

# Introduction to File I/O

*by Sophia*



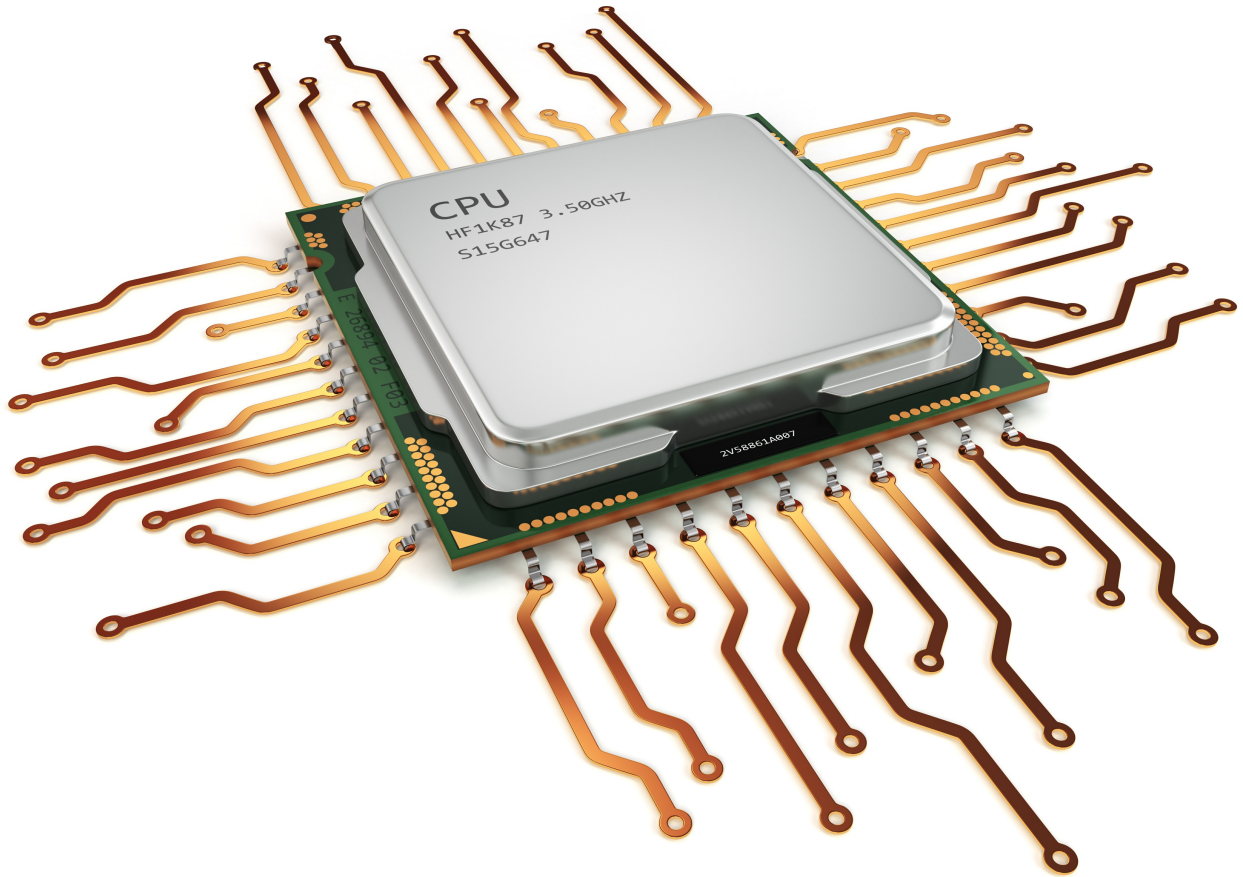
## WHAT'S COVERED

In this lesson, you will learn about some of the basic ways to work with files. These are reading, writing, and closing files. Specifically, this lesson covers:

### Table of Contents

- [1. Working With Files](#)
- [2. Reading Files](#)
- [3. Writing to Files](#)
- [4. Closing Files](#)

## 1. Working With Files



So far, you have learned how to write programs and communicate your intentions to the Central Processing Unit using conditional statements, functions, and iterations. The **Central Processing Unit** is the brain of any computer. It is what runs the software that we write. It's also called the "CPU" or the "processor." You have learned how to create and use data structures that are stored in the main memory of a computer. The **main memory** is used to store information that the CPU needs in a hurry. The main memory is nearly as fast as the CPU. The CPU and main memory are where our software works and runs. It is where all of the "thinking" happens.



Once the power is turned off, anything stored in either the CPU or main memory is erased. So, up to now, our programs have just been transient fun exercises to learn Java. Now you will start to work with secondary memory. **Secondary memory** is also used to store information, but it is much slower than main memory. The advantage of secondary memory is that it can store information even when there is no power to the computer.



Examples of secondary memory devices are a USB flash drive or hard drive.

By storing information on a USB flash drive, the information can be removed from the system and transported to another system.

We will primarily focus on reading and writing text files such as those we create in a text editor. To read or write a file on a drive, like your hard drive, you must first open the file. Opening the file communicates with your operating system, which knows where the data for each file is stored. When you open a file, you are asking the operating system to first find the file by name and make sure the file exists, and then make its contents available (open) for reading or writing.

#### IN CONTEXT

In the example below, we open the file *poem.txt*, which should be stored in the same folder as the Java file that will access it. In Replit, we can create a file, name it *poem.txt*, and add in the following text:

Roses Are Red,  
Violets are blue,  
Sugar is sweet,

And so are you.



TRY IT

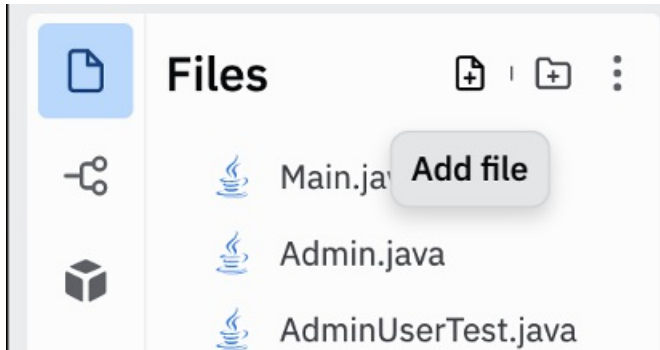
**Directions:** Try following the steps below to create a new file and add some code.



STEP BY STEP

1. Click on the 'Add file' icon at the top of the Files panel.

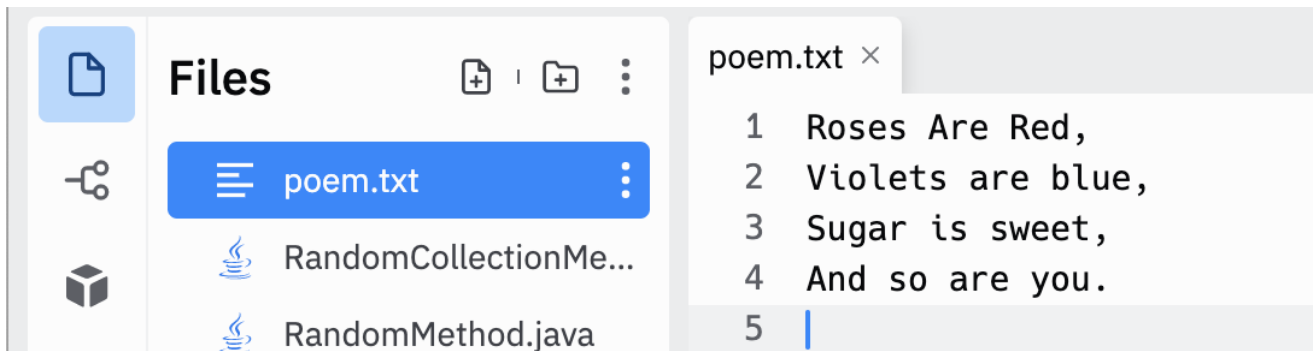
→ EXAMPLE



2. Next, we name the file; in this case, we will call it `poem.txt`. Remember, if you do not give a filename an extension, the default extension of `.java` will be added. We do not want to create a `.java` file, so we needed to add the `.txt` to make it a text file.

3. Then, we can paste in the code.

→ EXAMPLE



REFLECT

Notice that the icon for our *poem.txt* text file looks different than the Java code files (ending with the `.java` file extension).

Now that our *poem.txt* file has information inside of it, let's try to read the file.



TRY IT

**Directions:** First, we need to move back to the `FileExample.java` file to add the following code:

```
import java.io.File;
```

```

public class FileExample {
    public static void main(String[] args) {
        File poemFile = new File("poem.txt");
        System.out.println("File Information: ");
        System.out.println("File Name: " + poemFile.getName());
        System.out.println("File Path: " + poemFile.getPath());
        System.out.println("Full Path: " + poemFile.getAbsolutePath());
        System.out.println("File Size: " + poemFile.length());
        System.out.println("Can Read: " + poemFile.canRead());
        System.out.println("Can Write: " + poemFile.canWrite());
        System.out.println("Can Execute: " + poemFile.canExecute());
    }
}

```



REFLECT

It is worth noting that the Java File object is an abstraction of a file on disk. The file that it refers to may or may not exist. Depending on how the File object is used later in the code, the file may be created, opened, or read.



TRY IT

**Directions:** Next, run the program using the command `java FileExample.java`.

The output screen should look like this:

Console

Shell

```

> java FileExample.java
File Information:
File Name: poem.txt
File Path: poem.txt
Full Path: /home/runner/IntrotoJava/poem.txt
File Size: 68
Can Read: true
Can Write: true
Can Execute: false
> 

```



REFLECT

Not seeing the poem? That is because at this point we only accessed information about the file and we haven't read any data from the file. The output we see is just information about the file pointed to by the File object that we created.



### Central Processing Unit

The Central Processing Unit is the heart of any computer. It is what runs the software that we write; it is also called the “CPU” or the “processor.”

### Main Memory

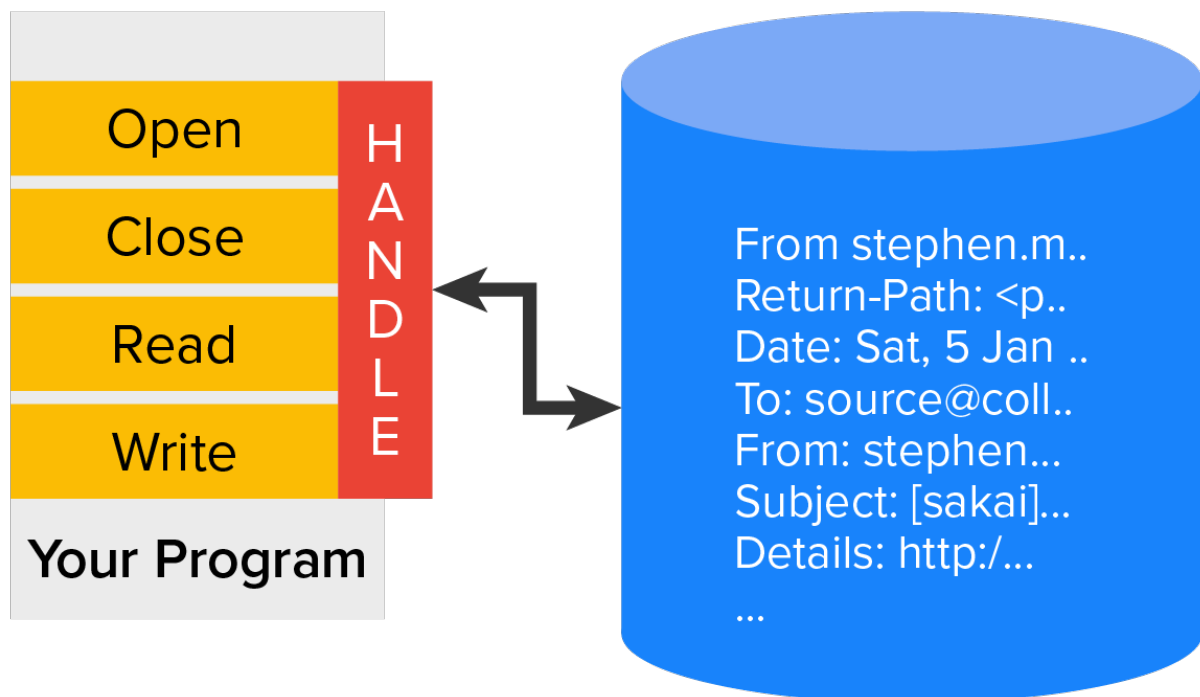
The main memory is used to store information that the CPU needs in a hurry. Main memory is nearly as fast as the CPU. The CPU and memory are where our software works and runs.

### Secondary Memory

Secondary memory is also used to store information, but it is much slower than main memory. The advantage of the secondary memory is that it can store information even when there is no power to the computer.

---

## 2. Reading Files



If opening a file is successful, the operating system returns a file handle. The **file handle** is not the actual data contained in the file, but instead it is a “handle” that can be used to read the data.



### BIG IDEA

Think of the file handle as being an address to a house. With the address to a house, you know where the house is, but the address itself is just pointing to the house location in the same way that a file handle is pointing to a file.

A text file can be thought of as a sequence of lines, much like a Java String can be thought of as a sequence of characters. The *poem.txt* file simply contains four lines of text for the poem.

While the file handle does not itself contain the data in the file, it is quite easy to write code using the `Files` utility class that reads the contents of the whole file and stores the lines in a `List` collection of `String` objects. The size of the `List` is the number of lines in the file.

When working with the methods in the `Files` utility class, the `File` object's `toPath()` method will be called. A `Path` object has many similarities to a `File` object, and both classes include methods for converting objects back and forth.



**Directions:** Revise the code in the `FileExample.java` file so that it looks like the following code:

```
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.util.List;

public class FileExample {
    public static void main(String[] args) {
        // File object for text file
        File poemFile = new File("poem.txt");
        try {
            // Files.readAllLines() reads entire file & puts lines in the
            // List<String>
            List<String> lines = Files.readAllLines(poemFile.toPath());
            System.out.println("Line count: " + lines.size());
        }
        // If Files.readAllLines() can't find or read file, it throws an
        // IOException
        catch(IOException ex) {
            System.out.println("Error accessing file: " + ex.getMessage());
        }
    }
}
```

This simple program should produce the following output:

```
Console Shell
> java FileExample.java
Line count: 4
>
```



**Directions:** Try using the `Files.readAllLines()` method to count the lines of our *poem.txt* file.

## ➞ EXAMPLE

Use the `Files.readAllLines()` method to count the lines of our *poem.txt* file.



## CONCEPT TO KNOW

Note that `Files.readAllLines()` does what the name indicates and reads all of the lines of the file into memory. This is fine for cases like this one where the file is small. In cases where the file is larger, it might be more appropriate to use a `Scanner` to read from the file using a while loop.

The `Scanner`'s `nextLine()` method reads each line until the new line character (`\n`) is encountered:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class FileScannerExample {
    public static void main(String[] args) {
        // Scanner can read from File pointing to file
        File poemFile = new File("poem.txt");
        int lineCount = 0;

        try {
            // Throws FileNotFoundException if file not found
            Scanner fileScanner = new Scanner(poemFile);
            // while loop runs as long another line is available
            while(fileScanner.hasNextLine()) {
                // Read line as String
                String line = fileScanner.nextLine();
                lineCount++;
            }
            // Must close Scanner when reading from a file.
            fileScanner.close();
            System.out.println("Line count: " + lineCount);
        }
        catch(FileNotFoundException ex) {
            System.out.println("Error accessing file: " + ex.getMessage());
        }
    }
}
```



## TRY IT

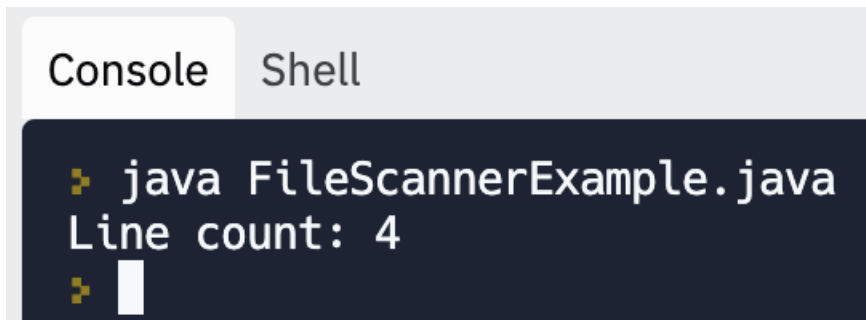
**Directions:** Type in the code above in a file named `FileScannerExample.java` and run it using the command:

```
java FileScannerExample.java
```

This example should produce the following output, which is the same as the version using



Files.readAllLines():



```
Console Shell
> java FileScannerExample.java
Line count: 4
>
```

#### REFLECT

We have seen a couple of the approaches that Java provides for reading files. If you continue on in the language past this course, you will undoubtedly learn more. Sometimes this variety reflects the evolution of the language, and sometimes it reflects demands by programmers. Think about how a choice of methods can make life both better and worse for programmers using the language.

#### CONCEPT TO KNOW

The newline character is a whitespace character similar to the space and tab character; it is not shown as a '\ ' (space) or a '\t' (tab). Even though the newline character is stored as a '\n' (newline), you would not see that character as a '\n' but rather you would see the text moving to the next line.

Because the `while` loop reads the data one line at a time, it can efficiently read and count the lines in very large files without running out of main memory to store the data. The above program can count the lines in any size file using very little memory since each line is read, counted, and then discarded.

If you know the file is relatively small compared to the size of your main memory, you can read the whole file into one string using the `Files.readAllLines()` method. What constitutes a large or small file is relative and depends on the amount of memory available and the number of files that may be open at one time.

Rather than reading the lines in the file to count them, it would be useful to actually display the contents. Since the code above that uses `File.readAllLines()` stores the lines of the file in a list, we can use a `for` loop to cycle through the list and print out the lines.

#### TRY IT

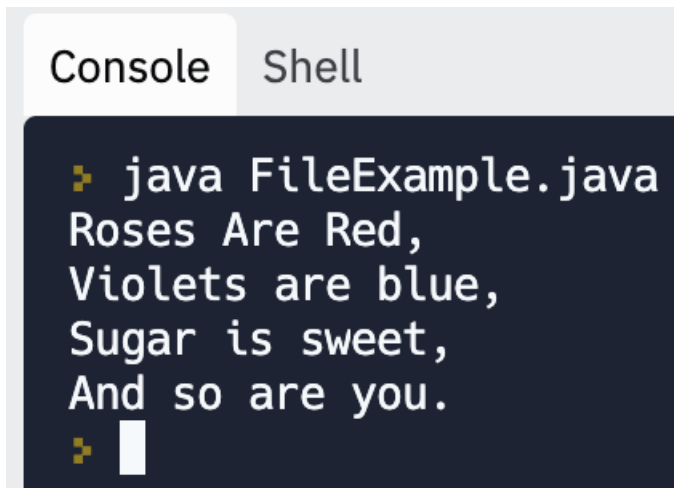
**Directions:** Modify the `FileExample.java` file so the code reads as follows:

```
import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.util.List;

public class FileExample {
    public static void main(String[] args) {
        // File object for text file
        File poemFile = new File("poem.txt");
        try {
```

```
// Files.readAllLines() reads entire file & puts lines in the
// List<String>
List<String> lines = Files.readAllLines(poemFile.toPath());
// Loop through the list and print out the lines
for(String line : lines) {
    System.out.println(line);
}
}
// If Files.readAllLines() can't find or read file, it throws an
// IOException
catch(IOException ex) {
    System.out.println("Error accessing file: " + ex.getMessage());
}
}
}
```

This would be the expected output:



```
Console Shell
➤ java FileExample.java
Roses Are Red,
Violets are blue,
Sugar is sweet,
And so are you.
➤
```

#### REFLECT

In this example, the entire contents of the file *poem.txt* are read directly into the List. Each line is stored as a String object in the List. The range-based for loop iterates over the List and prints out each line.

#### BRAINSTORM

Now try to use the `.read()` method and see if you can output the contents of our *poem.txt* file.

#### TERM TO KNOW

##### **File Handle**

The file handle is not the actual data contained in the file, but instead it is a “handle” that we can use to read the data.

## 3. Writing to Files

The Files class includes a method called `write()` that writes an iterable collection (like an ArrayList) to an output file. Depending on the options passed when the method is called, the data in the collection may be

written to a new file, overwrite the current contents of a file, or be appended to the contents of an existing file.

Here is a short program that creates a file and writes three lines of text to it. The code creates an `ArrayList<String>` containing the lines to be written to the file. Since there is the possibility of exceptions when working with a file, the relevant statements are in a try block, and there is a catch block for an `IOException`.

#### ➞ EXAMPLE

```
import java.io.*;
import java.nio.file.*;
import java.util.ArrayList;

public class WriteTextToFile {
    public static void main(String[] args) {
        // Create ArrayList of Strings & add lines of text
        ArrayList<String> lines = new ArrayList<>();
        lines.add("Line 1");
        lines.add("Line 2");
        lines.add("Line 3");
        // File object pointing to output.txt file (which may not exist yet)
        File outputFile = new File("output.txt");

        try {
            // StandardOpenOption.CREATE creates a new file. It will create the file
            // if it doesn't exist or overwrite it if it does.
            Files.write(outputFile.toPath(), lines, StandardOpenOption.CREATE);
        }
        catch(IOException ex) {
            System.out.println("Error writing to file: " + ex.getMessage());
        }
    }
}
```



#### CONCEPT TO KNOW

If the file already exists, opening it in `StandardOpenOption.CREATE` mode (`StandardOpenOption` is an example of a Java **enumeration**) clears out the old data and starts fresh, so be careful! If the file doesn't exist already, it will be created.



#### TERM TO KNOW

##### Enumeration

`StandardOpenOption` is an example of a Java enumeration. An enumeration is a named collection of constant values.

---

## 4. Closing Files

In the sample program showing how to use a `Scanner` to read a text file's contents, the `Scanner` object needs

to be closed (which then closes the underlying file). Note the following line:

```
fileScanner.close();
```

It carries out this task in the code:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class FileScannerExample {
    public static void main(String[] args) {
        // Scanner can read from File pointing to file
        File poemFile = new File("poem.txt");
        int lineCount = 0;

        try {
            // Throws FileNotFoundException if file not found
            Scanner fileScanner = new Scanner(poemFile);
            // while loop runs as long another line is available
            while(fileScanner.hasNextLine()) {
                // Read line as String
                String line = fileScanner.nextLine();
                lineCount++;
            }
            // Must close Scanner when reading from a file.
            fileScanner.close();
            System.out.println("Line count: " + lineCount);
        }
        catch(FileNotFoundException ex) {
            System.out.println("Error accessing file: " + ex.getMessage());
        }
    }
}
```



#### BIG IDEA

When using the methods for reading and writing text to and from a file, it is not necessary to close the file explicitly, since the methods provided by `java.nio.Files` take care of this step.

Recall how this version of the code for reading the file's contents using the `Files.readAllLines()` does not have a separate statement to close the file:

#### ➞ EXAMPLE

```
import java.io.File;
import java.io.IOException;
```

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

public class FileExample {
    public static void main(String[] args) {
        // File object for text file
        File poemFile = new File("poem.txt");
        try {
            // Files.readAllLines() reads entire file & puts lines in the
            // List<String>
            List<String> lines = Files.readAllLines(poemFile.toPath());
            System.out.println("Line count: " + lines.size());
        }
        // If Files.readAllLines() can't find or read file, it throws an
        // IOException
        catch(IOException ex) {
            System.out.println("Error accessing file: " + ex.getMessage());
        }
    }
}
```



REFLECT

If you continue with Java, you will encounter some of the older methods using the input and output streams mentioned above. When using these pre-java.nio approaches, it will be important to handle the closing of the files. This is another advantage of using the newer methods provided by the Files class.



## SUMMARY

In this lesson, you learned about different ways that you can **work with files**, including **reading a text file's** contents using the Files class and a Scanner. You also learned how to use the Files utility class to **write text to a file**. Finally, you learned about exception handling using try and catch when working with and **closing files**.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source [cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf](https://cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf)

It has also been adapted from “Python for Everybody” By Dr. Charles R. Severance. Source [py4e.com/html3/](https://py4e.com/html3/)



## TERMS TO KNOW

### Central Processing Unit

The Central Processing Unit is the heart of any computer. It is what runs the software that we write; it is also called the “CPU” or the “processor.”

### Enumeration

StandardOpenOption is an example of a Java enumeration. An enumeration is a named collection of

constant values.

**File Handle**

The file handle is not the actual data contained in the file, but instead it is a “handle” that we can use to read the data.

**Main Memory**

The main memory is used to store information that the CPU needs in a hurry. Main memory is nearly as fast as the CPU. The CPU and memory are where our software works and runs.

**Secondary Memory**

Secondary memory is also used to store information, but it is much slower than the main memory. The advantage of the secondary memory is that it can store information even when there is no power to the computer.