

Informatics 1: Object Oriented Programming

Development Tools

Adapted from [IPPO](#)* course by Paul Anderson

Volker Seeker (volker.seeker@ed.ac.uk)

Vidminas Vizgirda (s1750767@ed.ac.uk)

“Real” applications involve very large amounts of code. Most of this code is not written by the application developers themselves - it is “imported” from external libraries, or operating system frameworks, or generated by various tools. Even in a small project, this can be difficult to manage: both the internal and external code are liable to change frequently; bugs will develop and be fixed in later versions; versions of one component will be incompatible with some versions of other components; documentation will change...

Modern software development relies on a complex “toolchain” to manage this process (a toolchain is a set of tools that run on code one after another, in a ‘chain’). The tools themselves can change frequently, and different people, organisations and platforms will have different preferences. So, learning how to approach new and complex tools is an important skill.

In INF1B, we will use a small but very realistic collection of tools and libraries. You will certainly not be expected to understand all of these in depth (we don’t) but the aim is to understand the general principles, and to know how to locate and use the documentation for those features which you do need to understand in more detail.

INF1B uses the Java programming language – although the language itself is not the focus of this course. We could well have chosen to use a different language like Python, Go, or Rust but we chose Java because it works well across different operating systems, it has really good tool support, it is particularly focused on an object-oriented approach, and it is extremely common in industry. Because Java has been around since 1995, you can search for almost any error you will encounter on Google (or another search engine) and find someone else who has faced the same issue at some point, as well as advice about how to resolve it.

1 Common terminology

Java development tools typically include an integrated development environment (IDE), a compiler, and a debugger.

An IDE is a software application that provides a comprehensive environment for development, testing, and debugging Java applications. It includes a text editor, a code compiler, and a debugger, as well as other tools such as a version control system and a code profiler. Some popular IDEs for Java development include Eclipse, IntelliJ IDEA, and NetBeans (these programs are themselves written in Java). In INF1B, we will be using IntelliJ IDEA.

A compiler is a software program that converts source code written in a programming language into machine code that can be executed by a computer. In the context of Java development, the compiler converts Java source code into bytecode, which is then executed by the Java Virtual Machine (JVM).

A debugger is a tool that is used to identify and fix errors in code. It allows developers to step through their

*<https://course.inf.ed.ac.uk/ippo>

code line by line, inspect variables, and identify the root cause of any errors. Debuggers are an essential tool for Java developers, as they help to identify and fix issues in the code and improve the overall quality of the application.

In addition to these basic tools, a toolchain may also include other tools such as a version control system or a build system. These tools are used to manage the development process, track changes to the codebase, and automate the build and deployment process. We will only briefly see these in INF1B, but you can learn a lot more about them in the second year course, Informatics 2: Software Engineering and Professional Practice (INF2SEPP).

2 The Tools

All of the software required for this course is pre-installed on the Informatics [DICE](#)¹ systems. It is also freely available for you to install on your own machine. Whichever you plan to use, you should read section 3 which explains how to configure the software on DICE or install it on the various platforms.

2.1 Java

To develop Java programs, you will need the Java Development Kit (JDK). This is a set of tools including the Java compiler and runtime, which you will need to turn code into programs that can be executed.

2.2 IntelliJ

It is perfectly possible to create Java programs with a simple text editor (like Gedit or Notepad) and compile and run them from the command line. You will likely have already tried this with Haskell in INF1A. However, most programmers choose to use an IDE (*Integrated Development Environment*). This provides a text editor and file browser with specific support for code development – for example, displaying the location of errors in the source file, code completion, compiling, running and testing from the interface, etc.

For INF1B, we have decided to use IntelliJ and we strongly recommend that you use this for Java development in general – it is a good environment with support for many other tools such as Git, JUnit and Gradle. Using some IntelliJ features will be required for course activities, so it is compulsory for this course.

When you first start IntelliJ, it will ask some initial setup questions. If you are unsure what to choose, we recommend that you accept the default values - these can be changed later if necessary.

IntelliJ is a professional IDE which can be rather overwhelming for beginners. If you get lost – that’s okay, feel free to ask questions like “how to do X with IntelliJ?” on Piazza. We won’t need to learn everything that IntelliJ can do for this course, so don’t worry about understanding every button, menu, and setting.

2.3 JUnit

It is important to check that the code you write works as intended, to test it. When developing a small program, you can usually check if it works with different inputs. But whenever you make a bigger change, you have to try all the inputs again. With larger pieces of software, testing with the same inputs again and again between every change can take extremely long. At this point, we want the computer to take up the burden of running the programs and checking they still work. We do this by creating automated tests.

¹<http://computing.help.inf.ed.ac.uk/what-is-dice>

A common tool for creating automated tests in Java is JUnit. JUnit allows methods and classes to be tested in isolation, letting the programmer prove individual components of code work as expected, and giving them the confidence to use these components.

It is not the goal of this course to teach you how to come up with your own JUnit tests but rather you should be able to take a given set of tests, execute them for your own implementation and interpret the results. This can be done using an IDE such as IntelliJ or the command line. We will cover some more details about JUnit during the lectures.

We use version 5 of the JUnit framework for this course. IntelliJ can easily set it up on the go when a project includes JUnit code (we will see this in a later tutorial), so you don't need to set anything up for this.

2.4 Git

It is very common to be dealing with multiple versions of an application. These may be for different hardware/customers, or they may be development versions where several people are working on different aspects of the code. It is also *extremely* useful to be able to return to an older version of the code at any time – for example, if a bug is discovered, it is useful to be able to go back and find out what was changed at the time the bug was introduced, or to be able to discard some unsuccessful changes and return to a previous version.


A *version control system* (VCS) keeps track of changes and supports rollback to previous versions, and merging of independent changes into a new version. Git is a very common VCS which is freely available on the [Git²](https://git-scm.com/) website.

We would recommend that you consider using Git and we have a tutorial with corresponding self-study exercises at the end of the course. However, it is optional. Especially if you have not programmed before, you may be finding this list overwhelming, and it's okay if you decide that this is not essential for now.

3 Installation & Configuration

If you plan to use DICE, you will need to perform a small amount of configuration to ensure that you are using the correct versions of the tools (see 3.1). Even if you plan to use just your own machine, we would recommend that you do this anyway, so that you can run your code on the DICE systems in case you have problems with your own machine.

Installing the software on your own machine is usually straightforward. We have provided some instructions below for the various platforms. We can't spend a disproportionate amount of time helping with individual installations, but we will try to help with any problems, and the Piazza forum is usually a good source of advice. If you do have problems and you manage to find a solution, please let us know, so that we can update the documentation.

 If you are using your own machine ...
Make sure that you have good backups. It is also a good idea to make sure that you can run your software on some alternative system (e.g. DICE) as well, because failure of your own machine will not be taken into account for coursework submissions.

3.1 DICE setup

JDK, IntelliJ, and Git come pre-installed on DICE.

²<https://git-scm.com/>

You can run IntelliJ from the command line with the command:

```
$ ideaIC
```

☞ If you open IntelliJ without a project folder on DICE and use the 'import' or 'open project' options, it will most likely freeze. This happens because IntelliJ tries to be helpful and show all files and folders to you when you search for a place where to open a project, except on AFS (the file system that DICE uses) there are probably millions of student folders... To close the frozen IntelliJ, you can run `xkill` which will turn your mouse cursor into a big X of death and then click on IntelliJ to terminate the frozen process.

To work around this issue, you can open terminal, `cd` to the directory where you want to work and then open IntelliJ in that directory by running:

```
$ ideaIC .
```

(Don't forget the `.` at the end, which means "current directory")

We will also need the *JetBrains Academy* plugin for IntelliJ. You can get it by the following steps:

1. Open IntelliJ and its Settings/Preferences dialog (which can be found under `File > Settings`, or similar, or via the keyboard shortcut `Ctrl+Alt+S`), then select Plugins.
2. In the Plugins dialog, switch to the Marketplace tab.
3. In the dialog that opens, search for JetBrains Academy.
4. Click Install.
5. Click OK in the Settings dialog to apply the changes, and restart IntelliJ IDEA if prompted.

3.2 Own machine setup

To follow along the course, you will need to download and install the JDK and IntelliJ.

JDK is freely available from the [Oracle³](https://www.oracle.com/java/technologies/downloads/) website for you to install on your own machine (although you will need to create an account to download the installer). You can also download a zip version from [OpenJDK⁴](https://jdk.java.net/archive/) without an account, but then you will need to install it manually.

There are many JDK versions available. We recommend choosing the latest stable version supported by your operating system (JDK version 21 for Windows, Mac, and Linux at the time of writing), but any version ≥ 11 will do – we won't be using the newest features in our course for now.

The "Community Edition" of IntelliJ is freely available from the [JetBrains⁵](https://www.jetbrains.com/idea) website for you to install on your own machine. There is a specialised version ("IntelliJ IDEA for Education") that already includes the *JetBrains Academy* plugin pre-installed.

You can also get IntelliJ Ultimate for free through the [JetBrains Education programme⁶](https://www.jetbrains.com/idea/education/) – all you need to do is verify your university email address, however this is optional. The ultimate version has some nifty features like generating diagrams from code and code profiling, but we won't be using them in INF1B.

³<https://www.oracle.com/java/technologies/downloads/>

⁴<https://jdk.java.net/archive/>

⁵<https://www.jetbrains.com/idea>

⁶<https://www.jetbrains.com/community/education/#students>

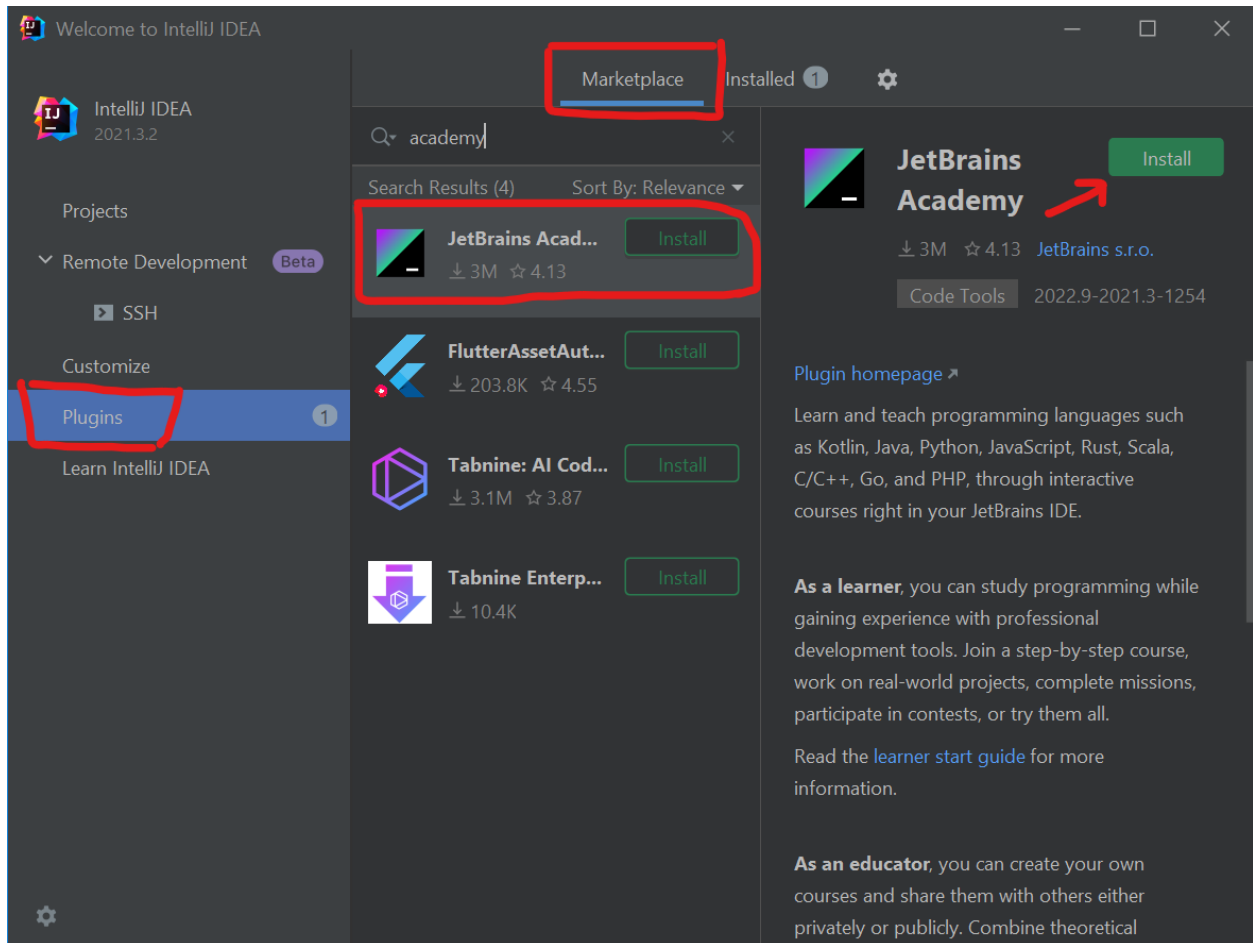


Figure 1: Screenshot showing the JetBrains Academy plugin in the IntelliJ plugin marketplace

If you use the Community or Ultimate editions of IntelliJ, follow the same steps as for DICE to set up the *JetBrains Academy* plugin.

If you need help setting up the JDK or IntelliJ, in the first instance, see the relevant official documentation: [for JDK⁷](https://docs.oracle.com/en/java/javase/19/install/overview-jdk-installation.html) or [for IntelliJ⁸](https://www.jetbrains.com/help/idea/installation-guide.html#standalone) (under “Standalone installation”). If still stuck, then another good place to look is YouTube (just search for something like “IntelliJ IDEA installation 2024” or whatever tool / year you need). If that still doesn’t cover it, then you can ask any of the demonstrators during lab sessions to help you out.

4 Testing the Tools

Now that you have installed the tools on your own machine (or you are just planning to use DICE), you should look at the **Getting Started** document which has a short exercise to make sure that the tools are working correctly, and that you understand how to use them.


⁷<https://docs.oracle.com/en/java/javase/19/install/overview-jdk-installation.html>

⁸<https://www.jetbrains.com/help/idea/installation-guide.html#standalone>


5 Appendix: command-line Java setup

You can check your default Java version as follows via the command line:

```
$ java --version
openjdk 14.0.2 2020-07-14
OpenJDK Runtime Environment (build 14.0.2+12-Ubuntu-120.04)
OpenJDK 64-Bit Server VM (build 14.0.2+12-Ubuntu-120.04, mixed mode, sharing)
```

 Note that we are using '\$' to represent the terminal prompt (a message that usually shows which directory you are currently in); this is not something you are meant to type.

A word about versions It can be surprisingly difficult to assemble a toolchain in which the versions of each tool are compatible with all of the others. The versions of the tools that we are recommending for Inf1B have been carefully chosen to be compatible. Where possible they are also LTS (long-term-support) versions which means that they are less likely to be upgraded to some incompatible version at an inconvenient time.

 If you are using your own machine and run into issues... Make sure that you search online for guidance for the specific version of JDK (or other relevant tool) that you have installed. Instructions and usage of different versions can vary, the documentation may not match, and you may get unpredictable errors.

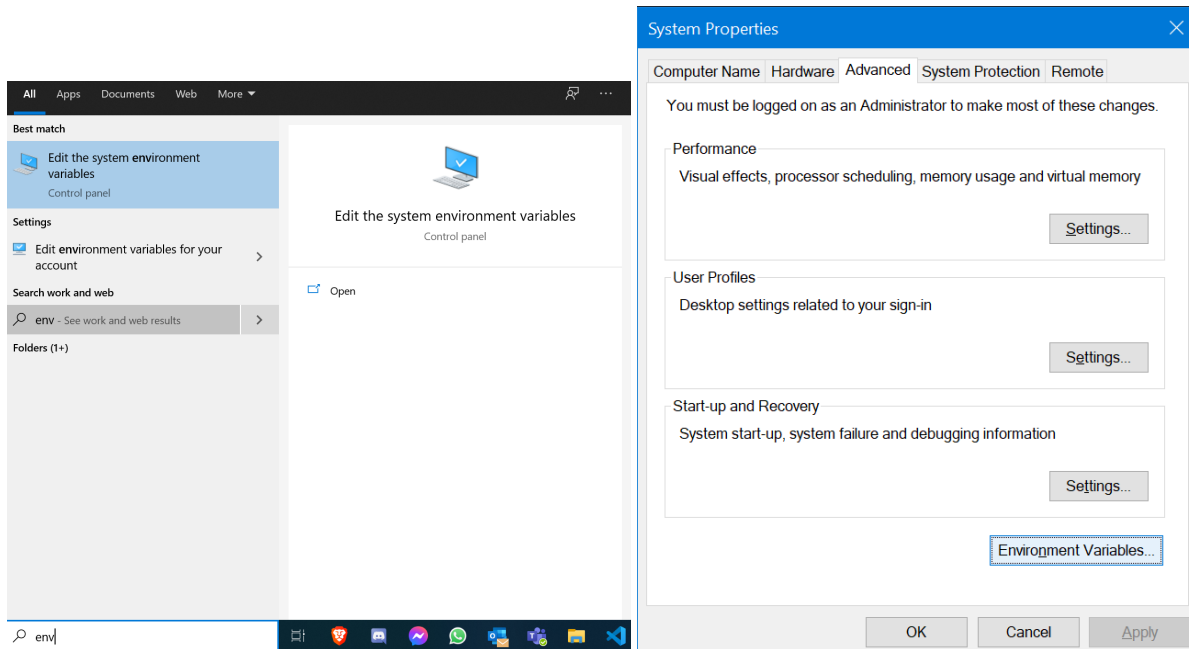
Unfortunately, it is not always easy to remove applications if you have previously installed a different version.

If you had previously installed a different version of Java on your machine, and this other version comes up in the command line, you can choose to either uninstall the other version or setup multiple versions of Java.

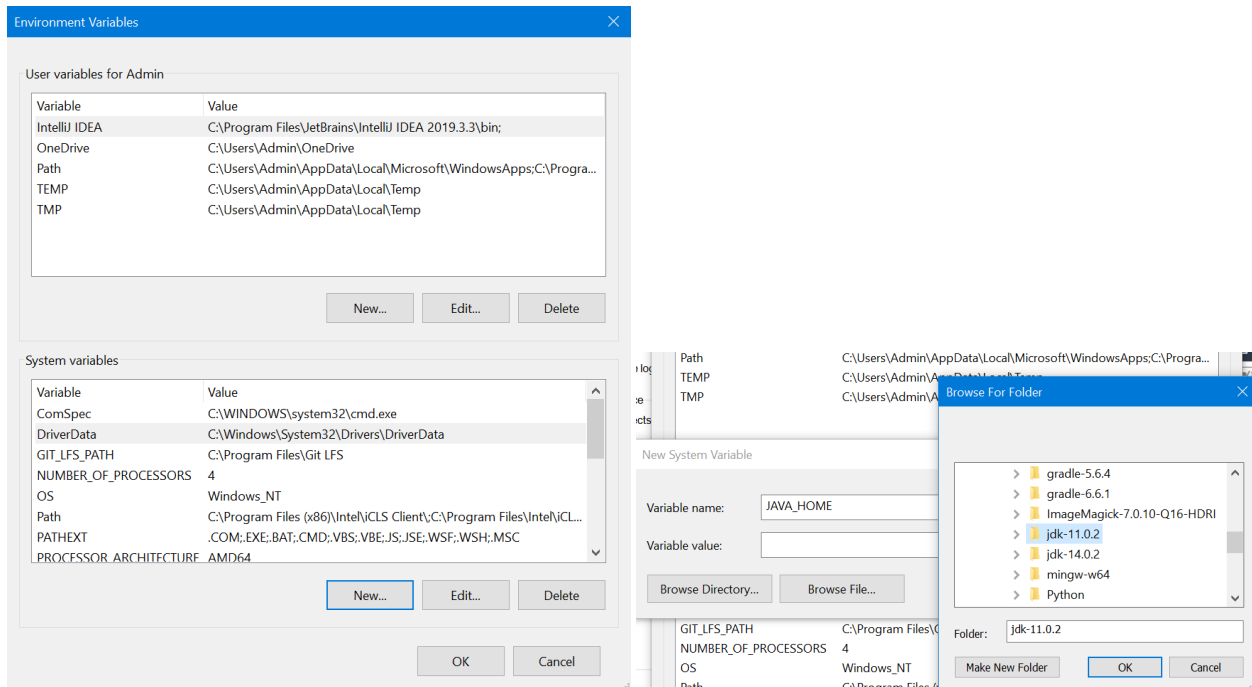
5.1 Windows environment variables

If you already have a different version of Java installed, you can download and install the newest version of Java (in the example below I use JDK 14, but the same instructions should work for any other version). The catch is that you will need to change two system environment variables to tell Windows which Java version to use.

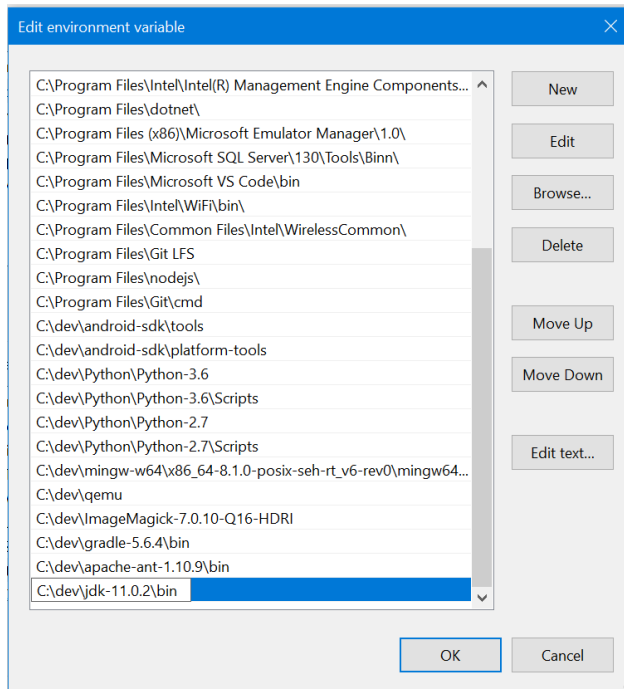
First, open the system environment variable settings:



Then, look for the JAVA_HOME variable under system environment variables and click "Edit". If the variable is not there, create a new one by clicking "New" and typing in JAVA_HOME as the variable name. Then for the variable value, click browse to directory, and select the folder where you installed Java - in the screenshot below the folder is called jdk-14.0.2 but it might be called differently on your system.



Finally, edit the Path system variable. Remove any lines that reference other versions of Java, then add a new line and browse to the bin sub-folder under the folder where you installed Java 14. The result should look similar to this:



Finally close any open terminal windows, then open a new one and check that Java 14 is correctly detected by running:

```
$ java -version
```

If you need to change back to another Java version, modify the `JAVA_HOME` and `Path` environment variables to reference the other Java installation folder.

5.2 MacOS or Linux

On MacOS or Linux, Java development tools can be installed from the command line.

We can use `SDKMAN!`, an SDK version manager to easily get the correct version of Java and help to switch between versions. The examples below show what commands you could use in terminal to install OpenJDK and check the installed Java version:

```
$ curl -s "https://get.sdkman.io" | bash
$ sdk install java 14.0.2-open
Do you want java 14.0.2-open to be set as default? (Y/n): <Press Enter to select Yes>
$ java -version
```

Multiple Java versions `SDKMAN!` should take care of configuring multiple Java versions and you can switch between them with the commands `sdk use` (to change the Java version used in the current terminal window) or `sdk default` (to change the Java version in the current and all new terminal windows):

```
$ sdk use java 14.0.2-open
$ sdk default java 14.0.2-open
```

If `SDKMAN!` fails to set up Java 14 to work for you, you can try some debugging steps in the advanced section at the end of this document.

IntelliJ IDEA can be downloaded and installed from the JetBrains website for MacOS.

On Linux, it can either be downloaded from the JetBrains and manually set up, or it can be installed in one step with Snappy:

```
$ sudo snap install intellij-idea-community --classic
```

6 Appendix: Advanced Linux Debugging Steps

For Linux machines and users that cannot find the right version of Java (Oracle Hotspot or OpenJDK) they want in their system's package repositories, alternative options described below are available.

Please read the entire section before attempting any of the steps in it, as there are several suggestions and complementary approaches to choose from.

Location Download and unpack the JDK archive in your desired location. Choosing a system directory like `/opt/local/` or `/usr/lib/jvm/` has the advantage that the changes are accessible system-wide and

thus to other users.

Setting the JAVA_HOME environment variable This is a convenient way for users and programs to access the preferred JDK or JRE installation. This uses the root directory (the location you chose in the previous step plus the JDK directory name) of the installation.

There are several options to make JAVA_HOME available on every login by adding the line

```
export JAVA_HOME=/usr/lib/jvm/java-14-openjdk-amd64/
```

in one of:

- Your user's profile file

```
~/profile
```

- The system's environment file (where available)

```
/etc/environment
```

making sure it is sourced in your ~/.bashrc file (usually it already is), if you are using bash.

- A custom and separate profile file (this is preferable than 1.)

```
/etc/profile.d/java_paths.sh  
chmod a+x </tt>/etc/profile.d/java_paths.sh
```

Sourcing any of the above files you chose to adopt, e.g.

```
$ source ~/.profile
```

will make the variable available immediately in the current shell.

Another simplification that you might want to consider prior to the above steps, is to use a symbolic link to the location of the JDK, e.g.

```
$ sudo ln -sf /usr/lib/jvm/java-14-openjdk-amd64/ /usr/lib/jvm/jdk
```

and use that for the contents of JAVA_HOME variable. This allows moving to another version of the JDK by simply creating a new symbolic link with the same name but to the new directory.

Setting the PATH environment variable Setting up the PATH environment variable provides access to the various Java executables, namely the javac compiler and java application launcher.

You can add the line

```
export PATH=/usr/lib/jvm/java-14-openjdk-amd64/bin:$PATH
```

to any of the files mentioned in the previous section. If you choose the symbolic link method use

```
export PATH=/usr/lib/jvm/jdk/bin:$PATH
```

Alternatives mechanism Otherwise, if your system uses the `update-alternatives` mechanism (e.g. Ubuntu, Fedora or other Debian-based distributions), you could configure that instead of setting your `PATH` environment variable directly. To read more about this mechanism, please see [here](#)⁹.

If executing `update-alternatives` gives you a help screen on the terminal, you can proceed with this method.

First, since each distribution can vary a lot on how it uses the alternatives mechanism, you need to establish what is already available for Java. For example, on DICE, we can execute:

```
$ update-alternatives --list java
/usr/lib/jvm/java-11-openjdk-amd64/bin/java
/usr/lib/jvm/java-14-openjdk-amd64/bin/java
/usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java
```

In the above case, there are already multiple `java` alternatives set up. We can check which directory the default version points to with:

```
$ update-alternatives --config java
There are 3 choices for the alternative java (providing /usr/bin/java).

  Selection    Path                                                    Priority  Status
-----
*  0            /usr/lib/jvm/java-14-openjdk-amd64/bin/java 1411    auto mode
   1            /usr/lib/jvm/java-11-openjdk-amd64/bin/java 1111    manual mode
   2            /usr/lib/jvm/java-14-openjdk-amd64/bin/java 1411    manual mode
   3            /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java 1081    manual mode

Press <enter> to keep the current choice[*], or type selection number:
```

Then press enter to just leave the default choice as it is.

To add our newly extracted `java` and `javac` binaries execute:

```
sudo update-alternatives --install "/usr/bin/java" "java"
"/usr/lib/jvm/jdk-14-openjdk-amd64/bin/java" 100
```

```
sudo update-alternatives --install "/usr/bin/javac" "javac"
"/usr/lib/jvm/java-14-openjdk-amd64/bin/javac" 100
```

The choice of 100 as priority is random here, but you can choose anything that is not in the list already and adjust based on what values are already present. Then, execute again:

```
sudo update-alternatives --config java
```

and choose the `java` executable that we just configured. Confirm that the right version has been set up by executing:

```
java --version
```

⁹<http://manpages.ubuntu.com/manpages/trusty/en/man8/update-alternatives.8.html>