Programming with Java

Java File Handling

The File class from the java.io package, allows us to work with files.

To use the File class, create an object of the class, and specify the filename or directory name:

Example

```
import java.io.File;  // Import the File class
File myObj = new File("filename.txt"); // Specify the filename
```

The File class has many useful methods for creating and getting information about files. For example:

Method	Type	Description
canRead()	Boolean	Tests whether the file is readable or not
canWrite()	Boolean	Tests whether the file is writable or not
createNewFile()	Boolean	Creates an empty file
delete()	Boolean	Deletes a file
exists()	Boolean	Tests whether the file exists
getName()	String	Returns the name of the file
getAbsolutePath()	String	Returns the absolute pathname of the file
length()	Long	Returns the size of the file in bytes
list()	String[]	Returns an array of the files in the directory
mkdir()	Boolean	Creates a directory

Create a File

To create a file in Java, you can use the createNewFile() method. This method returns a boolean value: true if the file was successfully created, and false if the file already exists.

Note that the method must be enclosed in a ***try...catch*** block. This is necessary because it throws an IOException if an error occurs (if the file cannot be created for some reason).

Example

```
import java.io.File;  // Import the File class
import java.io.IOException;//Import the IOException class to handle
errors
public class CreateFile {
  public static void main(String[] args) {
    try {
      File myObj = new File("filename.txt");
      if (myObj.createNewFile()) {
        System.out.println("File created: " + myObj.getName());
      } else {
        System.out.println("File already exists.");
      }
    } catch (IOException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
  }
}
```

To create a file in a specific directory (requires permission), specify the path of the file and use double backslashes to escape the "\" character (for Windows). On Mac and Linux, you can just write the path, like: /Users/name/filename.txt

Example

```
File myObj = new File("C:\\Users\\MyName\\filename.txt");
```

Get File Information

To get more information about a file, use any of the File methods mentioned previously.

Example

```
import java.io.File;  // Import the File class
public class Exercise1 {
  public static void main(String[] args) {
    File myObj = new File("filename.txt");
    if (myObj.exists()) {
      System.out.println("File name: " + myObj.getName());
      System.out.println("Absolute path: " + myObj.getAbsolutePath());
      System.out.println("Writeable: " + myObj.canWrite());
```

```
      System.out.println("Readable " + myObj.canRead());
      System.out.println("File size in bytes " + myObj.length());
    } else {
      System.out.println("The file does not exist.");
    }
  }
}
```

Delete a File

We use the delete() method to delete a file. Following is a demonstration of how to delete a file in Java.

Example:

```
//Import the File class
import java.io.File;

public class Exercise1 {
 public static void main(String[] args)
 {
     File Obj = new File("filename.txt");
     if (Obj.delete()) {
         System.out.println("The deleted file is: " + Obj.getName());
     }
     else {
        System.out.println("Failed in deleting the file.");
     }
 }
}
```

Write to a File

We use the ***FileWriter*** class together with its ***write()*** method to write some text to the file we created in the example above. Note that when you are done writing to the file, you should close it with the ***close()*** method:

Example

```
import java.io.FileWriter;   // Import the FileWriter class
import java.io.IOException;  // Import the IOException class to handle
errors
```

```
public class WriteToFile {
  public static void main(String[] args) {
    try {
      FileWriter myWriter = new FileWriter("filename.txt");
      myWriter.write("Files in Java might be tricky, but it is fun
enough!");
      myWriter.close();
      System.out.println("Successfully wrote to the file.");
    } catch (IOException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
  }
}
```

Note: The writer will rewrite the content of the file every time it is called. To make it append the new content to the old we use the append parameter in the constructor and set it to true.

```
FileWriter myWriter = new FileWriter("filename.txt", true);
```

Read a File

In this lecture we learn two ways to read from a file

1. ***Using Java FileReader Class***

   Java ***FileReader*** class is used to read data from the file. It returns data in byte format. It is character-oriented class that is used for file handling in java.

   We use the ***read()*** method from this class, and it also requires closing the file at the end.

   Example

```
import java.io.FileNotFoundException;
import java.io.FileReader;
public class FileReaderExample {
    public static void main(String args[])throws Exception{
         try {
            FileReader fr=new FileReader("filename.txt");
             int i;
              while((i=fr.read())!=-1)
              System.out.print((char)i);
              fr.close();
         }
```

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

2. ***Using the Scanner class***

   In the following example, we use the Scanner class to read the contents of the text file we created:

   Example

```
import java.io.File;  // Import the File class
import java.io.FileNotFoundException;  // Import this class to
handle errors
import java.util.Scanner; // Import the Scanner class to read
text files

public class ScannerReadExample {
  public static void main(String[] args) {
    try {
      File myObj = new File("filename.txt");
      Scanner myReader = new Scanner(myObj);
      while (myReader.hasNextLine()) {
        String data = myReader.nextLine();
        System.out.println(data);
      }
      myReader.close();
    } catch (FileNotFoundException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
  }
}
```

Note: There are many available classes in the Java API that can be used to read and write files in Java: FileReader, BufferedReader, Files, Scanner, FileInputStream, FileWriter, BufferedWriter, FileOutputStream, etc. Which one to use depends on the Java version you're working with and whether you need to read bytes or characters, and the size of the file/lines etc. Thus, your choice will depend on the project you are developing and your preference.