I/O

Java.io package

- Java I/O (Input and Output) is used to process the input and produce the output.
- The java.io package in Java provides classes for input and output operations.
- It is a part of the Java Standard Edition (SE) API and is used for handling various input and output streams, readers, and writers.
- The classes in this package support reading and writing data to files, streams, and other I/O sources.
- Java uses the concept of a stream to make I/O operation fast.
- The java.io package contains all the classes required for input and output operations.
- We can perform file handling in Java by Java I/O API.

Java.io package

- File handling is an important part of any application.
- Java has several methods for creating, reading, updating, and deleting files.
- 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***

Java.io package

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

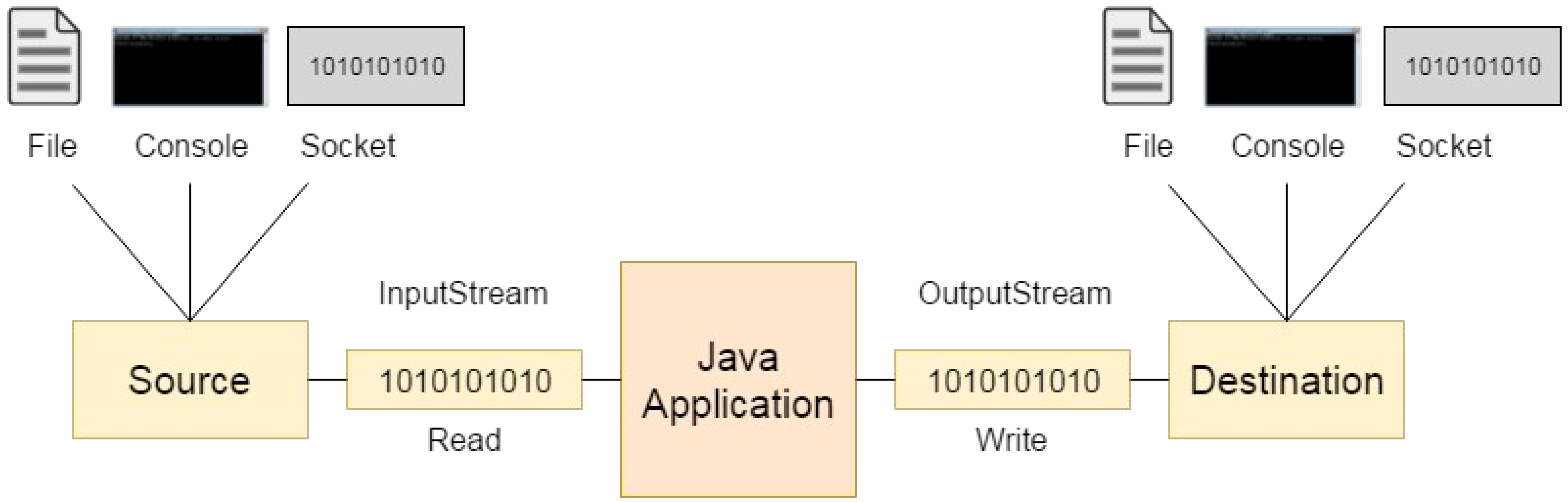
Method	Type	Description
`canRead()`	Boolean	Tests whether the file is readable or not
`canWrite()`	Boolean	Tests whether the file is writable or not
`createNewFile()`	Boolean	Creates an empty file
`delete()`	Boolean	Deletes a file
`exists()`	Boolean	Tests whether the file exists
`getName()`	String	Returns 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

Java.io package

- **Stream**
- A stream is a sequence of data. In Java, a stream is composed of bytes.
- It's called a stream because it is like a stream of water that continues to flow.
- In Java, 3 streams are created for us automatically. All these streams are attached with the console.
  - ***System.out:*** *standard output stream*
  - ***System.in:*** *standard input stream*
  - ***System.err:*** *standard error stream*

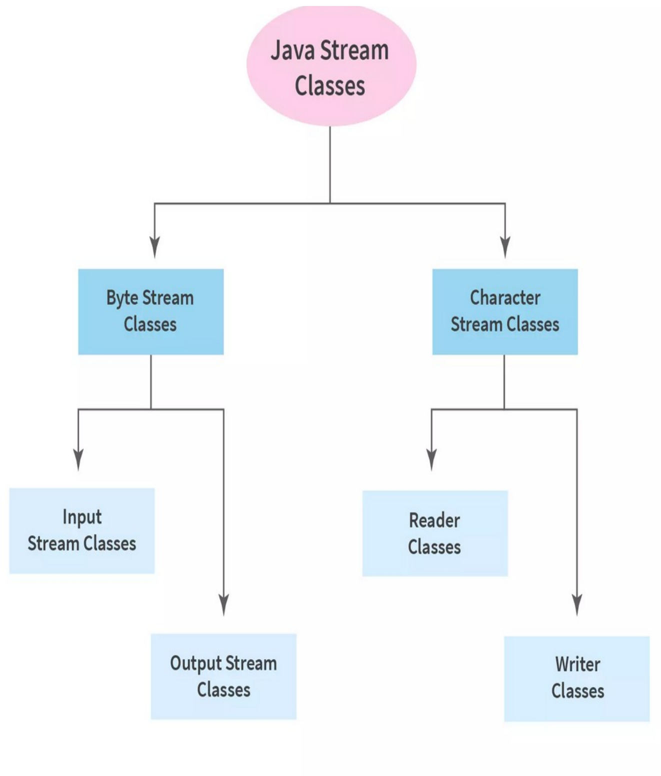
Java.io package

- **Stream**



Java.io package

- In Java, I/O streams are broadly classified into two categories:
- Byte Streams and Character Streams.
- Byte Streams are used for handling raw binary data, while Character Streams are used for handling character data.
- The distinction is important because it helps ensure proper handling of character encoding.



Classification of Java Stream Classes

Java.io package

- ***Byte Streams:***
- *Byte Streams are designed for handling input and output of raw binary data.*
- *They are suitable for all kinds of files, whether they contain text, images, audio, or other binary data.*
- *Some key classes for Byte Streams in the java.io package include:*
  - ***InputStream and OutputStream:*** *Abstract classes for reading and writing bytes, respectively.*
  - ***FileInputStream and FileOutputStream:*** *Read and write bytes to and from files.*
  - ***ByteArrayInputStream and ByteArrayOutputStream:*** *Read from or write to byte arrays.*
  - ***DataInputStream and DataOutputStream:*** *Read and write primitive data types in binary format.*

Java.io package

- ***Character Streams:***
- *Character Streams are designed specifically for handling character data.*
- *They are essential when working with text files to ensure proper character encoding.*
- *Some key classes for Character Streams include:*
  - ***Reader and Writer:** Abstract classes for reading and writing characters, respectively.*
  - ***FileReader and FileWriter:** Read and write characters to and from files.*
  - ***CharArrayReader and CharArrayWriter:** Read from or write to character arrays.*
  - ***BufferedReader and BufferedWriter:** Provide buffering for more efficient reading and writing of characters.*
  - ***PrintWriter and PrintStream:** Facilitate formatted writing of characters*

Java.io package

- *This example reads a file named "input.txt" using FileInputStream.*
- *The read() method is used to read one byte at a time.*

```
import java.io.FileInputStream;
import java.io.IOException;
public class FileInput {
    public static void main(String[] args) {
        try (FileInputStream inputStream = new FileInputStream("input.txt")) {
            int data;
            while ((data = inputStream.read()) != -1) {
                System.out.print((char) data);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Java.io package

- This example writes strings in a file named "output.txt" using FileOutputStream.
- The write() method is used to write byte array to the file.

```
import java.io.FileOutputStream;
import java.io.IOException;
public class Fileout {
    public static void main(String[] args) {
        String content = "Hello, this is a sample text to be written in a file.";
        try (FileOutputStream outputStream = new FileOutputStream("output.txt")) {
            // Convert the string to bytes and write to the file
            byte[ ] contentBytes = content.getBytes();
            outputStream.write(contentBytes);
            System.out.println("Text written to the file successfully.");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Java.io package

- ***Create a File***
- *To create a file in Java, we 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 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)*

```
import java.io.File;
import java.io.IOException;
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();
    }
  }
}
```

Java.io package

- ***Write To a File***
- *we use the FileWriter class together with its write() method to write some text to the file.*
- *Note that when we are done writing to the file, we should close it with the close() method:*

```
import java.io.FileWriter;
import java.io.IOException;
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();
    }
  }
}
```

Java.io package

- *The FileReader class in Java is used to read character data from a file.*
- *It is part of the java.io package and is designed specifically for reading characters.*

```
import java.io.FileReader;
import java.io.IOException;
public class FileReaderExample {
    public static void main(String[] args) {
        try (FileReader reader = new FileReader("example.txt")) {
            int charCode;
            // Read characters until the end of the file
            while ((charCode = reader.read()) != -1) {
                char character = (char) charCode;
                System.out.print(character);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Java.io package

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

```
import java.io.File;
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.");
    }
  }
}
```

Assignment

- Explain the difference between Byte Streams and Character Streams in Java I/O.
- What is the purpose of the java.io package in Java? Provide examples of scenarios where it is commonly used.
- Describe the role of InputStream and OutputStream classes in the context of Byte Streams. Provide an example use case for each.
- What is the significance of using FileReader and FileWriter classes when dealing with text files in Java? How do they differ from FileInputStream and FileOutputStream?
- Explain the concept of buffering in the context of I/O streams. Why is it beneficial to use BufferedInputStream and BufferedWriter classes?

Assignment

- Write a Java program using FileInputStream to read the contents of a binary file and display them on the console.
- Develop a Java application that uses FileOutputStream to create a new text file and write a series of lines to it.
- Create a program that reads data from an existing text file using FileReader and prints it to the console.
- Write a Java program that copies the content of one text file to another using FileReader and FileWriter classes.
- Develop a simple utility in Java that uses FileInputStream and FileOutputStream to copy the contents of one binary file to another.