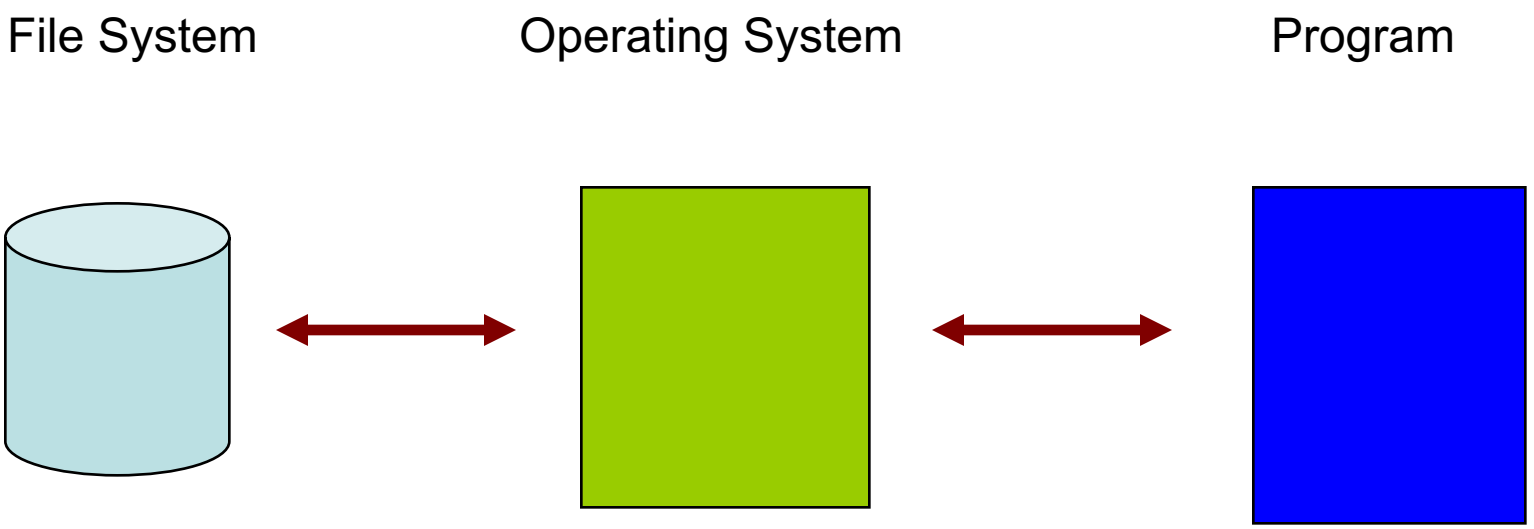
File Input and Output

File I/O is done through the Operating System

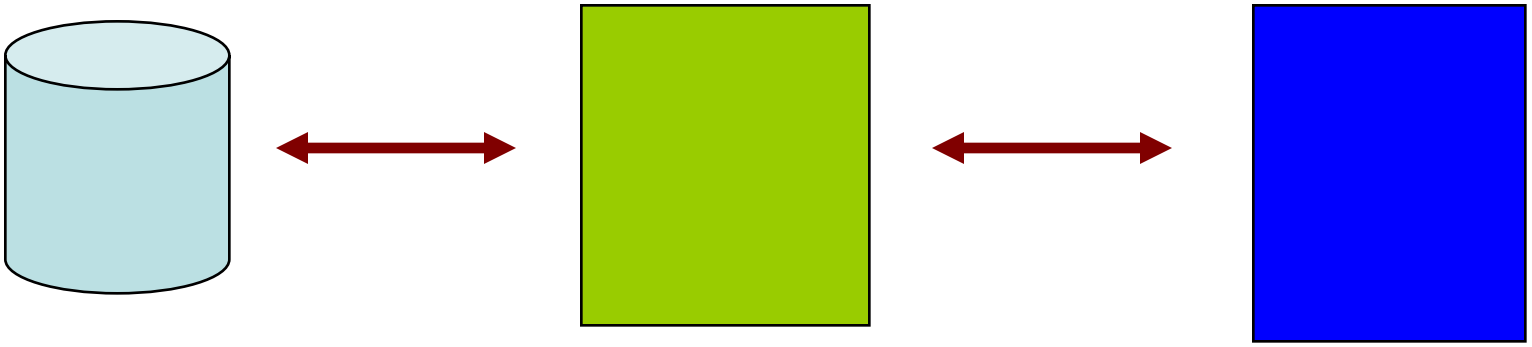


Files can be stored on a variety of devices

File System

Operating System

Program



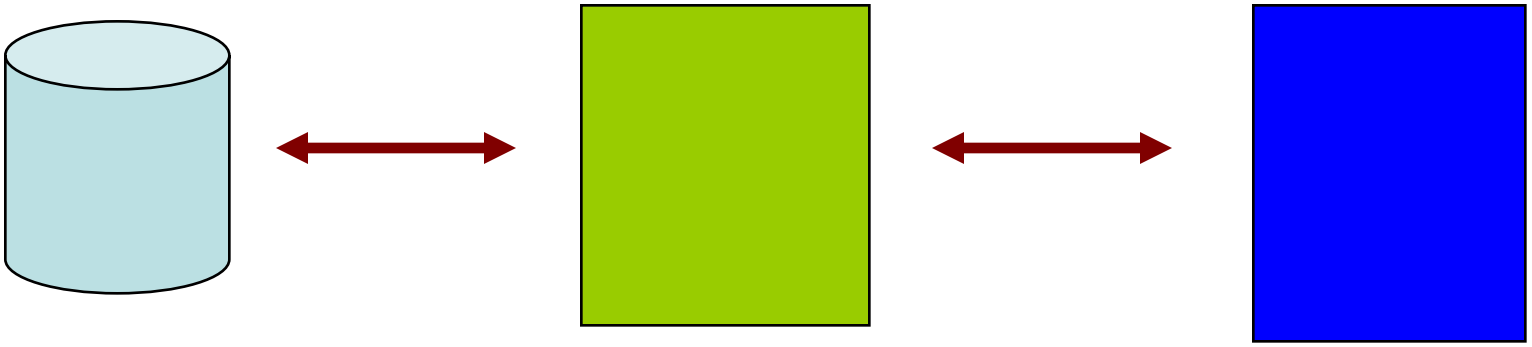
Hard drive
Jump drive
CD/DVD
Tape

Files can be read/written by many operating systems

File System

Operating System

Program



Hard drive
Jump drive
CD/DVD
Tape

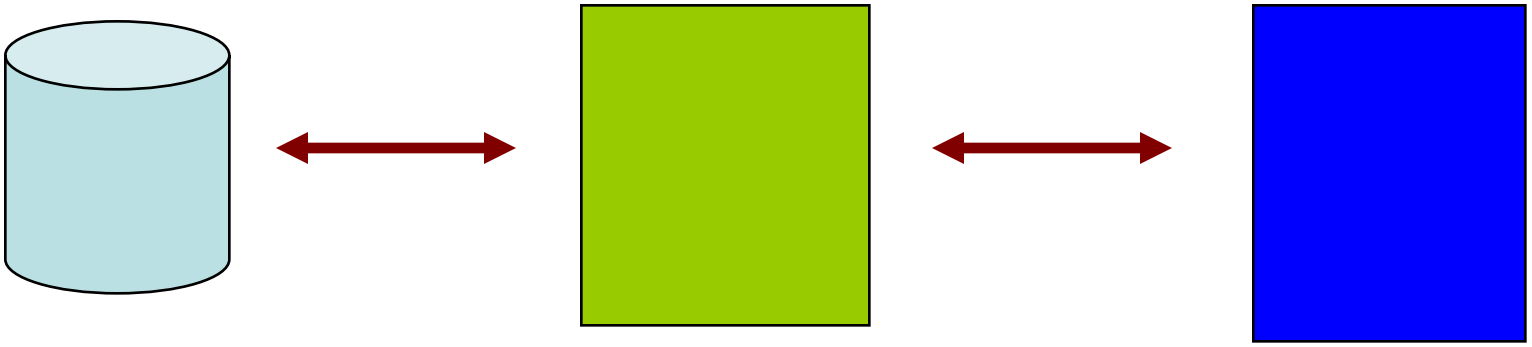
Microsoft Windows
(XP, Vista…)
Apple OSX
Linux

The OC can serve many programming languages

File System

Operating System

Program



Hard drive
Jump drive
CD/DVD
Tape

Microsoft Windows
(XP, Vista…)
Apple OSX
Linux

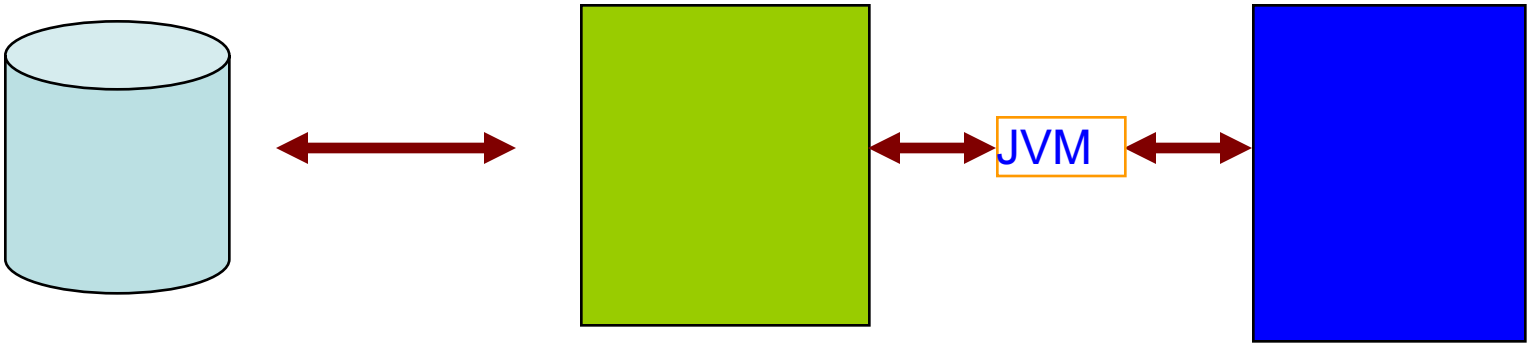
C++
Fortran
Ada
…

Including Java

File System

Operating System

Program

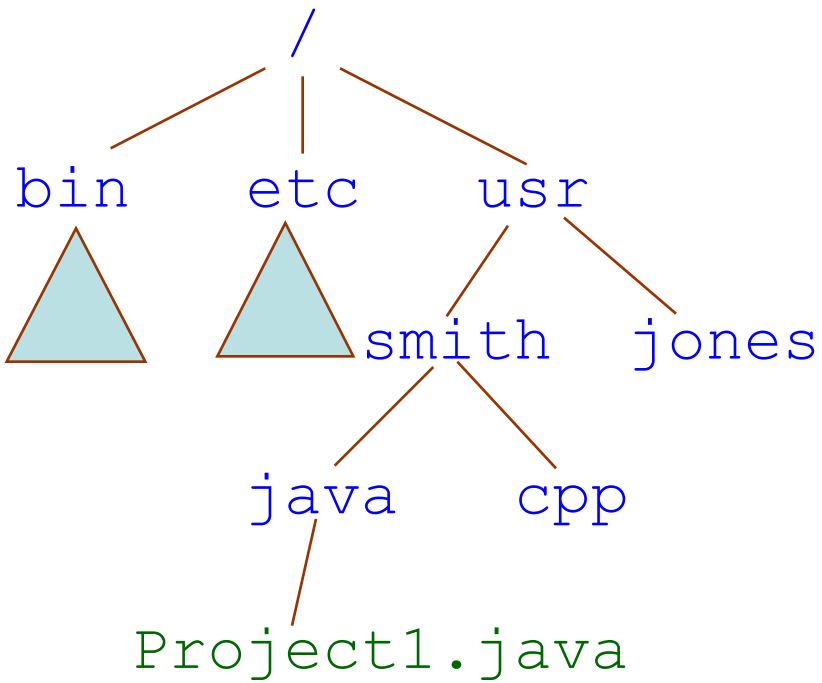


Hard drive
Jump drive
CD/DVD
Tape

Microsoft Windows
(XP, Vista…)
Apple OSX
Linux

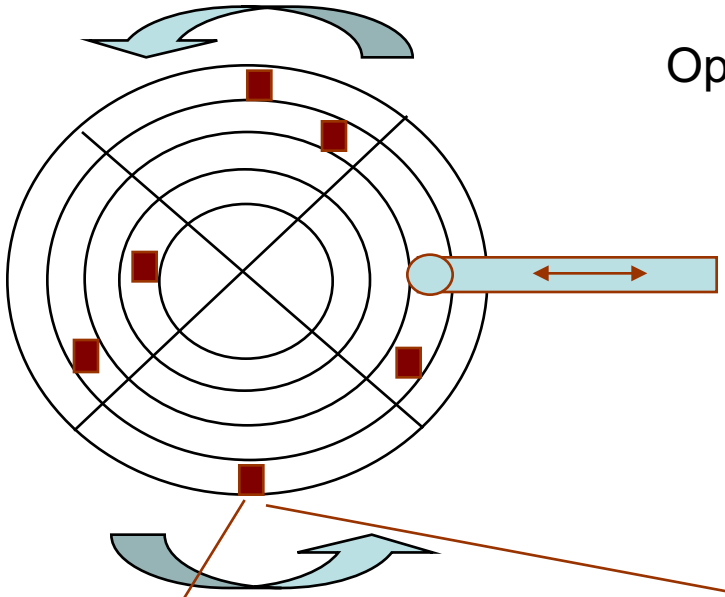
Java

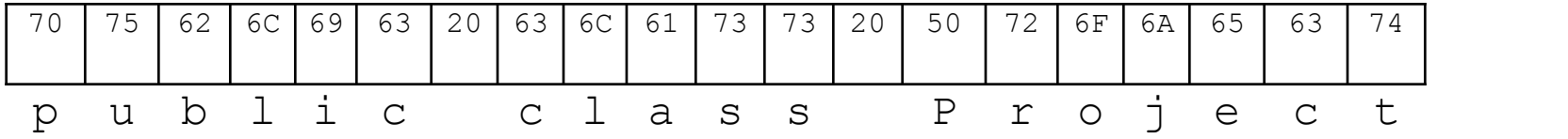
Operating System Logical View of a File



File name: `Project1.java`
File path: `/usr/smith/java/Project1.java`

Operating System Physical View of a File





java.nio.charset

Class Charset

java.lang.Object java.nio.charset.Charset

A named mapping between sequences of sixteen-bit Unicode code units and sequences of bytes.

Standard charsets

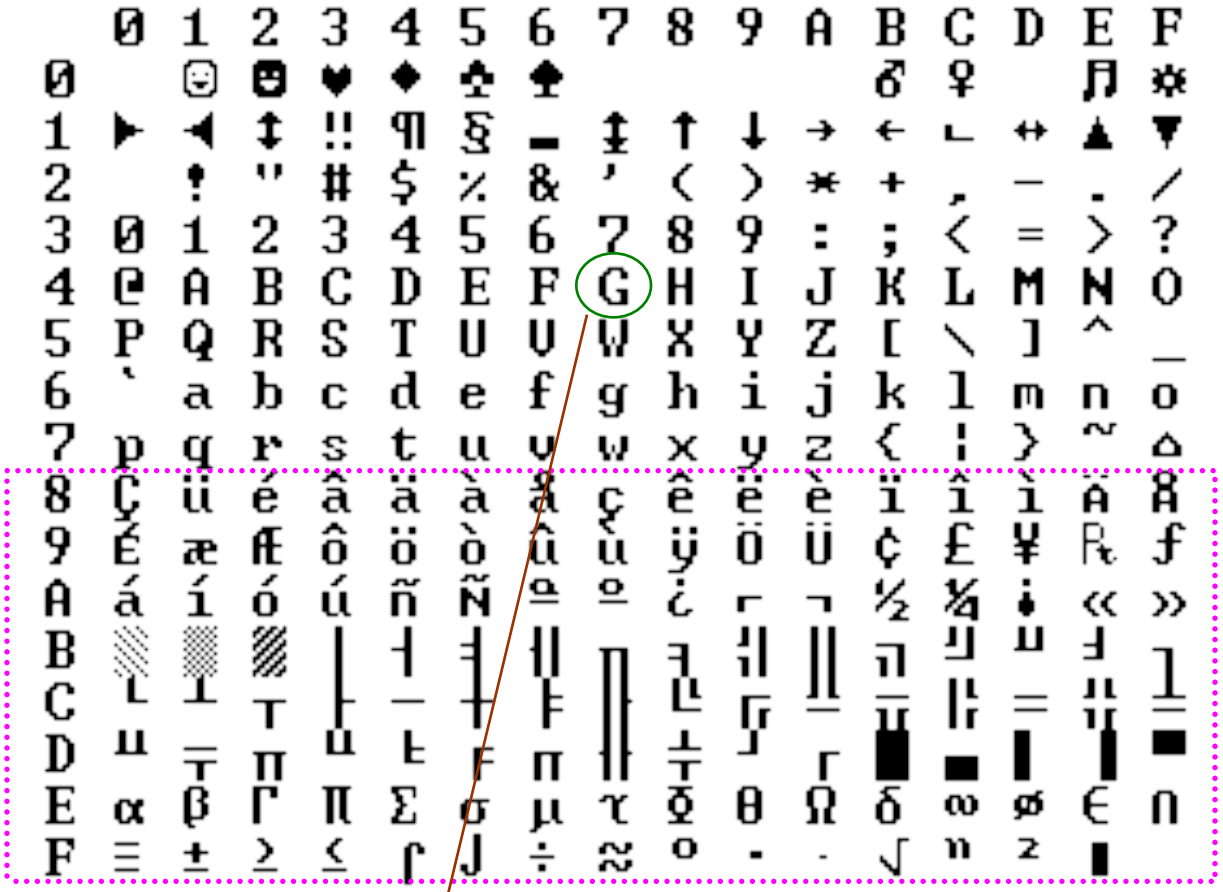
Every implementation of the Java platform is required to support the following standard charsets. Consult the release documentation for your implementation to see if any other charsets are supported. The behavior of such optional charsets may differ between implementations.

Charset	Description
US-ASCII	Seven-bit ASCII, a.k.a. ISO646-US, a.k.a. the Basic Latin block of the Unicode character set
ISO-8859-1	ISO Latin Alphabet No. 1, a.k.a. ISO-LATIN-1
UTF-8	Eight-bit UCS Transformation Format
UTF-16BE	Sixteen-bit UCS Transformation Format, big-endian byte order
UTF-16LE	Sixteen-bit UCS Transformation Format, little-endian byte order
UTF-16	Sixteen-bit UCS Transformation Format, byte order identified by an optional byte-order mark

ASCII – American Standard Code for Information Interchange

US-ASCII
00x-7Fx
0-127
00000000
01111111

ISO-LATIN-1
80x-FFx
128-255
10000000
11111111



‘G’ = 47x

Constructor for *TextFileInput*

```
public TextFileInput(String filename)
   {
      this.filename = filename;
      try  {
         br = new BufferedReader(
                  new InputStreamReader(
                      new FileInputStream(filename)));
      } catch ( IOException ioe )  {
         throw new RuntimeException(ioe);
      }  // catch
   }  // constructor
```

FileInputStream

public **FileInputStream**(String name) throws FileNotFoundException

Creates a FileInputStream by opening a connection to an actual file, the file named by the path name name in the file system. A new FileDescriptor
object is created to represent this file connection.

If the named file does not exist, is a directory rather than a regular file, or for some other reason cannot be opened for reading then a FileNotFoundException is thrown.

Parameters:

name - the system-dependent file name.

Throws:

FileNotFoundException - if the file does not exist, is a directory rather than
a regular file, or for some other reason cannot be opened for reading.

java.io

Class FileInputStream

java.lang.Object java.io.InputStream **java.io.FileInputStream**

read

public int **read**() throws IOException

Reads a byte of data from this input stream. This method blocks if no input is yet available.

```
FileInputStream fs = new FileInputStream(input.txt)
try {
    int myChar = fs.read()
}
catch (IOException ioe) {
…
}
```

myChar

00000070X

input.txt:

70	75	62	6C	69	63	20	63	6C	61	73	73	20	50	72	6F	6A	65	63	74
p	u	b	l	i	c		c	l	a	s	s		P	r	o	j	e	c	t

java.io

Class InputStreamReader

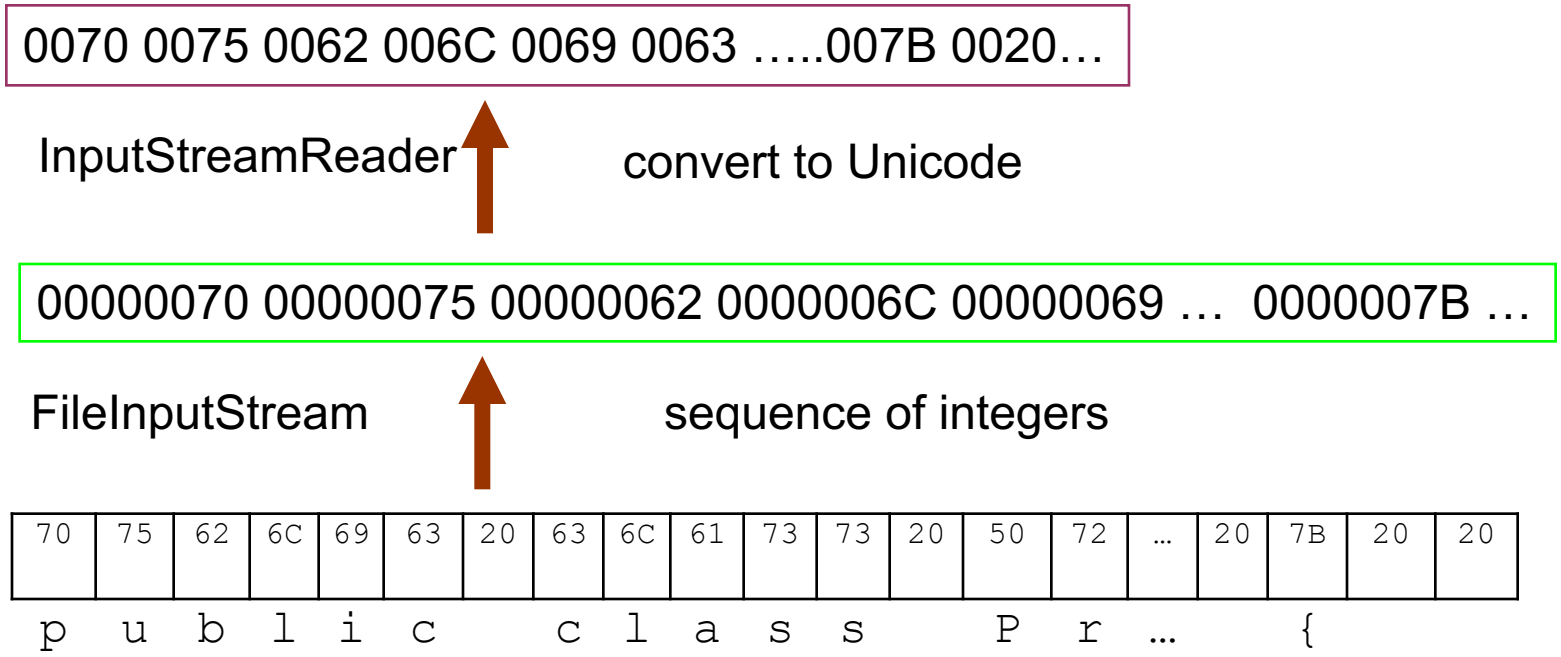
java.lang.Object java.io.Reader **java.io.InputStreamReader**

An InputStreamReader is a bridge from byte streams to character streams: It reads bytes and decodes them into characters using a specified charset. The charset that it uses may be specified by name or may be given explicitly, or the platform's default charset may be accepted.

```
public TextFileInput(String filename)
    {
       this.filename = filename;
       try  {
          br = new BufferedReader(
                    new InputStreamReader(
                        new FileInputStream(filename)));
       } catch ( IOException ioe )  {
          throw new RuntimeException(ioe);
       }  // catch
    }  // constructor
```

The *FileInputStream* reads ASCII bytes from the file and delivers a stream of 32-bit *int* valules.

The *InputStreamReader* converts the *int*s to a stream of Unicode characters (the default character set).



java.io

Class BufferedReader

java.lang.Object java.io.Reader **java.io.BufferedReader**

readLine

public String **readLine**() throws IOException

Read a line of text. A line is considered to be terminated by any one of a line feed ('\n'), a carriage return ('\r'), or a carriage return followed immediately by a linefeed.

Returns:

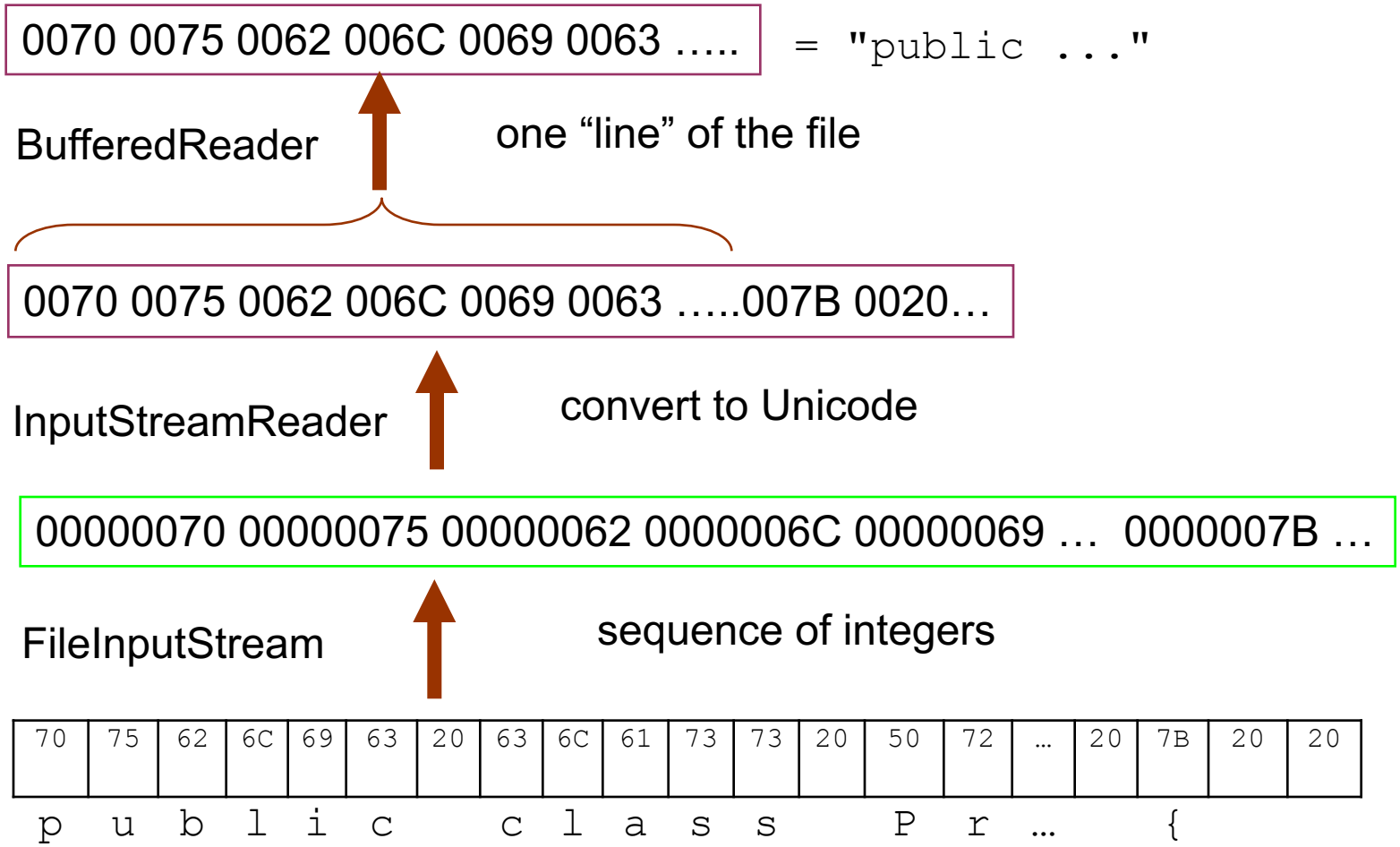
A String containing the contents of the line, not including any line-termination characters, or null if the end of the stream has been reached

Throws:

IOException - If an I/O error occurs

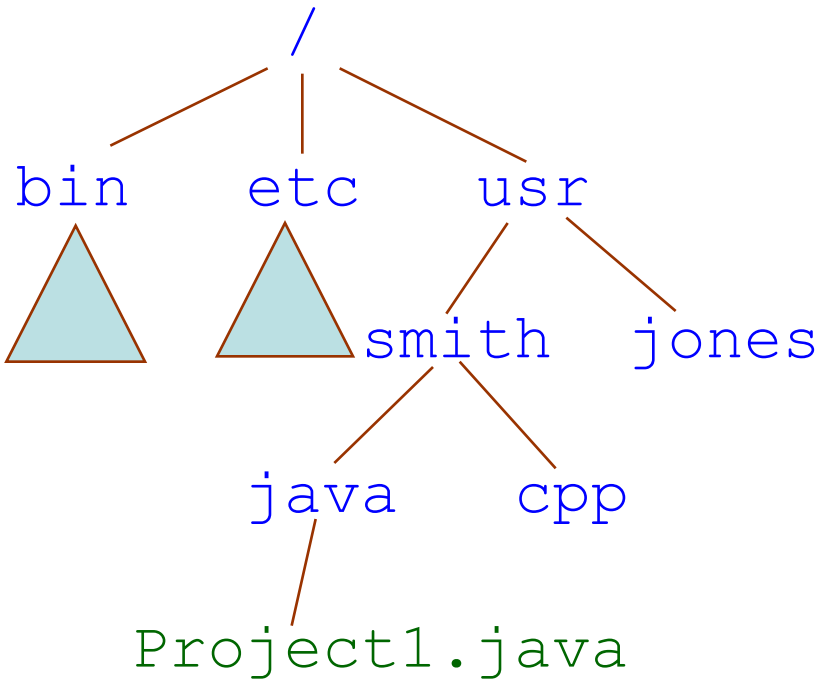
```
try  {
   br = new BufferedReader(
            new InputStreamReader(
                new FileInputStream(filename)));
} catch ( IOException ioe )  {
   throw new RuntimeException(ioe);
}  // catch
```

The *BufferedReader* separates the stream of Unicode characters into "lines" of the file (a line is terminated with *lineFeed \n, carriageReturn \r or \n\r.*



class *File*

An abstract representation of file and directory pathnames.




```
File myFile = new File("/usr/smith/java/Project1.java");
```

```
File myFile = new File("/usr/smith/java/Project1.java");
```

The object *myFile* refers to the operating system's file on disk.

What can we do with this *File* object?

```
import java.io.File;
import javax.swing.*;

public class SingleFile {
   public static void main (String args[]){
   JFileChooser fileChooser = new JFileChooser();
   fileChooser.showOpenDialog(null);
   File myFile = fileChooser.getSelectedFile();
   System.out.println("getName(): "+myFile.getName());
   System.out.println("getParent(): "+myFile.getParent());
   System.out.println("getPath(): "+myFile.getPath());
   System.out.println("lastModified(): "+myFile.lastModified());
   System.out.println("length(): "+myFile.length());
   }
}
```

```
import java.io.File;
import javax.swing.*;
public class ListFiles {

         public static void main(String[] args) {
        JFileChooser fd = new JFileChooser();
//        mode - the type of files to be displayed:
//            * JFileChooser.FILES_ONLY
//            * JFileChooser.DIRECTORIES_ONLY
//            * JFileChooser.FILES_AND_DIRECTORIES
        fd.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
        fd.showOpenDialog(null);
                 File f = fd.getSelectedFile();
                 listFiles(f,"");
         }
         public static void listFiles(File f, String indent) {
                 File files[] = f.listFiles();

                 for (int i = 0; i<files.length; i++) {
                         File f2 = files[i];
                         System.out.print(f2.getName());
                         if (f2.isDirectory())
                                 listFiles(f2, indent+"   ");
                         System.out.print("...");
                         System.out.print(f2.length());
                         System.out.println();
                 }
         }
}
```