FileHandling in JAVA

BY
M. BABY ANUSHA,
ASST.PROF IN CSE DEPT.,
RGUKT,NUZVID

Introduction :

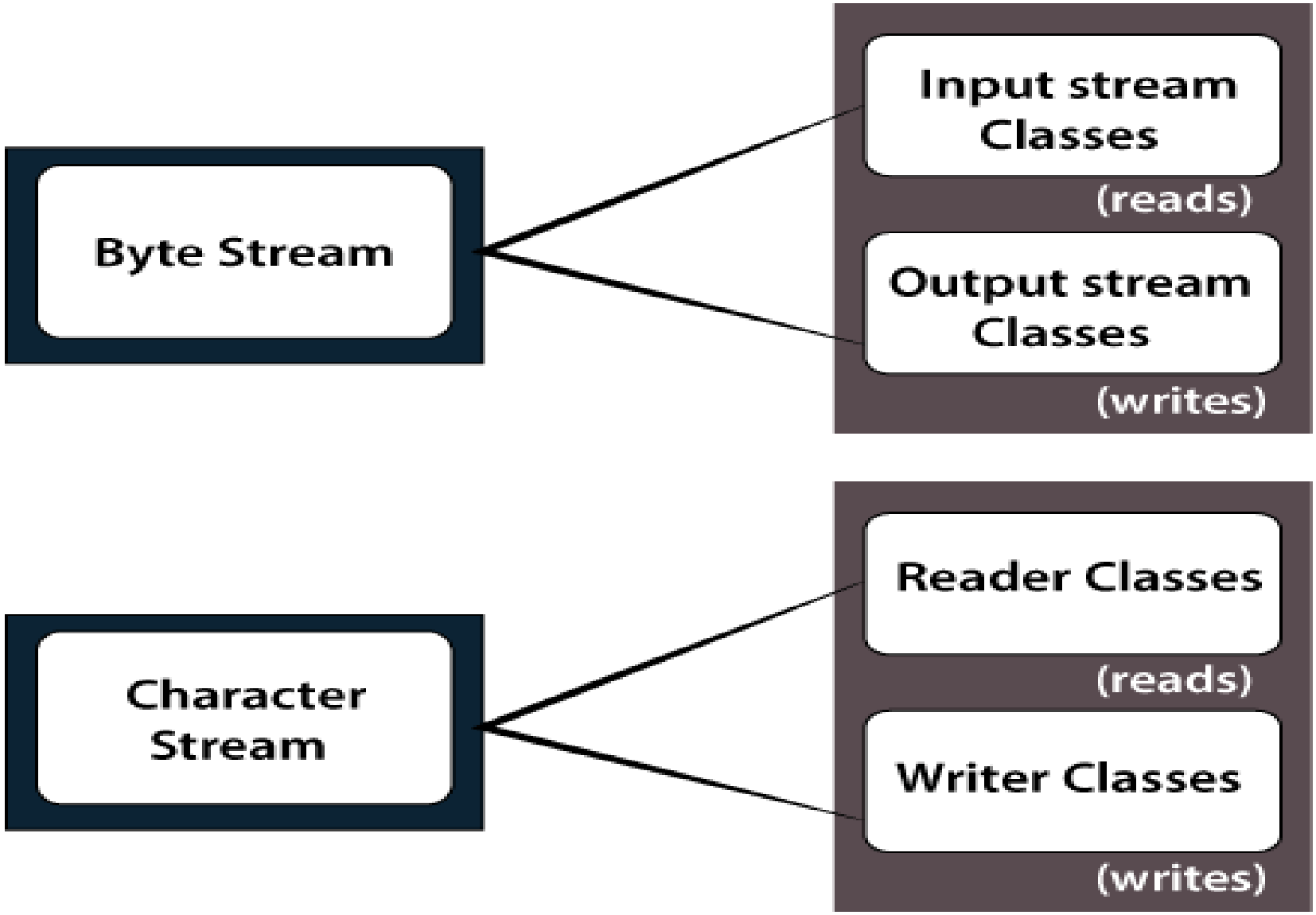
- In Java, a **File** is an abstract data type. A named location used to store related information is known as a **File**. There are several **File Operations** like **creating a new File, getting information about File, writing into a File, reading from a File** and **deleting a File**.
- Before understanding the File operations, it is required that we should have knowledge of **Stream** and **File methods.**

File Streams :

- A series of data is referred to as **a stream**. In Java , **Stream** is classified into two types, i.e., **Byte Stream** and **Character Stream**

Byte Stream :

- **Byte Stream** is mainly involved with byte data. A file handling process with a byte stream is a process in which an input is provided and executed with the byte data.



Brief classification of I/O streams

File Streams :

Character Stream :

- **Character Stream** is mainly involved with character data. A file handling process with a character stream is a process in which an input is provided and executed with the character data.

Java File Class Methods :

boolean canRead() :

- The **canRead()** method is used to check whether we can read the data of the file or not.

boolean createNewFile()

- The **createNewFile()** method is used to create a new empty file.

boolean canWrite()

- The **canWrite()** method is used to check whether we can write the data into the file or not.

Java File Class Methods :

boolean exists()

- The **exists()** method is used to check whether the specified file is present or not.

boolean delete()

- The **delete()** method is used to delete a file.

string getName()

- The **getName()** method is used to find the file name.

String getAbsolutePath()

- The **getAbsolutePath()** method is used to get the absolute pathname of the file.

Java File Class Methods :

long length()

- The **length()** method is used to get the size of the file in bytes.

String[] list()

- The **list()** method is used to get an array of the files available in the directory.

boolean mkdir()

- The **mkdir()** method is used for creating a new directory.

File Operations :

- We can perform the following operation on a file:
- Create a File
- Get File Information
- Write to a File
- Read from a File
- Delete a File

File Operations in Java

Create a File :

- **Create a File** operation is performed to create a new file. We use the **createNewFile()** method of file.
- The **createNewFile()** method returns true when it successfully creates a new file and returns false when the file already exists.

Create a File : 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(); }
} }
```

Create a File : Example

Output:

File created: filename.txt

Write to a File :

- The next operation which we can perform on a file is **"writing into a file"**.
- In order to write data into a file, we will use the **FileWriter** class and its **write()** method together. We need to close the stream using the **close()** method to retrieve the allocated resources.

Write To a File : 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();
}
} }
```

Write To a File : Example

Output :

Successfully wrote to the file.

Read from a File :

- The next operation which we can perform on a file is **"read from a file"**. In order to write data into a file, we will use the **Scanner** class.
- Here, we need to close the stream using the **close()** method. We will create an instance of the Scanner class and use the **hasNextLine()** method **nextLine()** method to get data from the file.

Read a File : 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 ReadFile {
 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();
}
} }
```

Read a File : Example

Output :

Files in Java might be tricky, but it is fun enough!

Delete a File :

- The next operation which we can perform on a file is **"deleting a file"**. In order to delete a file, we will use the **delete()** method of the file.
- We don't need to close the stream using the **close()** method because for deleting a file, we neither use the FileWriter class nor the Scanner class.

Delete a File : Example

```
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.");
 }
} }
```

Delete a File : Example

Output :

Deleted the file: filename.txt

Java BufferedReader Class :

- Java BufferedReader class is used to read the text from a character-based input stream.
- It can be used to read data line by line by readLine() method. It makes the performance fast. It inherits Reader class.
- Let's see the declaration for Java.io.BufferedReader class:

  public class BufferedReader **extends** Reader

Java BufferedReader class constructors:

BufferedReader(Reader rd)

- It is used to create a buffered character input stream that uses the default size for an input buffer.

BufferedReader(Reader rd, int size)

- It is used to create a buffered character input stream that uses the specified size for an input buffer.

Java BufferedReader class methods :

int read()

- It is used for reading a single character.

int read(char[] cbuf, int off, int len)

- It is used for reading characters into a portion of an array.

boolean markSupported()

- It is used to test the input stream support for the mark and reset method.

Java BufferedReader class methods :

boolean ready()

- It is used to test whether the input stream is ready to be read.

long skip(long n)

- It is used for skipping the characters.

void reset()

- It repositions the stream at a position the mark method was last called on this input stream.

Java BufferedReader class methods :

void mark(int readAheadLimit)

- It is used for marking the present position in a stream.

void close()

- It closes the input stream and releases any of the system resources associated with the stream.

String readLine()

- It is used for reading a line of text.

Java BufferedReader Example :

```
import java.io.*;

public class BufferedReaderExample {
   public static void main(String args[])throws Exception{
        FileReader fr=new FileReader("D:\\testout.txt");
        BufferedReader br=new BufferedReader(fr);
         int i;
        while((i=br.read())!=-1){
        System.out.print((char)i);
        }
        br.close();
        fr.close();
   }
}
```

Java BufferedWriter Class:

- Java BufferedWriter class is used to provide buffering for Writer instances.
- It makes the performance fast. It inherits Writer class. The buffering characters are used for providing the efficient writing of single arrays, characters, and strings.
- Let's see the declaration for Java.io.BufferedWriter class:

  public class BufferedWriter **extends** Writer

Java BufferedWriter Class constructors :

BufferedWriter(Writer wrt)

- It is used to create a buffered character output stream that uses the default size for an output buffer.

BufferedWriter(Writer wrt, int size)

- It is used to create a buffered character output stream that uses the specified size for an output buffer.

Java BufferedWriter Class methods :

void newLine()

- It is used to add a new line by writing a line separator.

void write(int c)

- It is used to write a single character.

void write(char[] cbuf, int off, int len)

- It is used to write a portion of an array of characters.

Java BufferedWriter Class methods :

void write(String s, int off, int len)

- It is used to write a portion of a string.

void flush()

- It is used to flushes the input stream.

void close()

- It is used to closes the input stream

Example of Java BufferedWriter :

```
import java.io.*;
public class BufferedWriterExample {
public static void main(String[] args) throws Exceptio
   n {
    FileWriter writer = new FileWriter("D:\\testout.txt");
    BufferedWriter buffer = new BufferedWriter(writer);
    buffer.write("Welcome to javaTpoint.");
    buffer.close();
    System.out.println("Success");
    }
}
```


Thank you!
shutterstock.com · 567687052