# Free Java Course



COMPUTER COURSES

Home
Beginners Computing
Word 2007 to 2013
Word 2000 to 2003
Excel 2007 to 2013
Excel to 2003
Excel VBA Programming for Beginners
Web Design
Javascript
Visual Basic .NET
Beginners PHP
C# .NET
Java for Beginners
10 Programming Exercises

## How to Read a Text File in Java

Manipulating text files is a skill that will serve you well in your programming career. In this section, you'll learn how to open and how to write to a text file. But by text file, we just mean a file with text in it - simple as that! You can create a text file in programmes like Notepad on a Windows computer, TextEdit on a Mac, Gedit in a Linux/Gnome environment.

The first thing we'll do is to open up a text file and read its contents.

### Read a Text File

Start a new project for this. Call the package **textfiles** and the class **FileData**. Add an import statement just below the package line and before the class name:

**import java.io.IOException;**

Your coding window will then look like this:

```
package textfiles;
import java.io.IOException;

public class FileData {

    public static void main(String[] args)  {

    }

}
```

To deal with anything going wrong with our file handling, add the following to the main method (the text in bold):

**public static void main(String[ ] args) throws IOException {**

**}**

We're telling Java that the main method will throw up an **IOException** error, and that it has to be dealt with. Later, we'll add a **try … catch** block to display an appropriate error message for the user, should something go wrong.

To open the text file, let's create a new class. So click **File > New File** from the NetBeans menu at the top. Create a new Java Class file and give it the name **ReadFile**. When your new class is created, add the following three import statements:

**import java.io.IOException;**
**import java.io.FileReader;**
**import java.io.BufferedReader;**

Your new class should then look like this one:

```
package textfiles;

import java.io.IOException;
import java.io.FileReader;
import java.io.BufferedReader;

public class ReadFile {

}
```

(The import lines are underlined because we haven't done anything with them yet. This is a NetBeans feature.)

We'll create a new object from this class to read a file. Add the following constructor to your code, along with the private String field called **path**:

```
public class ReadFile {

    private String path;

    public ReadFile(String file_path) {
        path = file_path;
    }
}
```

All we're doing here is passing in the name of a file, and then handing the file name over to the path field.

What we now need to do is create a method that returns all the lines of code from the text file. The lines will be held in an array. Add the following method declaration that will open up the file:

```
public class ReadFile {

    private String path;

    public ReadFile(String file_path) {
        path = file_path;
    }

    public String[] OpenFile() throws IOException {


    }
}
```

Don't worry about the red underline: it will go away once we've added some code. NetBeans has just added it because we have no return statement.

Notice that the method is set up to return a String array, though:

**public String[ ]**

The array will contain all the lines from the text file.

Notice, too, that we've added "throws IOException" to the end of the method header. Every method that deals with reading text files needs one of these. Java will throw any errors up the line, and they will be caught in our main method.

To read characters from a text file, the **FileReader** is used. This reads bytes from a text file, and each byte is a single character. You can read whole lines of text, rather than single characters. To do this, you can hand your FileReader over to something called a **BufferedReader**. The BufferedReader has a handy method called ReadLine. As its name suggests, it is used to read whole lines, rather than single characters. What the BufferedReader does, though, is to store characters in memory (the buffer) so that they can be manipulated more easily.

Add the following lines that set up a FileReader and a BufferedReader:

```
public String[] OpenFile() throws IOException {

    FileReader fr = new FileReader(path);
    BufferedReader textReader = new BufferedReader(fr);

}
```

We're creating two new objects here: one is a FileReader object which we've called **fr**; the other is a BufferedReader object with the name **textReader**.

The FileReader needs the name of file to open. For us, the file path and name is held in the field variable called path. So we can use this.

The BufferedReader is handed the FileReader object between its round brackets. All the characters from the file are then held in memory waiting to be manipulated. They are held under the variable name **textReader**.

Before we can read the lines of text, we need to set up an array. Each position in the array can then hold one complete line of text. So add the following two lines to your code:

```
int numberOfLines = 3;
String[ ] textData = new String[numberOfLines];
```

For now, we'll set the number of lines in the text file to just 3. Obviously, text files can hold any number of lines, and we usually don't know how many. So we'll change this soon. We'll write a separate method that gets the number of lines in a text file.

The second line of new code, though, sets up a String array. The number of positions in the array (its size) is set to the number of lines. We've put this between the square brackets.

To put all the lines of text from the file into each position in the array, we need a loop. The loop will get each line of text and place each line into the array. Add the following to your code:

```
int i;

for (i=0; i < numberOfLines; i++) {
        textData[ i ] = textReader.readLine();

}
```

Your coding window should now look like this:

```
public String[] OpenFile() throws IOException {

    FileReader fr = new FileReader(path);
    BufferedReader textReader = new BufferedReader(fr);

    int numberOfLines = 3;
    String[] textData = new String[numberOfLines];

    int i;

    for (i=0; i < numberOfLines; i++) {
        textData[i] = textReader.readLine();
    }

}
```

The for loop goes from 0 to just less than the number of lines. (Array positions, remember, start at 0. The 3 lines will be stored at positions 0, 1, and 2.)

The line that accesses the lines of text and stores them in the array is this one:

```
textData[i] = textReader.readLine( );
```

After the equals sign we have this:

```
textReader.readLine( );
```

The textReader object we set up is holding all the characters from the text file in memory (the

buffer). We can use the readLine method to read a complete line from the buffer. After the line is read, we store the line in an array position:

**textData[i]**

The variable called **i** will increment each time round the loop, thus going through the entire array storing lines of text.

Only two more lines of code to add to the method, now. So add these lines to your code:

```
textReader.close( );
return textData;
```

The close method flushes the temporary memory buffer called textReader. The return line returns the whole array. Notice that no square brackets are needed for the array name.

When you've added the code, all those ugly underlines should disappear. Your method should then look like this:

```
public String[] OpenFile() throws IOException {

    FileReader fr = new FileReader(path);
    BufferedReader textReader = new BufferedReader(fr);

    int numberOfLines = 3;
    String[] textData = new String[numberOfLines];

    int i;

    for (i=0; i < numberOfLines; i++) {
        textData[i] = textReader.readLine();
    }

    textReader.close();
    return textData;
}
```

There's still the problem of the number of lines, however. We've hard-coded this to 3. What we need is to go through any text file and count how many lines it has. So add the following method to your ReadFile class:

```
int readLines() throws IOException {

   FileReader file_to_read = new FileReader(path);
   BufferedReader bf = new BufferedReader(file_to_read);

   String aLine;
   int numberOfLines = 0;

   while (( aLine = bf.readLine()) != null) {
           numberOfLines++;
   }
   bf.close();

   return numberOfLines;

}
```

The new method is called **readLines**, and is set to return an integer value. This is the number of lines a text file has. Notice this method also has an **IOException** part to the method header.

The code for the method sets up another FileReader, and another BufferedReader. To loop round the lines of text, we have this:

```
while ( ( aLine = bf.readLine( ) ) != null ) {
        numberOfLines++;
}
```

The while loop looks a bit messy. But it just says "read each line of text and stop when a null value is reached." (If there's no more lines in a text file, Java returns a value of null.) Inside the curly

brackets, we increment a counter called numberOfLines.

The final two lines of code flush the memory buffer called bf, and returns the number of lines.

To call this new method into action, change this line in your OpenFile method:

**int numberOfLines = 3;**

Change it to this:

**int numberOfLines = readLines( );**

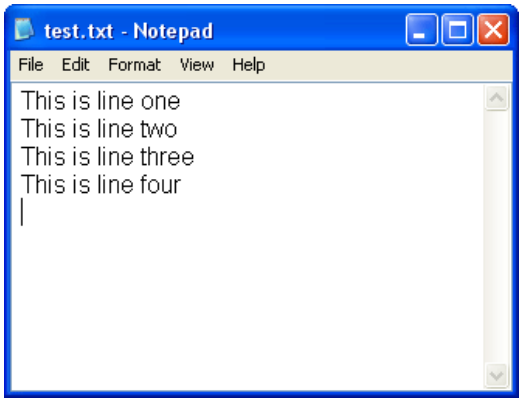
So instead of hard-coding the number of lines, we can call our new method and get the number of lines in any text file.

OK, time to put the new class to work and see if it opens a text file.

Go back to your **FileData** class, the one with the main method in it. Set up a string variable to hold the name of a text file:

```
package textfiles;
import java.io.IOException;

public class FileData {

    public static void main(String[] args) throws IOException {

        String file_name = "C:/test.txt";

    }
}
```

At this stage, you need to create a text file somewhere on your computer. We created this simple one in Notepad on a Windows machine:




The name of the text file is "test.txt". Create a similar text file on your own computer. Note where you saved it to because you need the file path as well:

**String file_name = "C:/test.txt";**

So our **test.txt** file is saved on the C drive. If we had created a folder called MyFiles to hold the file then the path would be "C:/MyFiles/test.txt". Change you file path, if need be.

The next thing to do is to create a new object from our **ReadFile** class. We can then call the method that opens the file. But we can do this in a **try … catch** block. Add the following code, just below your String variable line:

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

    String file_name = "C:/test.txt";

    try {
        ReadFile file = new ReadFile(file_name);
        String[] aryLines = file.OpenFile();

    }
    catch (IOException e) {
          System.out.println( e.getMessage() );
    }
}
```

Don't forget all the curly brackets for the try … catch block. You need one pair for the try part and another pair for the catch part. For the try part, we have this:

```
ReadFile file = new ReadFile( file_name );
String[ ] aryLines = file.OpenFile( );
```

The first line sets up a new ReadFile object called **file**. In between the round brackets of ReadFile, we added the **file_name** variable. This is enough to hand the constructor the file path it needs.

The second line of code sets up a String array called **aryLines**. After the equals sign, we've called the **OpenFile** method of our ReadFile class. If it successfully opens up the text file, then the array of text lines will be handed over to the new array aryLines.

If something goes wrong, however, an error is thrown up the line, and ends up in the catch part of the **try … catch** block:

```
catch ( IOException e ) {
        System.out.println( e.getMessage() );
}
```

After the word "catch" we have a pair of round brackets. Inside the round brackets, we have this:

```
IOException e
```

What this does is to set up a variable called e which is of type **IOException**. The IOException object has methods of its own that you can use. One of these methods is getMessage. The will give the user some information on what went wrong.

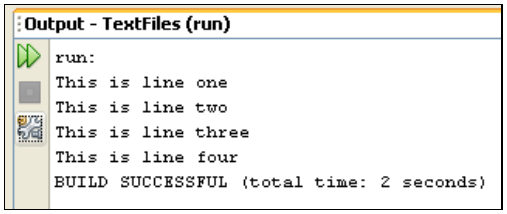
Before we see an example of an error message, let's loop through all the lines of the text file, printing out each one. Add the following loop code to the **try** part of the **try … catch** block:

```
int i;
for ( i=0; i < aryLines.length; i++ ) {
        System.out.println( aryLines[ i ] ) ;
}
```

Your coding window should now look like this:

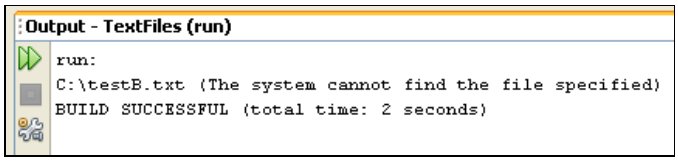
```
public static void main(String[] args) throws IOException {

    String file_name = "C:/test.txt";

    try {
        ReadFile file = new ReadFile(file_name);
        String[] aryLines = file.OpenFile();

        int i;
        for (i=0; i < aryLines.length; i++) {
            System.out.println(aryLines[i]);
        }
    }
    catch (IOException e) {
          System.out.println( e.getMessage() );
    }
}
```

When the programme is run, the Output window will print the following:



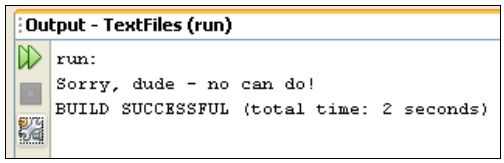

As you can see, each line from our text file has been printed out.

To test the error checking part of the code, change the name of your text file to one you know has not been created. Then run your code again. In the Output window below, you can see that our text file was changed to **testB**, and that it couldn't be found:




If you prefer, you can add your own error message to the catch block:

```
catch (IOException e) {
      System.out.println( "Sorry, dude - no can do!" );
}
```




It's probably better to leave it to Java, though!

In the next part, you'll learn how to write to a text file using Java code.

<-- Logic Errors | Write to a Text File -->

Back to the Home Page