File Handling in Java

File handling in Java is a crucial concept for reading from and writing to files. The Java `java.io` package provides classes like `File`, `FileReader`, `FileWriter`, `BufferedReader`, and `BufferedWriter` that make it easy to work with files.

In this tutorial, we'll explore the basics of file handling by covering the following operations:

1. **Creating a File**
2. **Writing to a File**
3. **Reading from a File**
4. **Appending to a File**
5. **Deleting a File**

---

1. Creating a File

To create a file in Java, you use the `File` class and its `createNewFile()` method. This method returns `true` if the file was created successfully, and `false` if the file already exists.

Example:

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

public class CreateFileExample {
    public static void main(String[] args) {
        try {
            File file = new File("example.txt");
            if (file.createNewFile()) {
                System.out.println("File created: " + file.ge
tName());
            } else {
                System.out.println("File already exists.");
            }
        } catch (IOException e) {
            System.out.println("An error occurred while creat
ing the file.");
            e.printStackTrace();
        }
    }
}
```

Explanation:

- The `File` object is created with the specified file name (`example.txt`).
- The `createNewFile()` method attempts to create the file.

---

2. Writing to a File

To write to a file, you can use the `FileWriter` class. The `write()` method is used to write content to the file.

Example:

```
import java.io.FileWriter;
import java.io.IOException;

public class WriteFileExample {
    public static void main(String[] args) {
        try {
            FileWriter writer = new FileWriter("example.tx
t");
            writer.write("Hello, this is a sample text.");
            writer.close();
            System.out.println("Successfully wrote to the fil
e.");
        } catch (IOException e) {
            System.out.println("An error occurred while writi
ng to the file.");
            e.printStackTrace();
        }
    }
}
```

Explanation:

- The `FileWriter` object is used to write text to the file.
- The `write()` method adds the text, and the `close()` method closes the file to save the changes.

---

3. Reading from a File

To read from a file, you can use the `FileReader` class along with `BufferedReader` for efficient reading.

Example:

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

public class ReadFileExample {
    public static void main(String[] args) {
        try {
            FileReader fileReader = new FileReader("example.t
xt");
            BufferedReader bufferedReader = new BufferedReade
r(fileReader);
            String line;
            while ((line = bufferedReader.readLine()) != nul
l) {
                System.out.println(line);
            }
            bufferedReader.close();
        } catch (IOException e) {
            System.out.println("An error occurred while readi
ng the file.");
```

```
            e.printStackTrace();
        }
    }
}
```

Explanation:

- `BufferedReader` is used for efficient reading line by line.
- The `readLine()` method reads each line until the end of the file (`null`).

---

4. Appending to a File

To append content to an existing file, you can use the `FileWriter` class with the `append` mode enabled by setting the second parameter to `true`.

Example:

```
import java.io.FileWriter;
import java.io.IOException;

public class AppendFileExample {
    public static void main(String[] args) {
        try {
            FileWriter writer = new FileWriter("example.txt",
true);
            writer.write("\\nThis is an appended line.");
            writer.close();
            System.out.println("Successfully appended to the
file.");
        } catch (IOException e) {
            System.out.println("An error occurred while appen
ding to the file.");
            e.printStackTrace();
        }
    }
}
```

Explanation:

- The second parameter in the `FileWriter` constructor is `true`, enabling append mode.

---

5. Deleting a File

To delete a file, you use the `delete()` method of the `File` class.

Example:

```
import java.io.File;

public class DeleteFileExample {
    public static void main(String[] args) {
        File file = new File("example.txt");
        if (file.delete()) {
            System.out.println("Deleted the file: " + file.ge
tName());
        } else {
            System.out.println("Failed to delete the file.");
        }
    }
}
```

Explanation:

- The `delete()` method attempts to delete the file and returns `true` if successful.

Conclusion

This tutorial demonstrates how to perform basic file operations in Java, including creating, writing, reading, appending, and deleting files. These operations form the foundation for file handling, which is essential for many real-world applications.

<u>Reading from a File in Java</u>