


Reading and writing files is an essential part of software development, especially when working on large projects. These files can be used for a variety of tasks, from documenting software health to initializing various parts of a program.

Reading and writing files is also the cornerstone of any good digital notebook. In this lab, you'll practice using code to handle files by developing an idea tracker – a space to capture, organize, and revisit your software project ideas whenever inspiration strikes.

Goal

Create a program that allows you to track all your software project ideas by reading ideas from a file, modifying them as needed, and writing new ideas back into the file.

 Note: When you encounter this icon, it's time to get into your IDE and start coding!

In your lab environment, open IntelliJ by double-clicking on the icon.

Before diving into coding, open the *sample-project-idea.txt* file from the *src* folder and observe its contents. (Remember this is a sample idea for a project; it shows what appears when your *IdeaTracker* writes an idea to a file.)

```
Idea 1: Create an application to track personal expenses.
```

```
This application will let a user register, log in, and create a persona for themselves.
```

```
The application will ask the user to enter a transaction they have done, and store it in a file or a database.
```


```
The application must also provide a report of some sort to the user to visualize their expenses so far.
```

Now, let's prepare your program to read those project ideas from a file. Since the content from files needs to be displayed in the idea tracker, you'll need a method to retrieve the file's contents.

Step 1: Initialize the file object

Open the *IdeaReader* class from the *src* folder and locate the *readIdea()* method.


Currently, this method only includes a `String` variable named *fileContents* that will hold the file's contents once it's read. Your task is to enhance this method so it can read the file's data and prepare it for display in your idea tracker.

 It's time to get coding!

- TODO 1: Create an object of the *File* class using the *filePath* parameter of the *readIdea()* method.
- TODO 2: Pass the *file* object to the *FileInputStream* object in the *try* block.

Step 2: Read and prepare file contents

All the requirements for reading a file are now complete – well done! You have a *file* object that specifies the file you want to read and a *FileInputStream* object that provides the methods to read its contents. Now, read the entire file's contents at once.

 It's time to get coding!

- TODO 3: Call the *readAllBytes()* method on the *inputStream* object and store them in an array of bytes. Name the array *fileContentsAsBytes*.

```
byte[] fileContentsAsBytes = inputStream.readAllBytes()
```


At this stage, the file can be read and its contents stored in an array of bytes, which is great. However, since the *readIdea()* method needs to return the content as a `String`, the next step is to convert the array. This conversion makes it easier to display and manipulate the data within your program.

- TODO 4: Pass the array inside a constructor of the *String* class. Then, store it back in the *fileContents* variable so it can be returned from the *readIdea()* method.

```
fileContents = new String(fileContentsAsBytes);
```

Step 3: Handle file not found error

The *FileNotFoundException* is triggered when the file path specified by the user cannot be found on the disk. When this happens, it's important to communicate the issue to the user clearly, so do that next.

 It's time to get coding!


- TODO 5: Print an error message using the *System.err.println()* method on the console to let the user know that they made a mistake and need to try again. The message could read: *"File Not Found! Please check the file path and try again!"*.

Take a moment to ensure that everything is functioning correctly so far. Run your code and test the *readIdea()* method to confirm that the file is being read properly

and that its contents are accurately converted to a string. This extra step will help you catch any issues early, which is the best time to catch them.

It's time to connect the pieces by integrating the reading method into the main program, allowing you to pull up and display project ideas directly from a file whenever needed.

Start by opening the *IdeaTracker* class from the *src* folder and locating *case 1* inside the switch statement. This case handles calling the *IdeaReader* class and calling the appropriate method to read an idea file for you!

 It's time to get coding!

- TODO 6: Run the *main()* method and select *1* as input for the menu to read a file. Write the following as input for the path of the file to be read from the program:

```
src/sample-project-idea.txt
```

Expected output

Compare your output with the expected output.

```
Welcome to the Idea Tracker!
```

```
Select an option to continue:
```

```
1. Read a file
```

```
2. Write a file
```

```
3. Exit
```

```
Enter your choice (1, 2, or 3): 1
```

```
Please enter the file of the path to read: src/sample-project-idea.txt
```

```
-----
```

```
Idea 1: Create an application to track personal expenses.
```

```
This application will let a user register, log in, and create a persona for themselves.
```

```
The application will ask the user to enter a transaction they have done, and store it in a file or a database.
```

The application must also provide a report of some sort to the user to visualize their expenses so far.

Welcome to the Idea Tracker!

Select an option to continue:

1. Read a file
2. Write a file
3. Exit

Enter your choice (1, 2, or 3):

Troubleshooting


If your output doesn't match the expected output, you can try the following steps:

1. Check the path of the file you have entered in the console. The “src/” is needed at the beginning to make sure your program can find the specified file.
2. Make sure the *File* object created in the *IdeaReader* class is initialized properly with the correct *filePath* parameter.
3. Make sure the *FileInputStream* object is initialized properly with the *file* object.

Important

At the moment, the file you want to read is in the same project folder as your code, so you don't need to type out the full file path – just use *src/sample-project-idea.txt*. But if your file is located somewhere else on your computer, you'll need to include the full path to where the file is stored.

To ensure your ideas are safely stored, complete the *writeIdea()* method in the *IdeaWriter* class. This method will take *filePath* and *fileContents* as arguments and create a new text file that contains your project idea with the specified description.

 It's time to get coding!

- **TODO 7:** Create an object of the *File* class. Name the variable *file* and pass in the *filePath* parameter of the *readIdea()* method into the constructor of the *File* class.

```
File file = new File(filePath);
```

You already have the contents of the file to be written in the *fileContents* variable as a String parameter of the *writeIdea()* method. Next, convert the String parameter into an array of bytes.

- TODO 8: Call the *getBytes()* method on the *fileContents* variable and then store it in a byte array called *fileContentsAsBytes*.

```
byte[] fileContentsAsBytes = fileContents.getBytes();
```

- TODO 9: Inside the *try* block's parenthesis, where the *FileOutputStream* object (*outputStream*) is being created, remove the *"* and pass in the file object you created earlier.


```
try (FileOutputStream outputStream = new FileOutputStream(file)) {}
```

- TODO 10: Call the *write()* method on the *outputStream* object and pass in the *fileContentsAsBytes* array.

```
outputStream.write(fileContentsAsBytes);
```

Excellent! Your *writeIdea()* method is now ready to save all your project ideas to a file. With this in place, you'll never miss out on capturing, finding, or editing a great idea again – right?

The moment of truth! Run your program to make sure everything works as planned. The program should now be able to read existing files and save new project ideas without a hitch.

 It's time to get coding!

- TODO 11: Run your code using the IDE. Your program should be able to read in the file already given in the project or write out a new file as specified in the console menu. (You may need to exit the program from the menu to see the created file in your project.)

In this lab, you've written Java code to read and write files using the *FileInputStream* and *FileOutputStream* classes. These byte stream methods are fundamental tools for handling files on your computer's hard disk. By completing this lab, you've gained the ability to manage basic file operations – an essential skill for any software developer.

As you progress in this course, the file handling techniques you've learned will be invaluable for tackling more complex projects. These skills not only prepare you for future course challenges but also position you to handle real-world coding tasks, such as logging data and processing files, with confidence.