https://github.com/kkh3363/bsu_java2025

Java Programming
Chapter 10

Java File

https://github.com/kkh3363/bsu_java2025

Java File

- File handling is an important part of any application.
- Java has several methods for creating, reading, updating, and deleting files.
- doc(java.io)

- 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:

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

File myObj = new File("filename.txt"); // Specify the filename
```

https://github.com/kkh3363/bsu_java2025

Java File

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

Java File

- Create a File
  - you can create a new file with the createNewFile() method from the File class.
  - This method returns:
    - true - if the file was created successfully
    - false - if the file already exists
  - Note that the method is 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):

Java File

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

public class CreateFile {
  public static void main(String[] args) {
    try {
      File myObj = new File("filename.txt"); // Create File object
      if (myObj.createNewFile()) {          // Try to create the file
        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(); // Print error details
    }
  }
}
```

Java File

- Create a File in a Specific Folder
  - 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

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

Java File

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

public class CreateFileDir {
  public static void main(String[] args) {
    try {
      File myObj = new File("C:\\Users\\MyName\\filename.txt");
      if (myObj.createNewFile()) {
        System.out.println("File created: " + myObj.getName());
        System.out.println("Absolute path: " + myObj.getAbsolutePath());
      } else {
        System.out.println("File already exists.");
      }
    } catch (IOException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
  }
}
```

https://github.com/kkh3363/bsu_java2025

Java File

- Java Write To Files
  - If you are just starting with Java, the easiest way to write text to a file is by using the FileWriter class.

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

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();  // must close manually
      System.out.println("Successfully wrote to the file.");
    } catch (IOException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
  }
}
```

Java File

- Since Java 7, you can use try-with-resources. This makes sure the writer is closed automatically, even if an error occurs:

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

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

- Append to a File
  - Normally, FileWriter will overwrite a file if it already exists.
  - If you want to add new content at the end of the file (without deleting what's already there), you can use the two-argument constructor and pass true as the second parameter.
  - This puts the writer into append mode:

https://github.com/kkh3363/bsu_java2025

Java File

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

public class AppendToFile {
  public static void main(String[] args) {
    // true = append mode
    try (FileWriter myWriter = new FileWriter("filename.txt", true)) {
      myWriter.write("\nAppended text!");
      System.out.println("Successfully appended to the file.");
    } catch (IOException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
  }
}
```

Java File

- Read a File
  - we use the Scanner class to read the contents of the text file we created in the previous chapter:

```java
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 ReadFile {
  public static void main(String[] args) {
    File myObj = new File("filename.txt");

    // try-with-resources: Scanner will be closed automatically
    try (Scanner myReader = new Scanner(myObj)) {
      while (myReader.hasNextLine()) {
        String data = myReader.nextLine();
        System.out.println(data);
      }
    } catch (FileNotFoundException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
  }
}
```

Java File

- Get File Information
  - To get more information about a file, use any of the File methods:

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

public class GetFileInfo {
  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.");
    }
  }
}
```

Java File

- Delete a File
  - To delete a file in Java, use the delete() method:

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

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

Java File

- Delete a Folder

```
import java.io.File;

public class DeleteFolder {
  public static void main(String[] args) {
    File myObj = new File("C:\\Users\\MyName\\Test");
    if (myObj.delete()) {
      System.out.println("Deleted the folder: " + myObj.getName());
    } else {
      System.out.println("Failed to delete the folder.");
    }
  }
}
```