The most basic input and output in Java (`System.in` and `System.out` fields that have been used in the Basic I/O) is done using streams. Streams are objects that represent sources and destinations of data. Streams that are sources of data can be read from, and streams that are destinations of data can be written to. A stream in Java is an ordered sequence of bytes of undetermined length. Streams are ordered and in sequence so that the java virtual machine can understand and work upon the stream. Streams are analogous to water streams. They exist as a communication medium, just like electromagnetic waves in communication. The order or sequence of bytes in a Java stream allow the virtual machine to classify it among other streams.

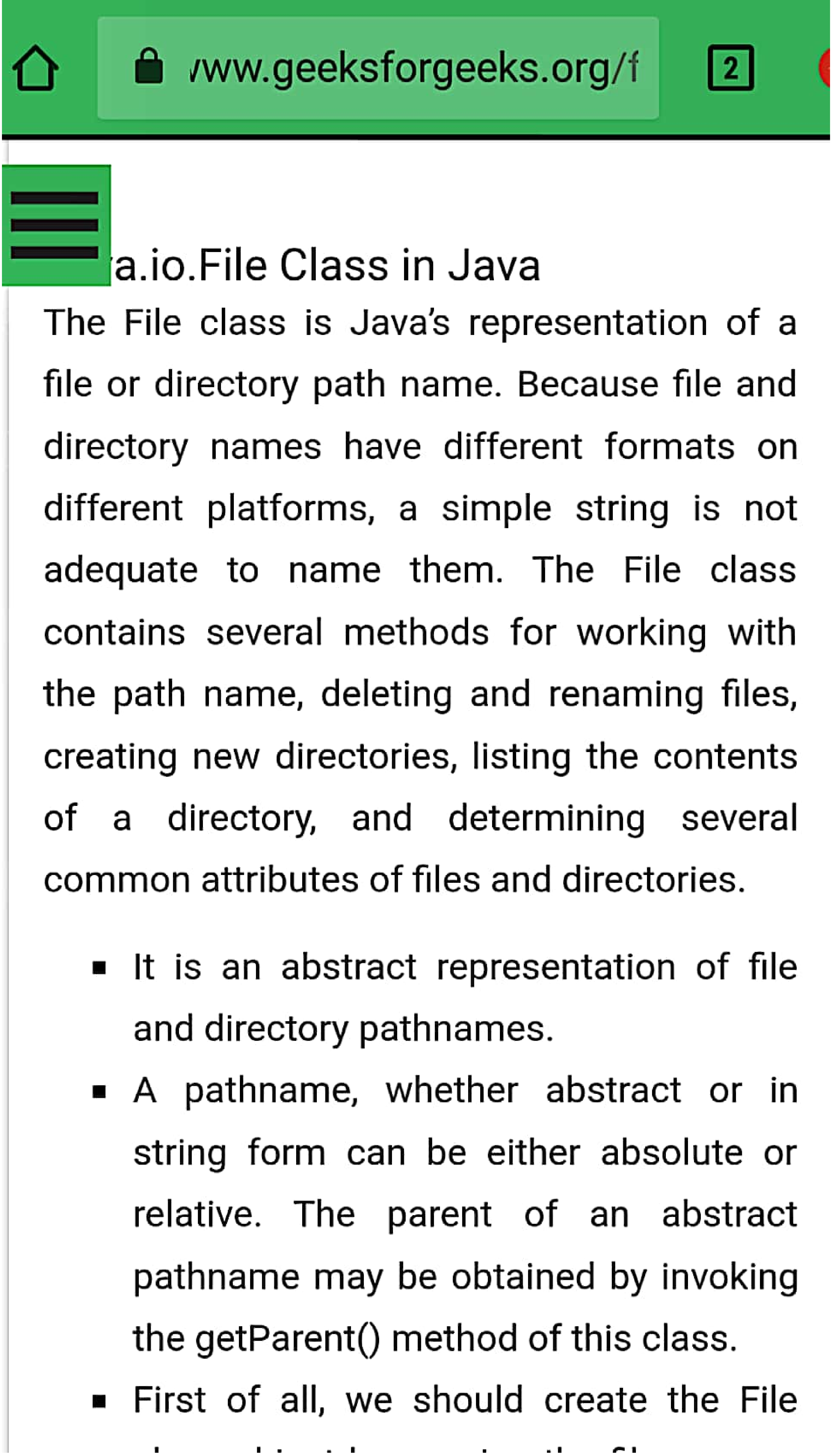
Java has various inbuilt streams implemented as classes in the package `java.io` like the

Java has various inbuilt streams implemented as classes in the package `java.io` like the classes of `System.in` and `System.out`. Streams can be classed as both input and output streams. All Java streams are derived from Input Stream (`java.io.InputStream`) and Output Stream (`java.io.OutputStream`) classes. They are abstract base classes meant for other stream classes. The `System.in` is the input stream class derivative and analogically `System.out` is the output counterpart. Both are basic classes used to directly interact with input and output through console, similarly follows `System.err`. Also Java has streams to communicate across different parts of a program or even among threads. There are also classes that "filter" streams, changing one format to another (e.g. class `DataOutputStream`, which translates various primitive types to byte streams).

Java File class represents the files and directory pathnames in an abstract manner. This class is used for creation of files and directories, file searching, file deletion, etc.

The File object represents the actual file/directory on the disk. Following is the list of constructors to create a File object.

Sr.No.	Method & Description
1	**File(File parent, String child)** This constructor creates a new File instance from a parent abstract pathname and a child pathname string.
2	**File(String pathname)** This constructor creates a new File instance by converting the given pathname string into an abstract pathname.



a.io.File Class in Java

The File class is Java's representation of a file or directory path name. Because file and directory names have different formats on different platforms, a simple string is not adequate to name them. The File class contains several methods for working with the path name, deleting and renaming files, creating new directories, listing the contents of a directory, and determining several common attributes of files and directories.

- It is an abstract representation of file and directory pathnames.
- A pathname, whether abstract or in string form can be either absolute or relative. The parent of an abstract pathname may be obtained by invoking the getParent() method of this class.
- First of all, we should create the File

pathname may be obtained by invoking the getParent() method of this class.

- First of all, we should create the File class object by passing the filename or directory name to it. A file system may implement restrictions to certain operations on the actual file-system object, such as reading, writing, and executing. These restrictions are collectively known as access permissions.
- Instances of the File class are immutable; that is, once created, the abstract pathname represented by a File object will never change.

How to create a File Object?

A File object is created by passing in a String that represents the name of a file, or a String or another File object. For example,

```
File a = new File("/usr/local/bin/g
```

defines an abstract file name for the geeks

Java Writer

It is an abstract class for writing to character streams. The methods that a subclass must implement are write(char[], int, int), flush(), and close(). Most subclasses will override some of the methods defined here to provide higher efficiency, functionality or both.

Fields

Modifier and Type	Field	Description
protected Object	lock	The object used to synchronize operations on this stream.

Constructor

Modifier	Constructor	Description

```java
mport java.io.*;
ublic class WriterExample {
  public static void main(String[] args) {
    try {
      Writer w = new FileWriter("output.txt"
      String content = "I love my country";
      w.write(content);
      w.close();
      System.out.println("Done");
    } catch (IOException e) {
      e.printStackTrace();
    }
  }
```

Output:

Java Reader

Java Reader is an abstract class for reading character streams. The only methods that a subclass must implement are read(char[], int, int) and close(). Most subclasses, however, will override some of the methods to provide higher efficiency, additional functionality, or both.

Some of the implementation class are BufferedReader, CharArrayReader, FilterReader, InputStreamReader, PipedReader, StringReader

Fields

```
port java.io.*;

blic class ReaderExample {

ublic static void main(String[] args) {

  try {

    Reader reader = new FileReader("file.tx

    int data = reader.read();

    while (data != -1) {

      System.out.print((char) data);

      data = reader.read();

    }

    reader.close();

  } catch (Exception ex) {

    System.out.println(ex.getMessage());

  }
```

```
ort java.io.*;
ss FileDemo {
  public static void main(String args[]) {
    try {
        FileReader fr=new FileReader("1.txt"
        FileWriter fw=new FileWriter("2.txt"
        int c=fr.read();
        while(c!=-1) {
          fw.write(c);
        }
    } catch(IOException e) {
        System.out.println(e);
    } finally() {
        fr.close();
        fw.close();
    }
  }
```

Scanned by CamScanner