Compiling & Running from the Command Line

Contents

[1 Using Eclipse to Write Java Programs to Read/Write Text Files 1](#_Toc119321165)

[2 Change the Default Location that Eclipse uses to Look for a Text File 2](#_Toc119321166)

[3 Example 1 – Portable Approach for Reading a Text File 3](#_Toc119321167)

[3.1 Eclipse Development 3](#_Toc119321168)

[3.2 Distributing the Code 3](#_Toc119321169)

[4 Example 2 – Text File in Subfolder 4](#_Toc119321170)

[5 Example 3 – Code in a Package 5](#_Toc119321171)

[6 Distributing Code – JAR Files 6](#_Toc119321172)

[6.1 Creating a Jar File with Eclipse 6](#_Toc119321173)

[Appendix 1 Error Running a Java File 8](#_Toc119321174)

[Appendix 2 Reference Material 8](#_Toc119321175)

# Using Eclipse to Write Java Programs to Read/Write Text Files

The purpose of this document is to illustrate how to use Eclipse (or other IDE) to develop Java programs to read/write text files and how to distribute them so that they can be run from the command line on another computer.

Consider the Eclipse Package Explorer shown below, on the left. Using Eclipse to write a Java program (*ReadFile.java*) to read a text file (*numbers.txt*), one creates a project (*project*) which creates a *src* folder. Inside *src*, we generally specify a package (*ver1*). The default location where Java looks for a text file, is the project folder.

|  |
| --- |
| Eclipse Package Explorer: |
| projectsrcver1ReadFile.javanumbers.txt |

In such a situation, we would use a *File* object as shown below:

File inFile = **new** File("numbers.txt");

It is convenient when developing and testing code to place the file that is to be read, in the folder where the code is. file to place the text file at the same location where the code is, in the *package* folder as shown in the table below, on the left and the *File* object required is shown on the right.

|  |  |
| --- | --- |
| Eclipse Package Explorer: | File Object |
| projectsrcver1ReadFile.javanumbers.txt | File inFile = **new** File("src\\ver1\\numbers.txt"); |

However, hard-coding the path to the text file as shown above is not a portable solution. Code using such a setup, can run in Eclipse, on another computer, provided the package structure is duplicated. However, running it from the command line is problematic. One problem is the hard-coded path to the file in the *File* statement above, the *src\ver1* folder structure would have to be created. Second, the fact that the code is in a package, makes it a bit more awkward to run from the command line. [Gradle](https://docs.gradle.org/current/userguide/what_is_gradle.html) and [Maven](https://maven.apache.org/) are two build automation tools that can automate these tasks. However, this is not a good use for either simply because of the hard-coded path – we don’t need to specify it so rigidly.

In the next section, we illustrate how to change the default location that Eclipse uses to look for text files. This is useful so that we can develop and test the code in Eclipse without the hard-coded path. The section after that, we show a good approach to developing a simple program that is portable.

# Change the Default Location that Eclipse uses to Look for a Text File

You can change the default location of where Java looks for text files by following the process below. However, this still doesn’t address the portability issue, which we consider in the next section.

1. Right-click the file in the Package Explorer and choose: Run As, Run Configurations
2. Choose the *Arguments* tab and scroll down to the *Working Directory*
3. Choose: *Other* and then specify the location. For the example below, the project is: *IOExamples2* and the package is *ver1*. Thus, the location is: ${workspace\_loc:IOExamples2/src/ver1}



1. Choose: Apply and Close (or Run). After doing this, the file can be placed in the package folder, alongside the code, and the *File* can be created as shown below:

File inFile = **new** File("numbers.txt");

As we see in the next section, this solves some of the portability problem.

# Example 1 – Portable Approach for Reading a Text File

If you want to use Eclipse to develop and test code to read a text file, that is also relatively portable, the simplest approach is:

* Do not specify a package for the classes you write. (the default package, as it is called)
* Change the default location of text files in Eclipse to the *src* folder.

Next, we illustrate this with an example.

## Eclipse Development

(The code for this example is in *ReadFile1.java*) For this example: (a) the project is: *IOExamples*, (b) the code is: *ReadFile1.java*, (c)and the file being read is *numbers.txt*. To start development in Eclipse:

1. Create an Eclipse project, *IOExamples*, and the copy (or write) the two files above into the *src* folder (no package).
2. [Change the default location](#_Change_the_Default_1) to: ${workspace\_loc:IOExamples/src}

Then, the layout in Eclipse, and the required *File* object are as shown below.

|  |  |
| --- | --- |
| Eclipse Package Explorer: | File Object |
| IOExamplessrcReadFile1.javanumbers.txt | File inFile = **new** File("numbers.txt"); |

## Distributing the Code

Next, we’ll simulate distributing this code and running it on another machine. In theory, all we need to distribute is the class file, *ReadFile1.class* and the text file, *numbers.txt*. Do the following:

1. Copy these two files (*ReadFile1.class* is located in: *IOExamples\bin*) to a folder somewhere else on your drive, in this case, I use *temp.*

Thus, the folder structure in this new location is shown below:

|  |
| --- |
| Folder Structure |
| tempReadFile1.classnumbers.txt |

1. Open a command prompt.

One way to open the command prompt in Windows is: (a) press Windows key + R, (b) type: “cmd”, (c) press OK).

1. Navigate to the *temp* folder.

On a PC, you can use [MS-DOS](https://www.c3scripts.com/tutorials/msdos/commands.html), the *cd* command.

1. From the command prompt, issue this command (red) to run the program:

C:\...\temp>java ReadFile1

They program may not run as explained in this [Appendix](#ErrorRunningJavaFile). If this is the case, then you need to compile the source code first, then run as above. To do this, copy *ReadFile1.java* (located in *IOExamples\src*) to the *temp* folder and then compile and run as shown below.

|  |  |
| --- | --- |
| Folder Structure | Compile Program |
| tempReadFile1.javaReadFile1.classnumbers.txt | C:\...\temp>javac ReadFile1.java |
| Run Program |
| C:\...\temp>java ReadFile1 |

# Example 2 – Text File in Subfolder

(The code for this example is in *ReadFile2.java*) This is similar to the last example, except the file to read is now in a subfolder, *data.* For this example: (a) the project is: *IOExamples*, (b) the code is: *ReadFile2.java*, (c)and the file being read is *numbers.txt*. To start development in Eclipse:

1. Create a project, *IOExamples*, and the copy (or write) the two files above into the *src* folder (no package).
2. [Change the default location](#_Change_the_Default_1) to: ${workspace\_loc:IOExamples/src}

Then, the layout in Eclipse, and the required *File* object are as shown below.

|  |  |
| --- | --- |
| Eclipse Package Explorer: | File Object |
| IOExamplessrcReadFile2.javadatanumbers.txt | File inFile = **new** File("data\_ex2\\numbers.txt"); |

Next, we’ll simulate distributing this code and running it on another machine. Do the following:

1. Copy, *ReadFile2.java* to a folder somewhere else on your drive, in this case, I use *temp.* Then create the folder *data­* in *temp* and copy *numbers.txt* there.

Thus, the folder structure in this new location is shown below:

|  |
| --- |
| Folder Structure |
| tempReadFile2.classReadFile2.javadatanumbers.txt |

1. Open a command prompt and compile and run:

C:\...\temp>javac ReadFile2.java

C:\...\temp>java ReadFile2

# Example 3 – Code in a Package

(The code for this example is in the *ver1* package, *ReadFile3.java*) For this example, the code is in package and the text file is at the same level. For this example: (a) the project is: *IOExamples*, (b) the code is: *ReadFile3.java*, (c)and the file being read is *numbers.txt*. To start development in Eclipse:

1. Create a project, *IOExamples*, a *ver1* package, and then copy (or write) the two files above into the *ver1* folder.
2. [Change the default location](#_Change_the_Default_1) to: ${workspace\_loc:IOExamples/src/ver1}

Then, the layout in Eclipse, and the required *File* object are as shown below.

|  |  |
| --- | --- |
| Eclipse Package Explorer: | File Object |
| IOExamplessrcver1ReadFile3.javanumbers.txt | File inFile = **new** File("numbers.txt"); |

Next, we’ll simulate distributing this code and running it on another machine. Do the following:

1. Go to some folder on your drive, in this case, I use *temp.*
2. Create a folder, *ver1* in the *temp* folder.
3. Copy *ReadFile3.java* and *numbers.txt* to the *ver1* folder. The folder structure in this new location is shown below:

|  |
| --- |
| Folder Structure |
| tempver1ReadFile3.javanumbers.txt |

1. Open a command prompt and compile and run:

C:\...\temp>javac ver1\ReadFile3.java

C:\...\temp>java ver1.ReadFile3

|  |
| --- |
| Also works:* example3\ver1>javac ReadFile3.java

Do not work:* example3\ver1>javac ver1.ReadFile3.java (file not found)
* example3>javac ver1.ReadFile3.java (file not found)
* example3\ver1>java ver1.ReadFile3 (couldn’t find file or load main class)
* example3\ver1>java ReadFile3 (couldn’t find file or load main class)
 |

# Distributing Code – JAR Files

[Incomplete – to be continued] Another approach to distributing code is to create a JAR file. A JAR file enables you to package together all the files you need for an application into a single archive file. A Jar file can be executable and, in that case, when it is double-clicked, it will run the program provided the JRE version on the host computer is more the same or more recent than the version on computer where the Jar was created. An executable Jar file can be created in Eclipse or from the command line – we show both approaches here. Finally, a Jar file can also be a library, just a collection of classes; however, we won’t consider that here, though the only difference is that an executable Jar’s manifest specifies the class with the *main* method to run the program.

## Creating a Jar File with Eclipse

For this example, the code is in a package and the text file is “in” the package also.

For this example: (a) the project is: *IOExamples*, (b) the code is: *ReadFileLauncher.java* and *ReadFile4.java*, (c) *ReadFIleLauncher* starts the program, calling *ReadFile4* to read a text file, (d)and the file being read is *numbers.txt*. (The source code for this example is in an accompanying zip file.) To start development in Eclipse:

1. Create a project, *IOExamples*, a *ver2* package, and then copy (or write) the two Java files and the text file into the *ver2* package.
2. [Change the default location](#_Change_the_Default_1) to: ${workspace\_loc:IOExamples/src/ver2}

Then, the layout in Eclipse, and the required *File* object are as shown below.

|  |  |
| --- | --- |
| Eclipse Package Explorer: | File Object |
| IOExamplessrcver2ReadFileLauncher.javaReadFile4.javanumbers.txt | File inFile = **new** File("numbers.txt"); |

Next, we’ll simulate distributing this code and running it on another machine. Do the following:

1. When you create a Jar file, you can use a *manifest* file which has information about the class that contains *main* (to start the program), and other required libraries (we won’t have any of these). We will have Eclipse make one for use. So, create an empty file on your drive, in the workspace folder. I named mine: *rflmanifest.mf.* I’m not sure that this is necessary, but it will work this way.
2. Select the *ver2* folder in the Package Explorer.
3. Choose: File, Export, Java, JAR File, Next
4. Do the following (reference the figure below):
5. Drill down and select the two Java files. I unselected the text files as we would want the program to read any text file. Thus, we’ll put the text file right beside the Jar file in the File Explorer.
6. Choose: Browse to select the “Export Destination”, *i.e.* the location where you want to save the Jar file. Specify a file name. I used: *rfl.jar*
7. Choose: Next, Next



1. Choose the options below:
2. Select: “Save the manifest…”. This is optional, but we will do it just to see the manifest file that is created.
3. Browse to the folder (I used the project folder, *IOExamples*) and specify a file name. I used: *rflmanifest.mf*
4. Browse to select the “Main class”.
5. Choose: Finish.
6. Navigate to the location where the manifest file is and display its contents:

Manifest-Version: 1.0

Main-Class: ver2.ReadFileLauncher

1. Navigate to the location of the Jar file, I’ll use *temp*. Make sure the text file is there also.

|  |
| --- |
| Folder Structure |
| temprfl.jarnumbers.txt |

1. Open a command prompt and run:

C:\...\temp>java -jar rfl.jar

LEFT OFF HERE

1. To compile jar file.

<https://docs.oracle.com/javase/tutorial/deployment/jar/index.html>

<https://www.baeldung.com/java-create-jar>

<https://www.geeksforgeeks.org/jar-files-java/>

<https://www3.ntu.edu.sg/home/ehchua/programming/java/J9d_Jar.html>

 Appendix

1. Error Running a Java File

If you run a class file generated by Eclipse from the command line, you may get this error:

|  |
| --- |
| Error: A JNI error has occurred, please check your installation and try again Exception in thread "main" java.lang.UnsupportedClassVersionError: ReadFile1 has been compiled by a more recent version of the Java Runtime (class file version 60.0), this version of the Java Runtime only recognizes class file versions up to 52.0 |

This indicates the JRE Eclipse is using is more recent than the one you are trying to run it under on your (or some other) machine. This can be fixed in at least three ways: (a) Compile the source file from the command line, (b) in Eclipse, in the compiler options, set it to compile to a lower version (I think 1.8 would work), (c) change your PATH (Windows) so that the more recent JRE is referenced earlier in the statement. This question explains how to do *b* and *c*:

<https://stackoverflow.com/questions/58125830/has-been-compiled-by-a-more-recent-version-of-the-java-runtime-class-file-versi>

These answers discuss some of reasons a Java class or Jar file might not run on another computer.

<https://stackoverflow.com/questions/31915959/executable-jar-not-working-on-other-computers>

1. Reference Material

This is background information that I have found useful.

1. [Difference Between JDK and JRE in Java](https://www.geeksforgeeks.org/difference-between-jdk-and-jre-in-java/) – Useful, the last row summarizes the relationship.
2. [How to Set Java Path in Windows and Linux](https://www.geeksforgeeks.org/how-to-set-java-path-in-windows-and-linux/) – Discussion of PATH environment variable and how to set it so that the operating system knows where to look for the java executables (java.exe, javac.exe, *etc.*). I followed the directions to just see the PATH (not change it). Mine was set to:

C:\Program Files\Eclipse Foundation\jdk-8.0.302.8-hotspot\bin

Meanwhile, I have another installation at:

C:\Program Files\Java\jdk1.8.0\_231\bin

1. [How to run Eclipse under different version of JDK or JRE](https://www.codejava.net/ides/eclipse/how-to-run-eclipse-under-different-version-of-jdk-or-jre) – This is the version of Java that is running Eclipse (which is written in Java). On my machine the vm is set to a location I’m not familiar with.

\*\*\* Date: Wednesday, September 28, 2022 at 9:31:10 AM Eastern Daylight Time

-vm

C:/Users/dgibson/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86\_64\_16.0.2.v20210721-1149/jre/bin\server\jvm.dll

eclipse.vm=C:/Users/dgibson/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86\_64\_16.0.2.v20210721-1149/jre/bin\server\jvm.dll

eclipse.vmargs=-Dosgi.requiredJavaVersion=11

java.vm.compressedOopsMode=32-bit

java.vm.info=mixed mode

java.vm.name=OpenJDK 64-Bit Server VM

java.vm.specification.name=Java Virtual Machine Specification

java.vm.specification.vendor=Oracle Corporation

java.vm.specification.version=16

java.vm.vendor=Oracle Corporation

java.vm.version=16.0.2+7-67

1. [JPMS-Modules](https://www.baeldung.com/java-9-modularity) – Java Platform Module System (JPMS) is a level of abstraction above packages (introduced in Java 9).
2. [jLink](https://www.baeldung.com/jlink) – A tool that generates a custom JRE image. Drawback is that it can’t create cross-platform executables. This may require [javapackager](https://docs.oracle.com/javase/9/tools/javapackager.htm#JSWOR719).