

Working with IntelliJ for CS106A

This guide is written to help you get up and running with JDK 11, IntelliJ to reduce the headache of getting to writing and running code for the CS106A class.

Introduction/Pre-requisites

The first step is to obtain a JDK, which is the “Java Development Kit.” This has the JRE (Java Runtime), needed to execute the compiled Java code, and the Java framework, which is all the capabilities that Java gives programmers automatically (such as working with files, showing graphics/text on screen, etc.)

There are regular updates to Java, but this is a case where we don’t want to use the latest and greatest. While JDK 8 is a viable choice, I recommend JDK 11 because the current Java certification exams also use JDK 11. While it may feel like this is using an “outdated” version, the reality is Java evolves faster than most businesses adapt, and the changes from, for example, JDK 11 to JDK 12, or JDK 12 to JDK 13, are generally relatively small. Keep the certifications in mind because when one comes out that uses a version of Java more recent than 11, then that’s when you can look to using a more recent Java.

You can download the official JDK 11 at:

<https://www.oracle.com/java/technologies/javase/jdk11-archive-downloads.html>

You can also use something called the OpenJDK, which you can get at:

<https://openjdk.java.net/projects/jdk/11/>

The OpenJDK is an alternate implementation of the Java language/runtime/framework. The key difference is in the licensing. For our purposes here, though, you can literally use either of the above JDK packages.

I install JDKs into “c:\java” for ease of reference. If you download the OpenJDK you’ll get a ZIP file you can simply extract into “c:\java”. I’ll be using “c:\java\<jdk path>” for rest of this document. If you do NOT place JDKs into “c:\java” then simply replace these paths in your mind to whichever paths you’re using.

Installing IntelliJ

Download IntelliJ at:

<https://www.jetbrains.com/idea/download/#section=windows>

You can also simply go to [jetbrains.com](https://www.jetbrains.com) and navigate to the IntelliJ Download.

You'll want the Community edition. This is free and doesn't lose you anything.



The screenshot shows the IntelliJ IDEA download page. On the left is the IntelliJ logo with version information: Version: 2021.3.2, Build: 213.6777.52, 27 January 2022, and a link to Release notes. The main heading is 'Download IntelliJ IDEA'. Below it are tabs for Windows, macOS, and Linux. There are two main sections: 'Ultimate' and 'Community'. The 'Ultimate' section is for web and enterprise development and has a blue 'Download' button and a '.exe' dropdown menu. The 'Community' section is for JVM and Android development and has a dark grey 'Download' button and a '.exe' dropdown menu. Both sections note they are free.

Edition	Description	Download Button	Format	Notes
Ultimate	For web and enterprise development	Download	.exe	Free 30-day trial
Community	For JVM and Android development	Download	.exe	Free, built on open source

The reason I strongly recommend IntelliJ over Eclipse is because IntelliJ is a far more consistent IDE (Integrated Development Environment). Eclipse tries too hard to do too much without keeping the average developer in mind, from a User Interface/User Experience perspective. I have never gotten someone to switch from Eclipse to IntelliJ that ever went back to Eclipse. I converted my entire team at work without even trying (I simply decided to use IntelliJ personally, and people chose to follow my example) and while my company had to pay for the Ultimate edition, I wasn't aware of any pushback from management since IntelliJ helped us do our jobs considerably better/faster.

Obtaining the CS106A Material

First and foremost, the CS106A site: <https://see.stanford.edu/Course/CS106A>

Other than a slightly modified Karel, there is nothing I am contributing to CS106A other than organizing empty projects and providing information on using IntelliJ.

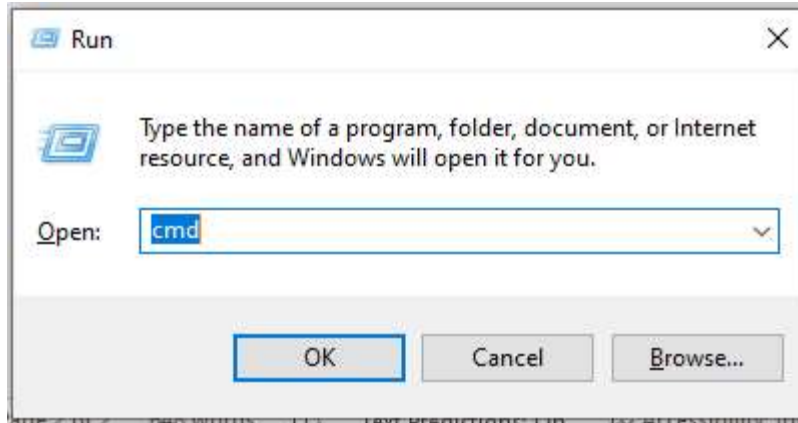
Now, if you're reading this document after cloning my github, you can disregard this next section.

The "Git" program is used for "source code control" also commonly called "version control." This is a tool that helps us track code changes over time. The full capabilities of Git are nowhere near touched for small projects like CS106A, but it still provides significant usefulness. It's also incredibly common in the software world so it should be learned by everyone at some point.

Go to following site: <https://gitforwindows.org/>

And click the download button. Installing is straightforward and you can't really go too wrong with any options you select, so if you're in doubt with what to choose on any screens, simply go with the default.

After installation, the important thing is to ensure the "git" program is in your path. This will alleviate headaches along the way. You can verify this by hitting Win-R and typing "cmd" and enter.



Then in the command shell that opens, type “git” and enter, or like I do here, type “git –version”.

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.19042.1466]
(c) Microsoft Corporation. All rights reserved.

C:\Users\...>git --version
git version 2.31.1.windows.1

C:\Users\...>
```

If “git” is NOT in your path, then you’ll want to “X” out of the above command prompt, add it to your user or system environment variables (at end of the PATH variable), then re-open the command prompt and verify it works. You cannot change your system/user path and see the changes reflect immediately in the command prompt because when you first run “cmd” it takes a snapshot of your environment variables. Keep this command prompt open.

I use a root folder named “dev” but you can use anything that makes sense to you. Again, if you see my path choices referenced here, replace them with yours.

I also use a “github” folder under “dev” to organize specifically what I get from github, but you may want to avoid doing this as it is an extra folder. My “dev” folder is quite huge, so I like this extra level of organization.

Once in “c:\dev\github” (or c:\dev or whichever folder you like) you’re going to clone the CS106A repository like this:

```
C:\WINDOWS\system32\cmd.exe

C:\dev\github>git clone https://github.com/velinar/cs106a
Cloning into 'cs106a'...
```

After this, you'll see a "cs106a" folder beneath "c:\dev\github" which contains all the empty projects for IntelliJ/CS106A as well as the extended Karel library.

The important folders in the github repository are:

```
<DIR>      lib
<DIR>      projects
<DIR>      section
<DIR>      test
```

The "lib" folder contains the extended Karel JAR file. This is a collection of compiled Java code that gives you the Karel program to work on the Karel projects from CS106A. I modified it to make it easier to specify world files, as well as upgrading it to Java 8.

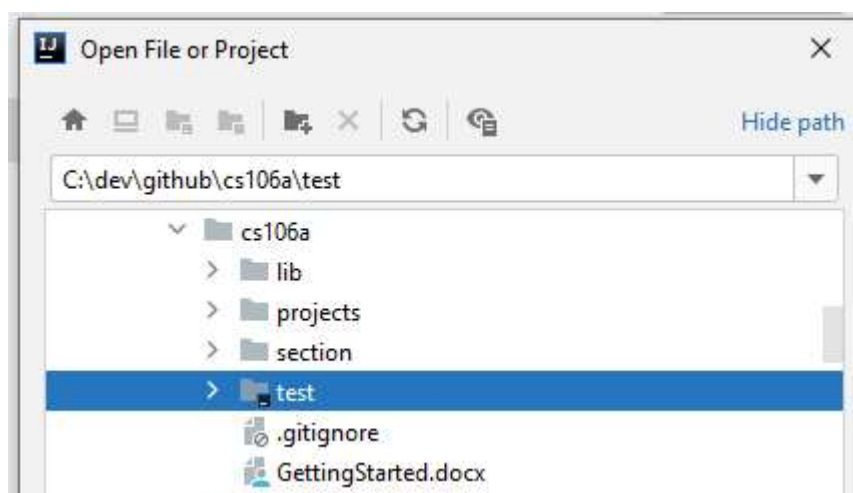
The "projects" folder contains empty/starter IntelliJ projects for the 7 "programming assignments" from the CS106A class.

The "section" folder contains similar, but for the 9 section assignments.

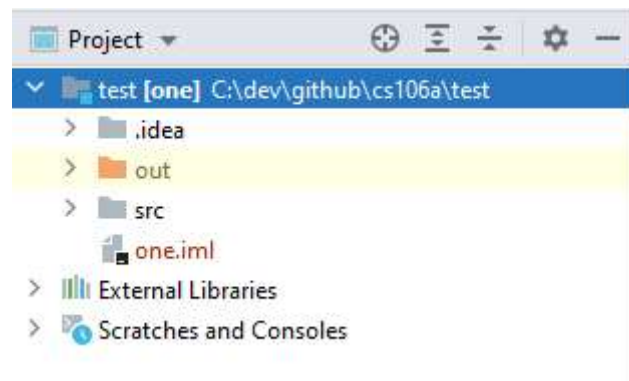
Configuring IntelliJ, Testing Karel

Next, you're going to start IntelliJ and navigate to the "c:\dev\github\cs106a\test" folder. This is a test project and once this is running for you, you'll have high confidence the other projects will work, and you can get to the hard work of learning Java via the assignments/projects.

When you start IntelliJ, click on the "Open" button, and navigate to the "test" folder. You should see a little black icon in the lower right corner, this indicates the folder contains an IntelliJ project.



Once open, you'll see something like this:

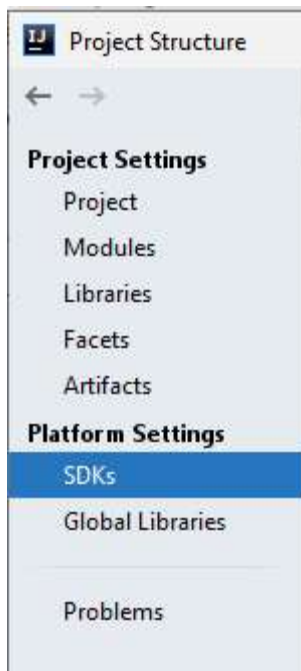


You shouldn't see the "out" folder until you build the project.

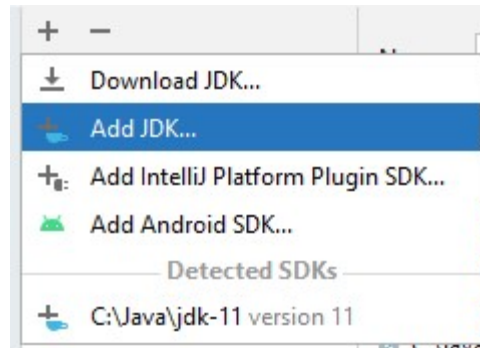
There are two other items to check/fix before the project builds. For both, we'll modify the settings for the project. You can access the settings by either doing Ctrl-Alt-Shift-S, or clicking on the "test [one]" row in the project so it is highlighted (like in the above screenshot) and hitting F4.

First, we need to let IntelliJ know about the JDK we have. IntelliJ lets us use as many different JDKs we want and let's us have control of which we tell the IDE about.

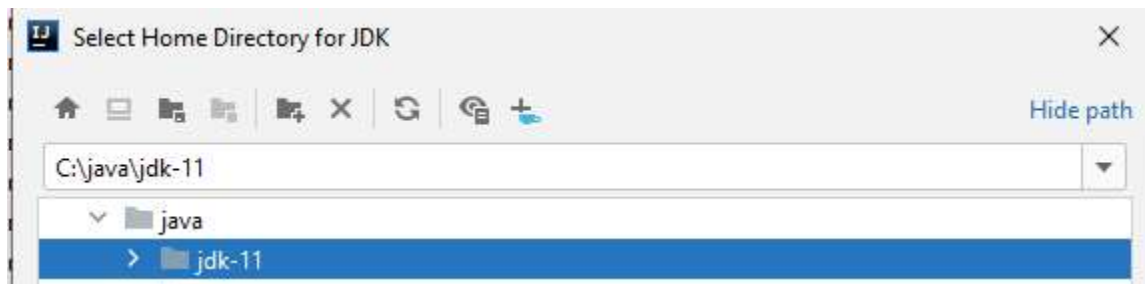
In the "Project Structure," click on "SDKs":



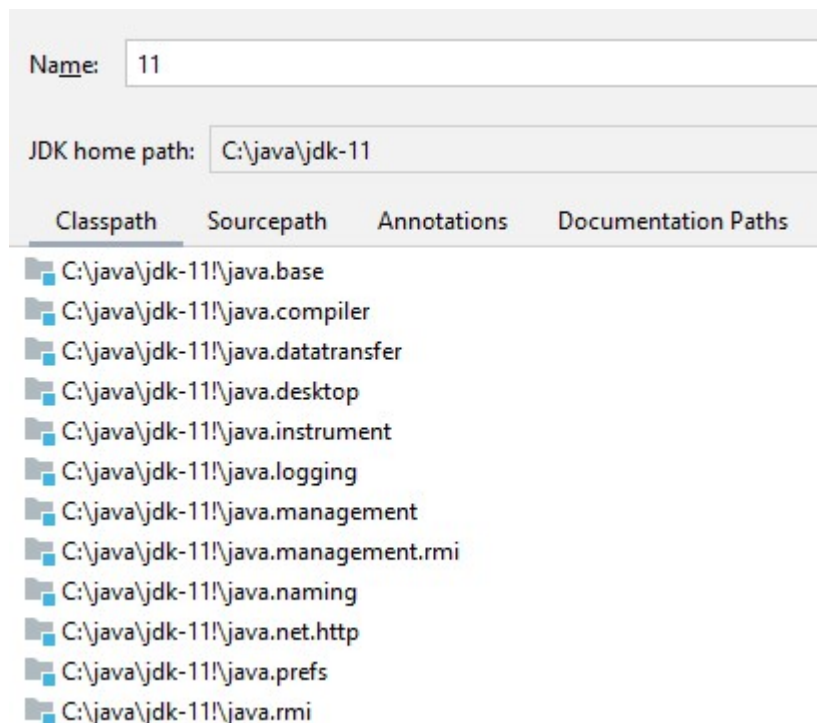
Then click on the "+" icon at top, then click "Add JDK...". You might be able to shortcut this if you see the JDK you placed into "c:\java" in the "Detected SDKs" but let's assume you don't see it there.



Navigate to the folder you have the JDK in and select it:



You'll see the JDK added to your list of JDKs now.






You can change the "Name" to whatever you want. In this case, I'll rename it to OpenJDK 11 to distinguish it from the other JDKs.

Name:

JDK home path:

Classpath Sourcepath Annotations Documentation Paths

-  C:\java\jdk-11\java.base
-  C:\java\jdk-11\java.compiler
-  C:\java\jdk-11\java.datatransfer

After this, go to the “Project” under “Project Settings” in the “Project Structure” dialog you should still have open, and make sure the “Project SDK” is set, like this:

Project Structure

← →

Project Settings

- Project**
- Modules
- Libraries
- Facets
- Artifacts

Platform Settings


- SDKs
- Global Libraries

Problems

Project name:

Project SDK:

This SDK is default for all project modules.
A module specific SDK can be configured for each of the modules as required.

 OpenJDK 11 version 11

Project language level:

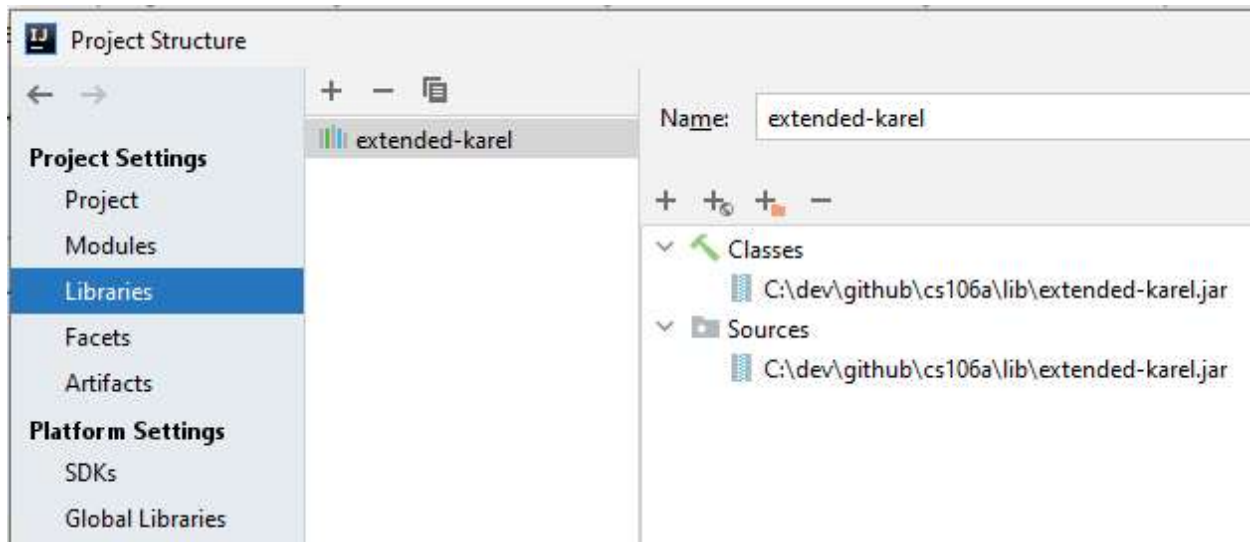
This language level is default for all project modules.
A module specific language level can be configured for each of the modules as required.

Project compiler output:

This path is used to store all project compilation results.
A directory corresponding to each module is created under this path.
This directory contains two subdirectories: Production and Test for production code and test sources, respectively.
A module specific compiler output path can be configured for each of the modules as required.

One final item to check, we need to make sure the “extended Karel” library is known by the project. It should be fine since the project is using a relative path, but it’s important to note this in case it’s broken either in the test project or a future project, or if you go to create a new project you’ll need to add this library if it needs to use Karel.

Click on the “Libraries” option in the left menu and you’ll see:



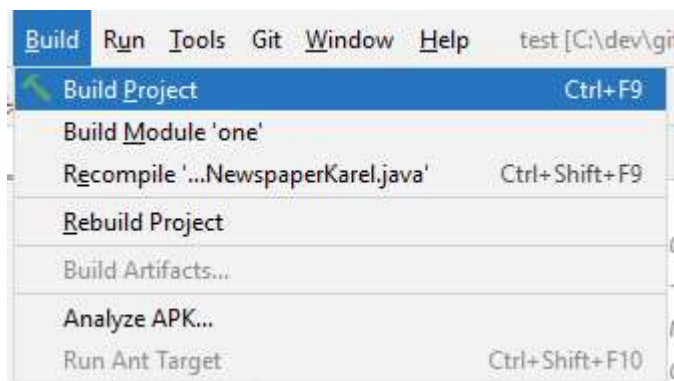
If this library is missing, click the “+” to add it. Then navigate to the “lib” folder under “cs106a” and add it.

If the path is wrong, click on the “-” to remove it, then add it again.

Now you can close the “Project Structure” dialog, everything should be set.

Building and Running the “test” Project

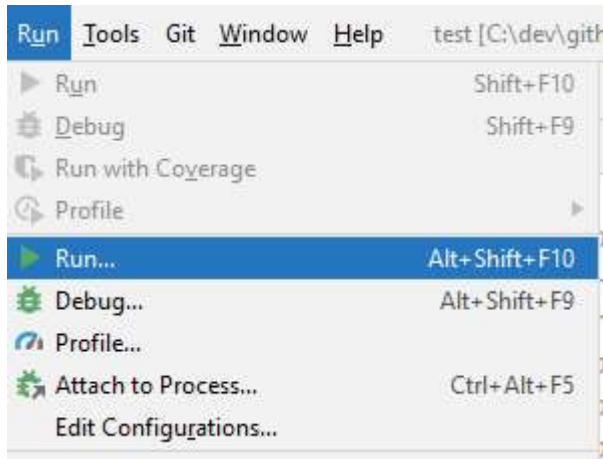
After all that configuration is done, the project should build and run. First build it. You can do this by navigating to the “Build” menu at top of the IDE:



Or you can just hit Ctrl-F9.

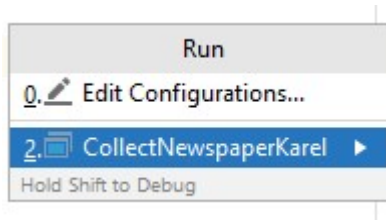
The difference between “Build” and “Rebuild” are subtle this early in learning to program, but generally “Rebuild” does a lot of extra work that is sometimes needed and sometimes not, like recompiling code that is already compiled and hasn’t changed.

Next, go to the “Run” menu at top and choose “Run”:



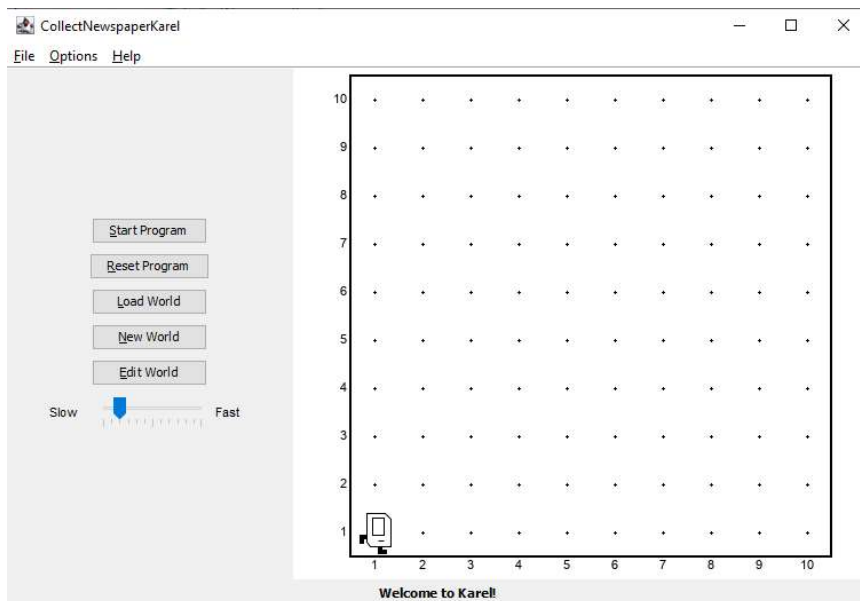
Initially we do not have any “Run” configurations set so we need to configure one for this project.

After clicking “Run...” (or you can just hit Alt-Shift-F10), you’ll see this in the middle of the IDE:



Click on the highlighted “CollectNewspaperKarel” and you should see Karel start.

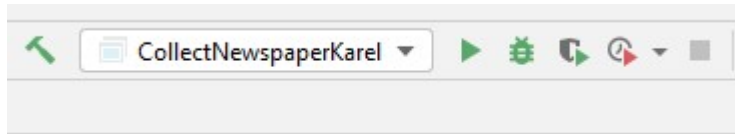
This is what Karel looks like:



However, this is not the one we want. If you look at the output in the IDE’s run output window (at bottom of IDE), you’ll see:

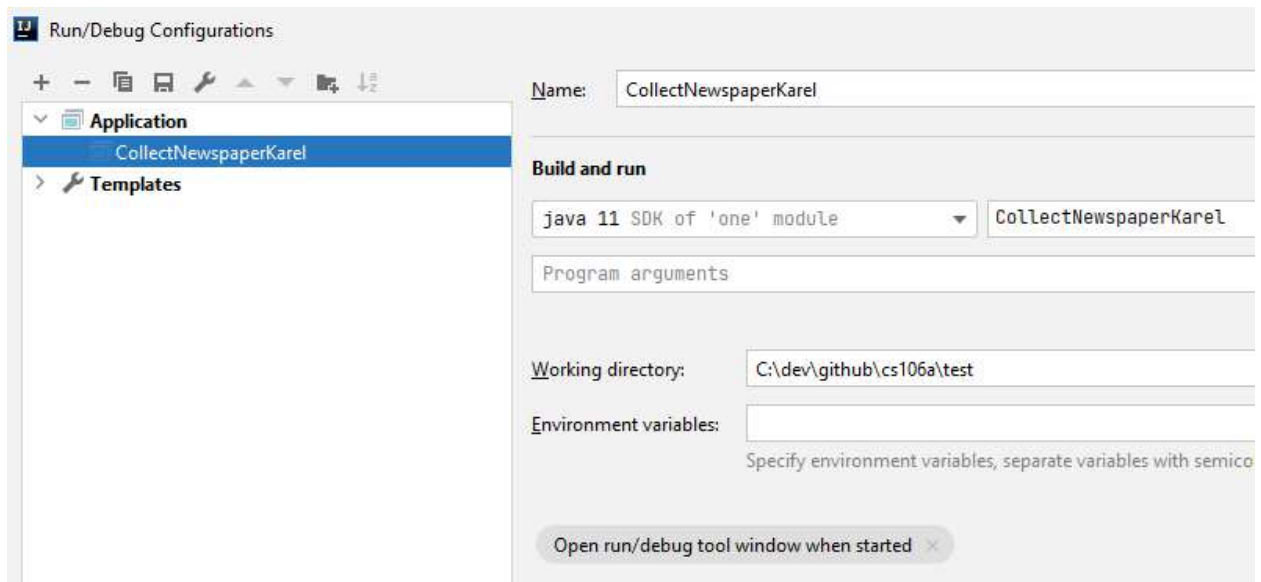
World file not found [CollectNewspaperKarel.w]

So, we have an error to fix. “X” out of the Karel window to stop program execution and go to the top right of the IntelliJ IDE.

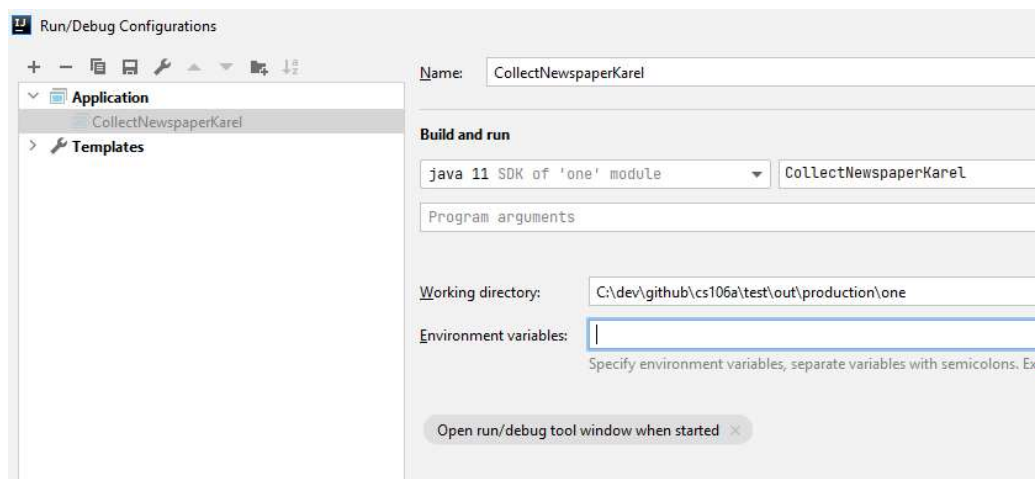


Click on the drop down and click “Edit Configurations...”

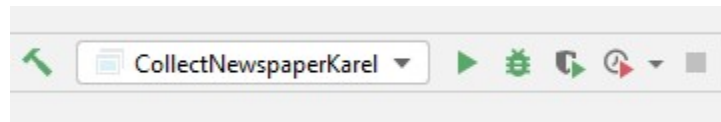
You should see this:



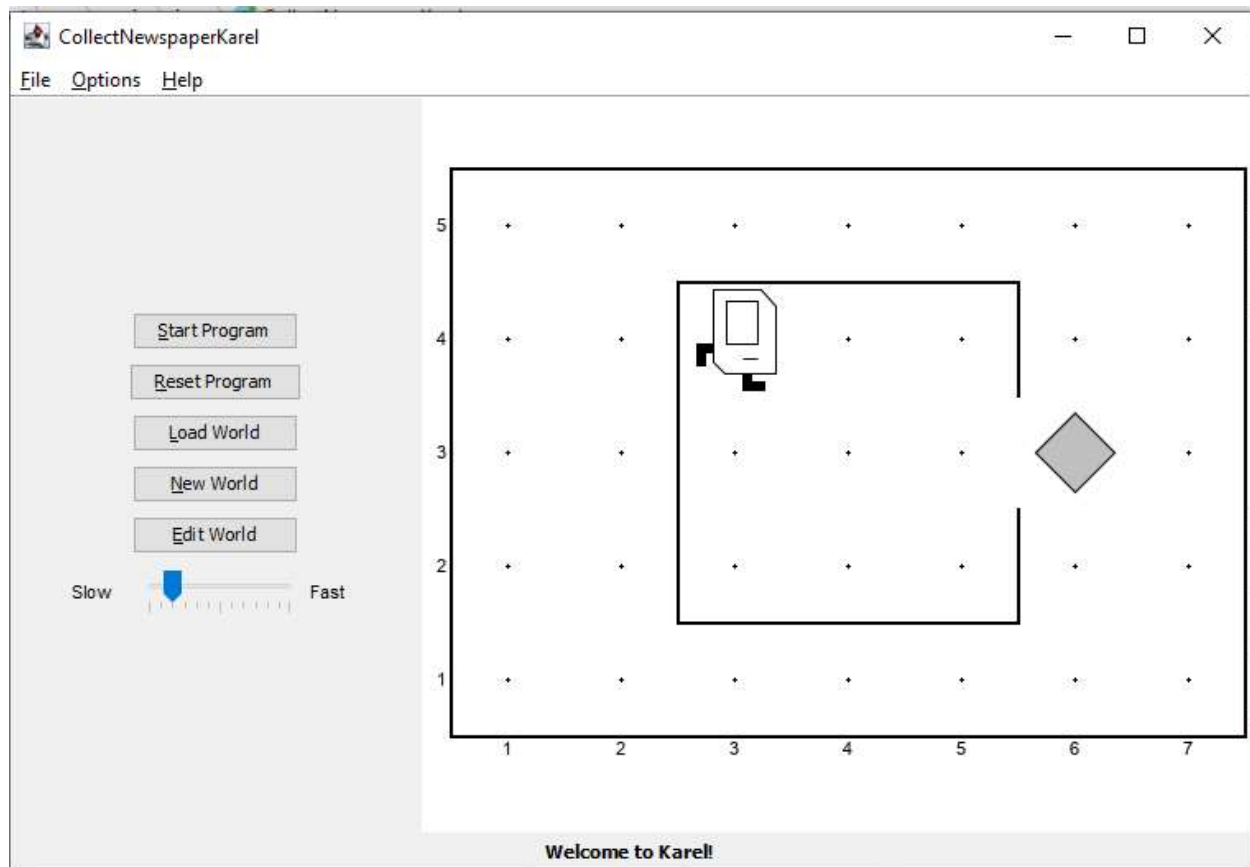
The problem here is the “Working directory” is set to the project root folder instead of the project output folder. This means the “world file” that Karel needs and is looking for is in the wrong place. Instead of moving the world file, the better way to fix this is to simply change the working directory to the output folder, like this:



Click “OK” then you can hit Shift-F10, go to the “Run” menu and click the first menu item, or you can go back to the configurations area and simply click the green arrow:



Now you should see this:



This means the world file is now loaded correctly.

Conclusion

All Karel related projects are going to operate the same way. There will be a world file under the “resources” folder in a project, this world file is copied to the output of the project build, the project execution should have a run configuration that specifies the output folder as the working directory, and if you see the world being drawn correctly you can do the assignments.

Take the time to understand not just the small steps to getting Karel projects running but the larger lessons, such as installing/managing JDKs, adding libraries to projects, how to build and run code, etc. This knowledge will help you in all Java programming work.

Further Reading

I like to separate my projects into the “sources root” and “resources root”. This helps not just in IntelliJ but in general. I haven’t gone into detail about these, but you can learn about them here:

<https://www.jetbrains.com/help/idea/content-roots.html>

And by observing how the projects are already set up.

I also didn’t go into detail about creating a Java project from scratch in IntelliJ. You have projects set up for all major class assignments and first section assignment, but beyond those you’ll need to learn how to create a project from scratch. Again, use the existing projects as examples, but you can learn about creating projects from scratch from:

<https://www.jetbrains.com/help/idea/new-project-wizard.html>

As for git, one of the best write ups on exactly what it is and how it works is from one of the founders of GitHub:

<https://tom.preston-werner.com/2009/05/19/the-git-parable.html>

It may take a little time to wrap your head around it all, but the moment it all clicks you’ll see the beauty of Git.

As for learning Git, you’ll want to consult this:

<https://book.git-scm.com/book/en/v2>

It’s the entire “Pro Git” book free online and you can also download as PDF if you want.

The most vital skill as a programmer/developer/software engineer is your ability to problem solve. This is a difficult skill to teach, and it’s the key reason why students taking an undergraduate degree in Computer Science need roughly 5-10 math classes, but it’s still learnable. If you work in any field anywhere that requires problem solving/troubleshooting/diagnosis, then you’re already ahead of the game. I usually suggest logic puzzles as a great way to exercise your problem-solving muscles. My favorites are the books by Raymond M. Smullyan, this one being the one I usually recommend first:

https://www.amazon.com/gp/product/048647027X/ref=dbs_a_def_rwt_bibl_vppi_i18

The book is called “The Lady or the Tiger?” and involves logic puzzle set ups where a prisoner must choose one of two (or more) doors, behind each of which is either a tiger or a lady. If you get this or any logic puzzle books, remember the goal isn’t to solve the problem – it’s to learn HOW to solve the problem. Answers are easy and usually included in this and other logic puzzle books. Getting used to going from problem set up to problem solution, however, is challenging and developing that muscle WILL make you a far better programmer/developer/engineer than a lot of people out there.