Business success depends on easily accessible reports and information. How do you know this? Because it’s your boss’ motto and he repeats it several times a day!

You already appreciate the importance of efficient management of monthly reports because you know that these essential documents allow the management team to perform operational analysis and make effective decisions.

In this activity, you will create code to make copies of the reports. The copies may be created within the same directory or in a different one. Your company can then use these copies either for backup purposes or reroute them for further processing.

Goal

In this activity, you will successfully implement the *copyFile* method in the *FileManager* class within your Java project. This gives you the opportunity to create effective code that will perform essential file management tasks within a Java application.

🖥️ 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.

Are you excited to get going? Well, your code will build on an existing project, so begin by opening the Java project named *FileManagementSystem* in IntelliJ IDEA. This project includes one Java file under the *‘src’* folder (*FileManager.java*) and two folders: *source_folder* (containing one text file with text) and an empty *target_folder*.

The *FileManager* class is designed to handle the task of copying files from one location to another within a Java application.

Open the *FileManager.java* file from the *src* folder and observe its contents. It encapsulates the logic for file copying, ensuring that files are transferred correctly and efficiently. It includes the main method to test the *copyFile* method. The method *copyFile* defines two parameters: *sourcePath* and *targetPath*.

You will need to complete four tasks to ensure that your copyfile method can work as you want it.

🖥️ It’s time to get coding!

- TODO 1: Convert the targetPath string into a *Path* object.

You already have the *sourcePath* converted to a *Path* object (source). Now, you need to do the same for *targetPath*. Remember that you can use the *Paths.get()* method to create a *Path* object from a String representing a file path.

- TODO 2: Copy the source file to the target location, replacing any existing file.

Use the *Files.copy()* method to copy the file from the source location to the target location. The *Files.copy()* method takes the source *Path*, the target *Path*, and optional copy options as arguments (such as *StandardCopyOption.REPLACE_EXISTING*).

- TODO 3: Print a message indicating that the file has been successfully copied.

Of course, on a practical level, you would need to know if a copy has been successful so you decide to ask for a message to be printed indicating if that is the case. Create code that confirms that the file is copied successfully by printing a message to the console indicating that the operation was completed without errors.

- TODO 4: If an *IOException* occurs, print an error message with the exception's message.

1. The *Files.copy()* method may fail and throw an *IOException* due to issues like missing files or permission problems. If this happens, printing the exception's details can provide valuable information.
2. Use a *try-catch* block to handle the *IOException*.
3. Inside the *catch* block, print an error message to the console. Use *e.getMessage()* to include the specific error message from the exception, which can be helpful for debugging.

Here is an example of the partially completed *FileManager* class with the TODO statements.

```
import java.nio.file.Files;


import java.nio.file.Path;


import java.nio.file.Paths;


import java.nio.file.StandardCopyOption;


import java.io.IOException;




public class FileManager {




    private static void copyFile(String sourcePath, String targetPath) {
        Path source = Paths.get(sourcePath);

// TODO 1: Convert the targetPath string into a Path object



        try {

// TODO 2: Copy the source file to the target location, replacing any existing file

// TODO 3: Print a message indicating that the file has been successfully copied

        } catch (IOException e) {
// TODO 4: If an IOException occurs, print an error message with the exception's
message



            e.printStackTrace();
        }
    }


}
```

Well done! You’ve added all the necessary code elements to implement your method and copy files, flag, and handle any errors. Now, it’s time to put your method to the test and verify that a file can be copied to a new location. Your carefully crafted code should generate an output message saying that the file has been successfully copied.

1. Of course, you sensibly decide not to test it on any company report or document. Instead, begin by copying a test file called *textFile1.txt*.
2. In the IDE, run the following:

```
public static void main(String[] args) {



      String sourceFile = "./source_folder/textFile1.txt";

      String targetFile = "./target_folder/textFile1.txt";



      copyFile(sourceFile, targetFile);

  }
```

How did it go? Did you get this message saying that the test file has been successfully copied?

```
File has been successfully copied from .source_folder/textFile1.txt to
./target_folder/textFile1.txt
```

If you did, well done! The *textFile1.txt* file is now in *target_folder.*

If you get an *IOException*, first check the file's name, path, and extension when creating a *Path* object for a file.