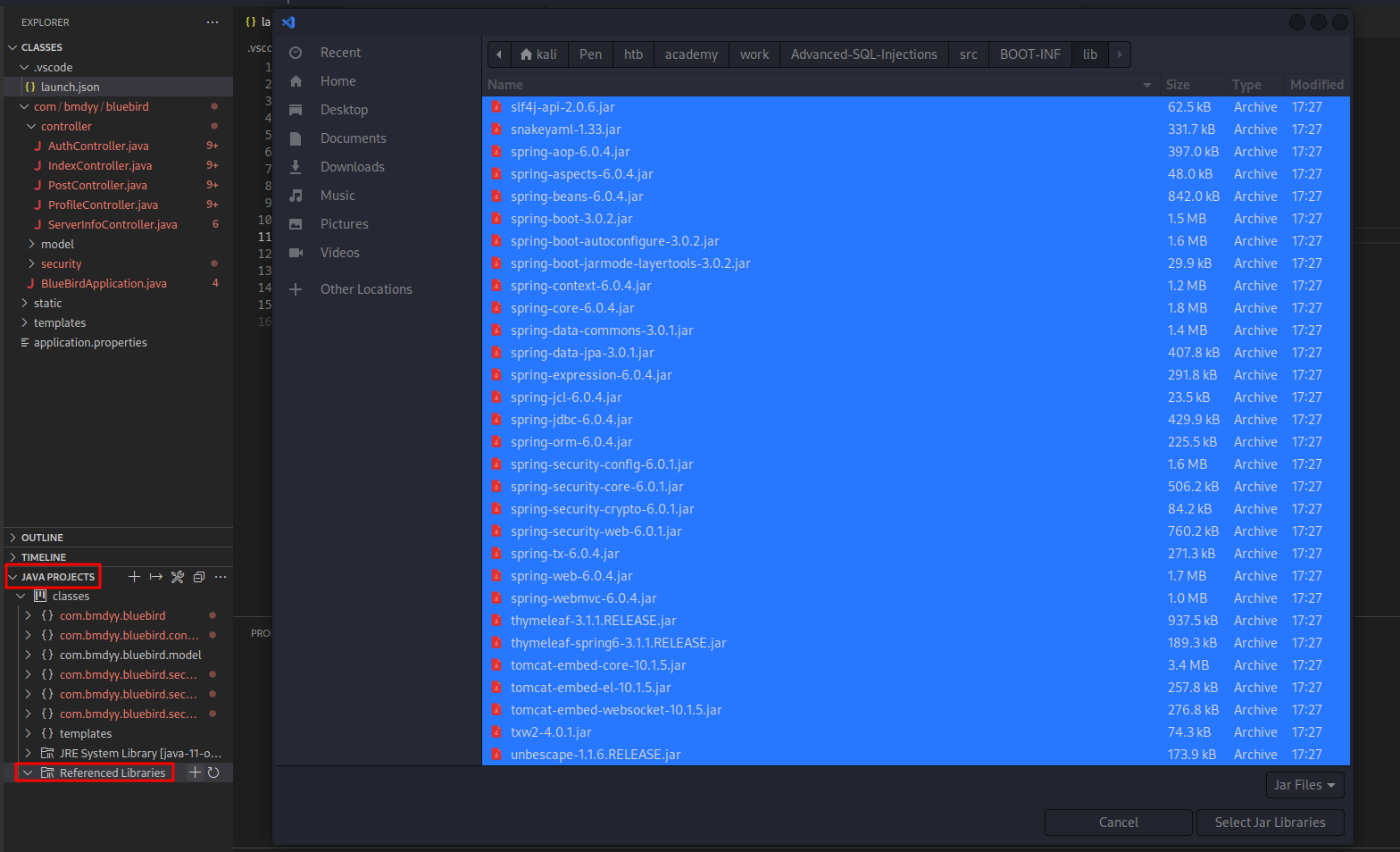
yovecio@htb[/htb]$ mkdir src

yovecio@htb[/htb]$ java -jar fernflower.jar BlueBird-0.0.1-SNAPSHOT.jar src

yovecio@htb[/htb]$ cd src

yovecio@htb[/htb]$ jar -xf BlueBird-0.0.1-SNAPSHOT.jar

At this point, we can launch VSCode and open the folder src/BOOT-INF/classes. We should have all the source files open, but a lot of lines will be underlined in red due to unresolved imports. We can fix this by navigating to Java Projects > Referenced Libraries on the lefthand sidebar, clicking the + icon and selecting all the JAR files from the decompiled src/BOOT-INF/libs folder. After this is done, the errors should disappear.



Now, you we want to hit [CTRL]+[SHIFT]+[D] to bring up the debug pane, and create a launch.json file with the following contents:

Code: json

{

"version": "0.2.0",

"configurations": [

{

"type": "java",

"name": "Remote Debugging",

"request": "attach",

"hostName": "127.0.0.1",

"port": 8000

}

]

}

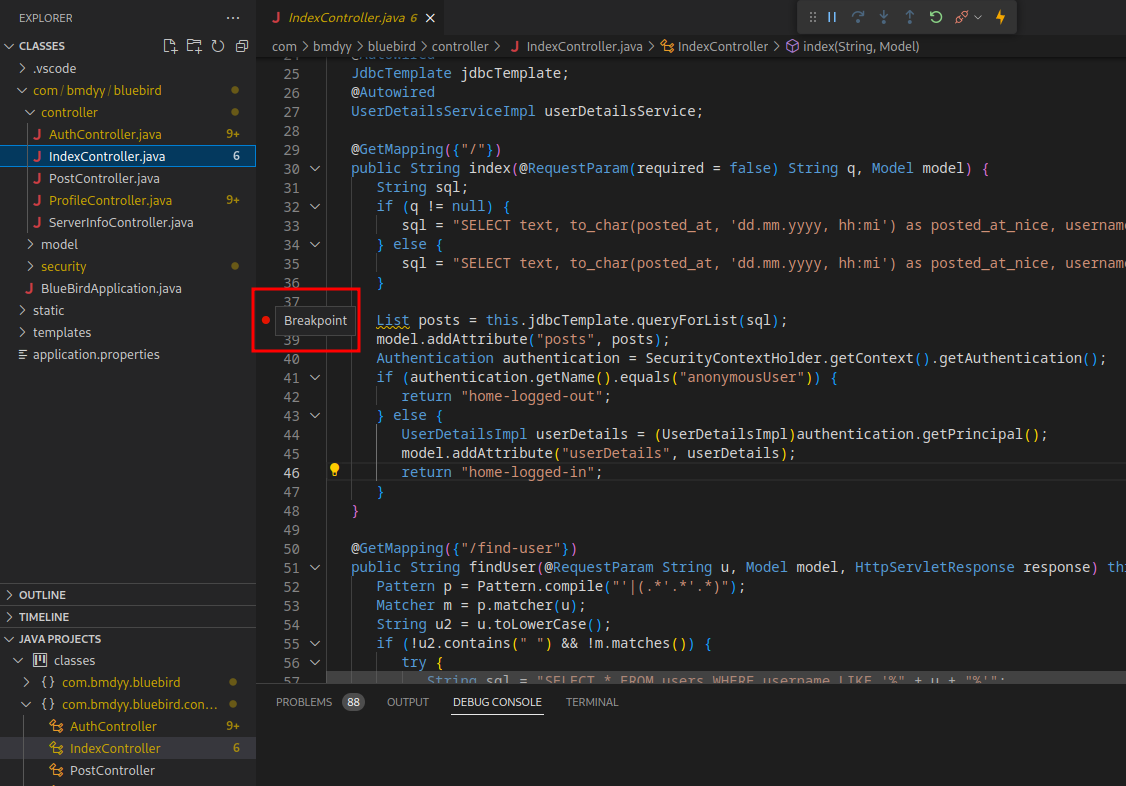
With that all prepared, connect to the VM through SSH with this command which will forward port 8000, and then run the second command to launch BlueBird in remote debugging mode.

Live-Debugging Java Applications

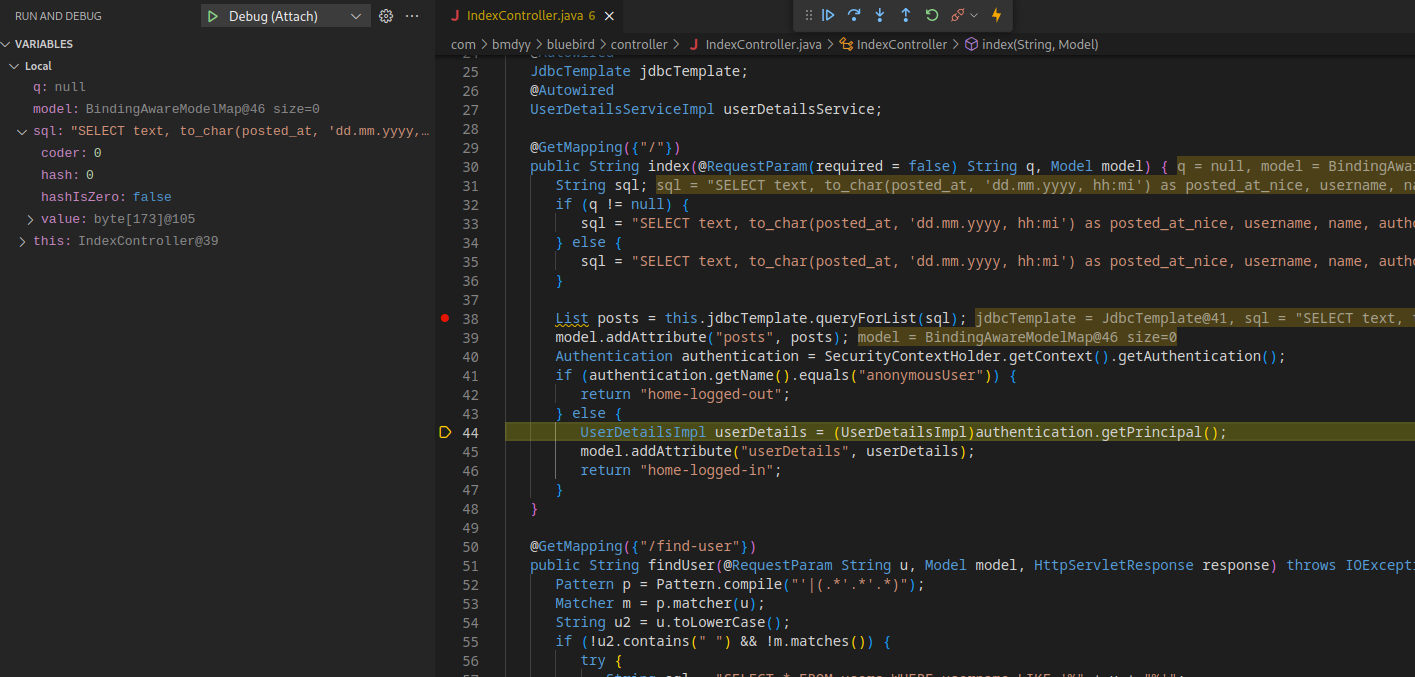
yovecio@htb[/htb]$ ssh -L 8000:127.0.0.1:8000 student@x.x.x.x

yovecio@htb[/htb]$ java -Xdebug -Xrunjdwp:transport=dt\_socket,address=8000,server=y,suspend=y -jar BlueBird-0.0.1-SNAPSHOT.jar

Finally, we can go back to VSCode and hit [F5] to start debugging. You can set a breakpoint by left-clicking to the left of a line number like this:

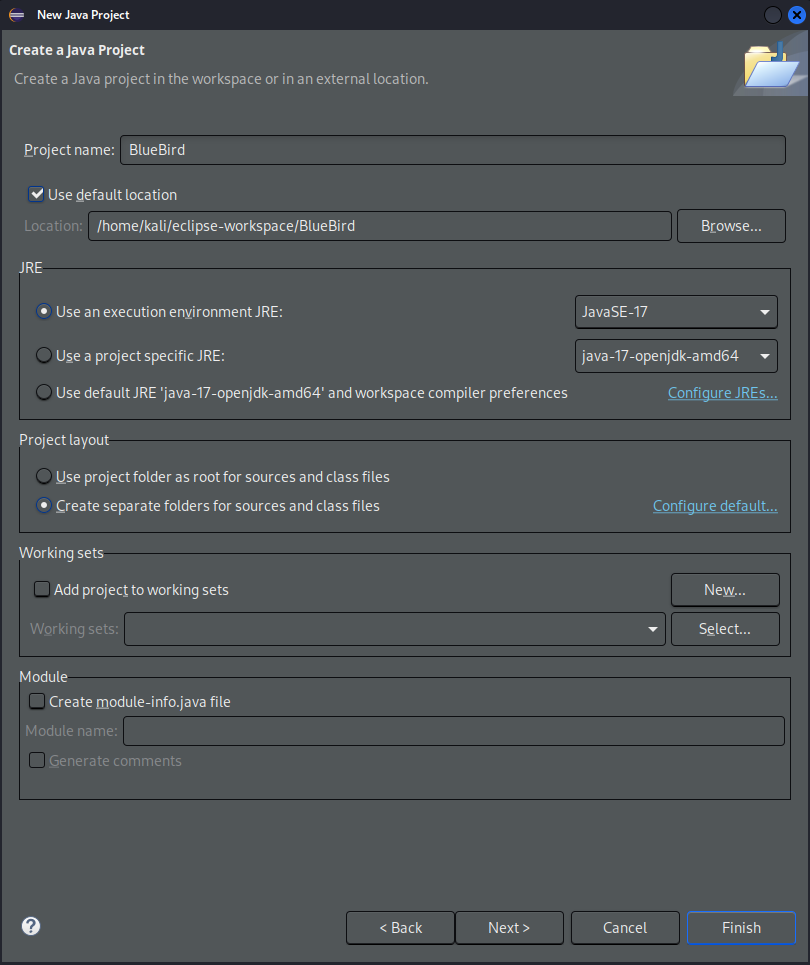


When lines with breakpoints are hit, execution will pause so that you can inspect variable values and control the program's flow by stepping through the lines of code.



**Remote Debugging with Eclipse**

Perhaps you're a fan of Eclipse. That's alright, the process is quite similar in this case. Go ahead and create a new Java Project with the following settings:

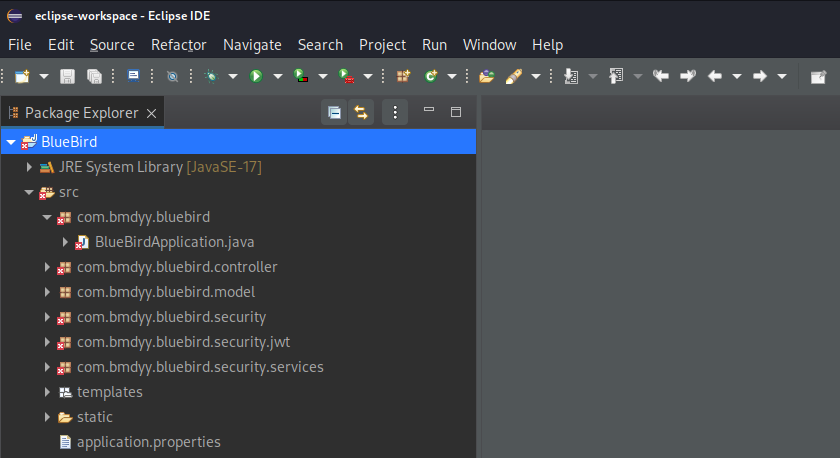


We are going to import the "source" of BlueBird into the Eclipse project, so if you haven't already, decompile BlueBird-0.0.1-SNAPSHOT.jar using Fernflower as described in the Decompiling Java Archives section. Once that's ready, go ahead and copy the contents of the decompiled classes/ folder into the src/ folder for the Eclipse project we just made.

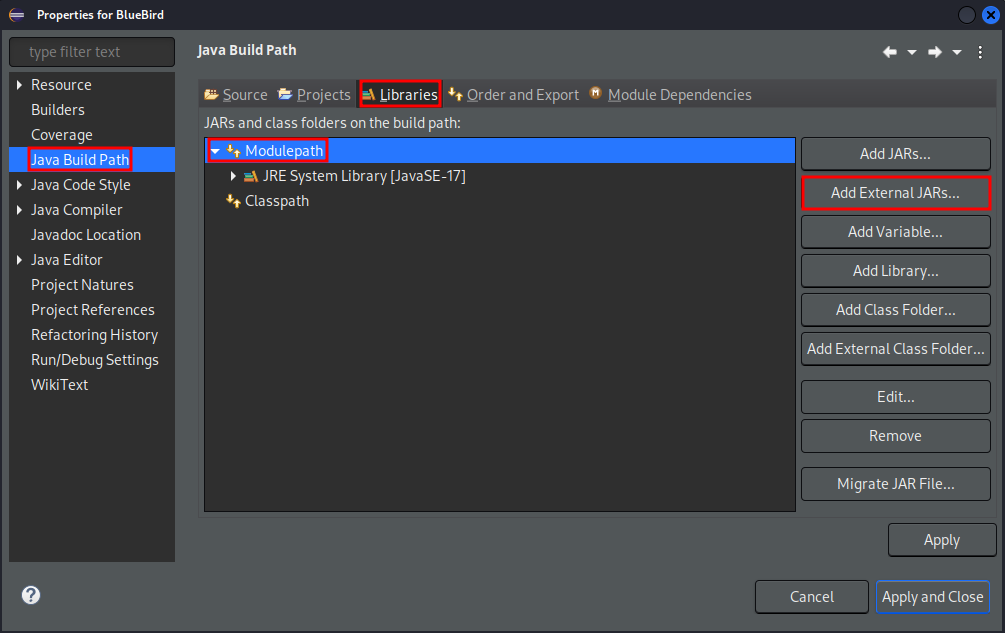
Live-Debugging Java Applications

yovecio@htb[/htb]$ cp -r src/BOOT-INF/classes/\* ~/eclipse-workspace/BlueBird/src

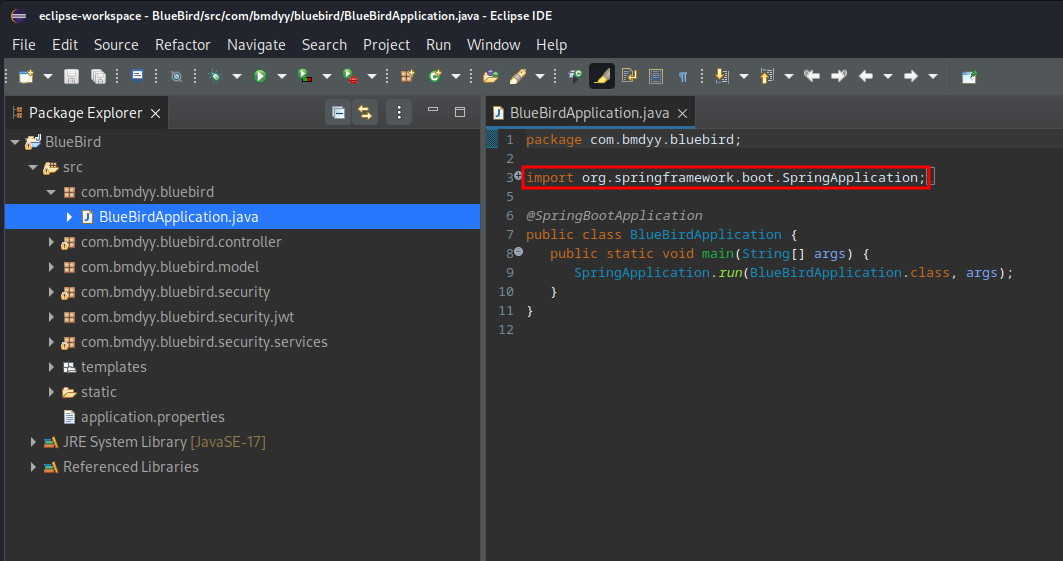
If you did that correctly, right-clicking in the Package Explorer and hitting Refresh should result in all the packages showing up (with red errors).



The reason the packages have errors is due to missing imports. To resolve this issue, we will import all the dependencies from the decompiled JAR. Go to File > Properties > Java Build Path > Libraries > Modulepath > Add External JARs and add all the JAR files from lib/ (created by Fernflower when decompiling). Click Apply and Close once imported.



If you did this step correctly, there should be no more red error signs or underlines on import statements as show in the screenshot below.



At this point we can open up a terminal and run the following command to start the JAR file in remote debugging mode.

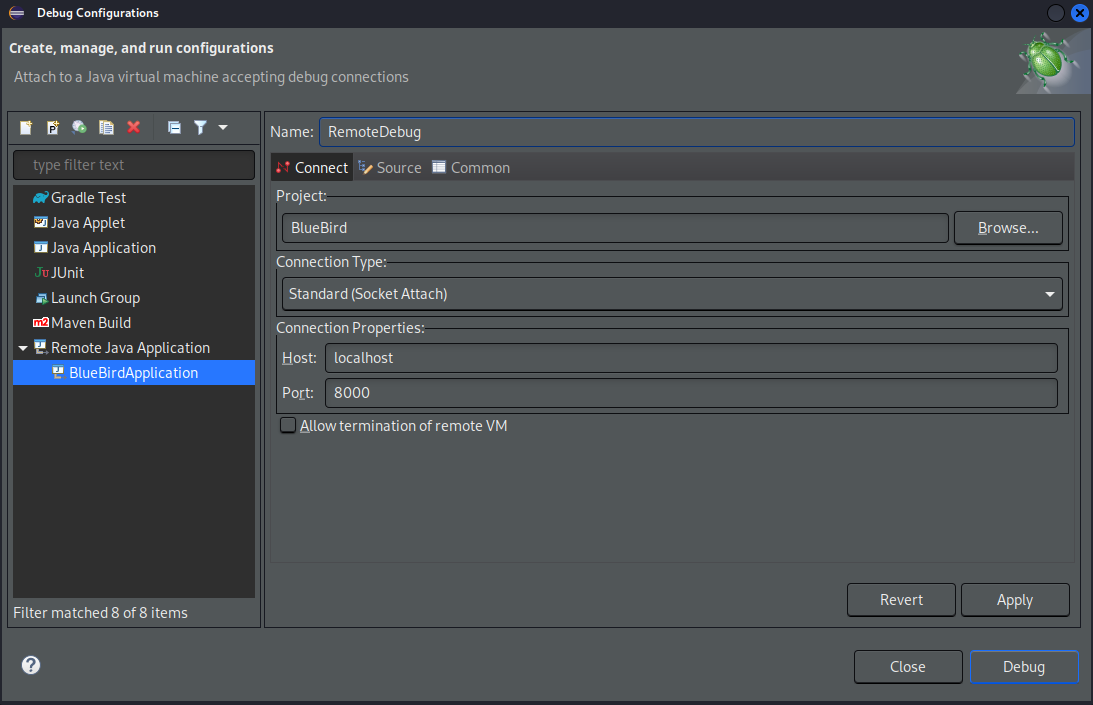
Live-Debugging Java Applications

yovecio@htb[/htb]$ java -Xdebug -Xrunjdwp:transport=dt\_socket,address=8000,server=y,suspend=y -jar BlueBird-0.0.1-SNAPSHOT.jar

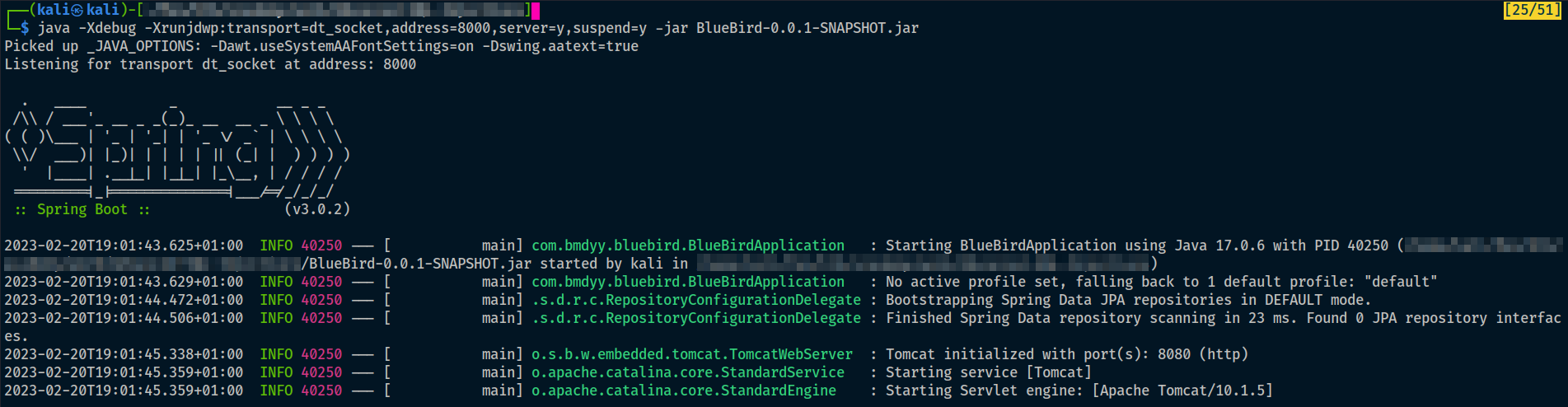
Picked up \_JAVA\_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true

Listening for transport dt\_socket at address: 8000

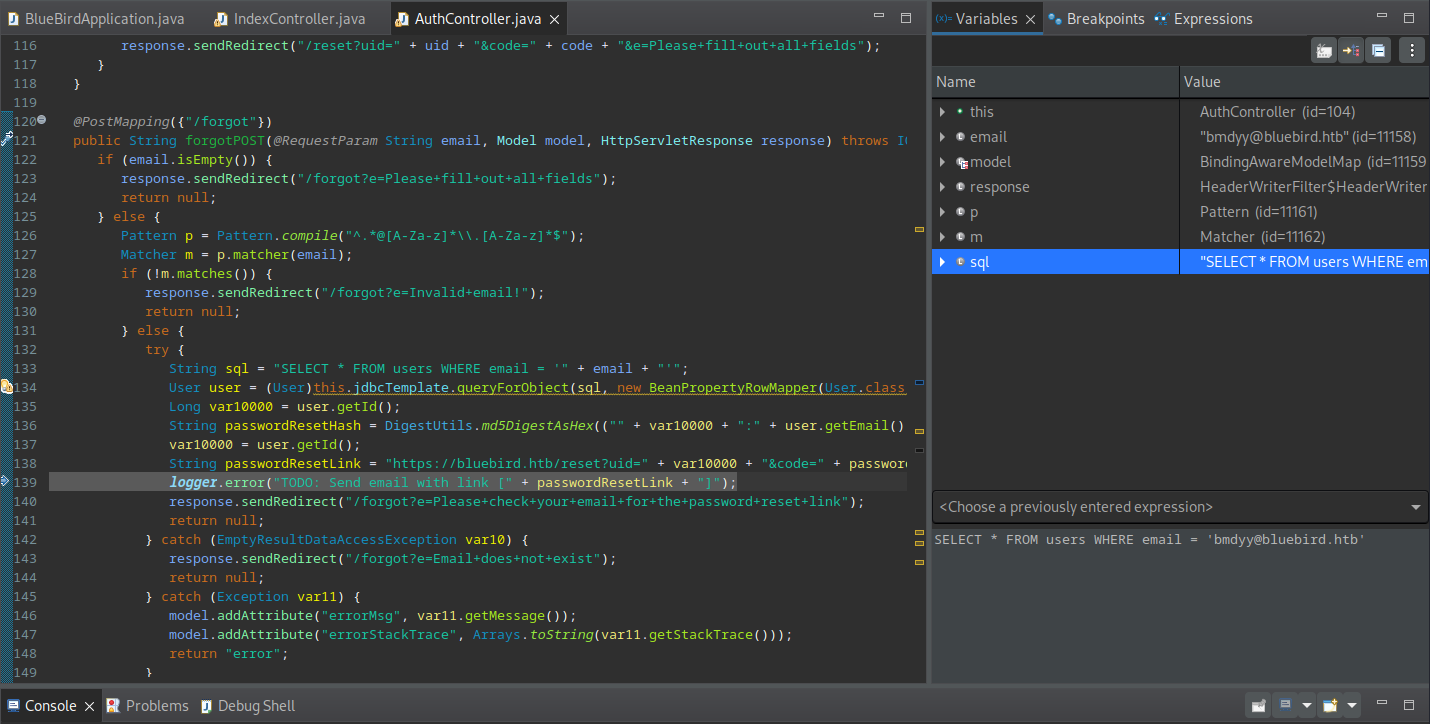
To attach to this, we need to head back to Eclipse, go to Run > Debug Configurations and create a new Remote Java Application with the following settings (should be default):



Click Apply and then Debug. If you look at the console, you should see the Spring startup log messages.



Inside Eclipse click Window > Perspective > Open Perspective > Debug to show the debugging windows. At this point we can place breakpoints in the project and live-debug BlueBird. To to so, right-click on the line number and select Toggle Breakpoint. When this line will be reached, the application will freeze and we can step through execution line by line to see what happens exactly. You can see the values of variables as they change in the Variables window (just make sure the Debug Perspective is open).



**Conclusion**

Live debugging can be a very powerful technique, but it will not always work 100% correctly since we are working with decompiled source code and not with the actual source code. In the end, everybody has their own preferred workstyles, so the best thing is to just try it out and see for yourself.