COMP 110-001
Streams and File I/O

Yi Hong

June 10, 2015

Today

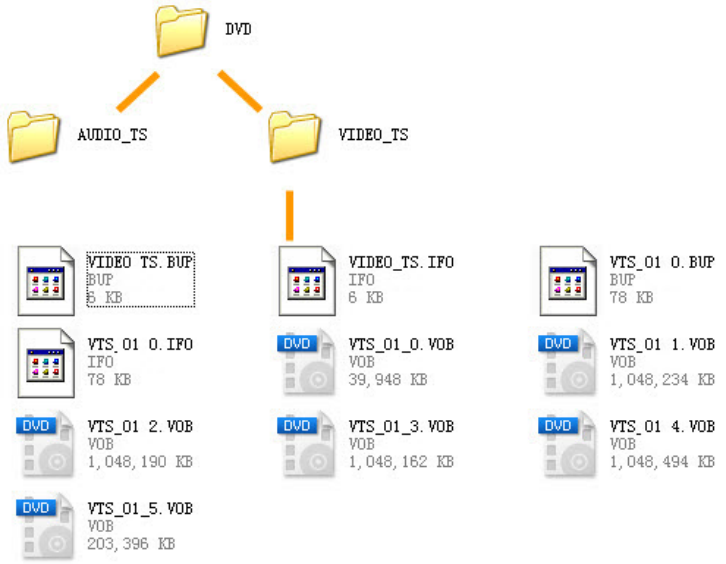
- Files, Directories, Path
- Streams
- Reading from a file
- Writing to a file

Why Use Files for I/O?

- RAM is not persistent
- Data in a file remains after program execution, stored permanently

Working With Files

- The data stored in these persistent storage are normally in the form of files
- Have you tried to open a movie DVD in your computer using file explorer?
- You will probably see some folders and files like this:



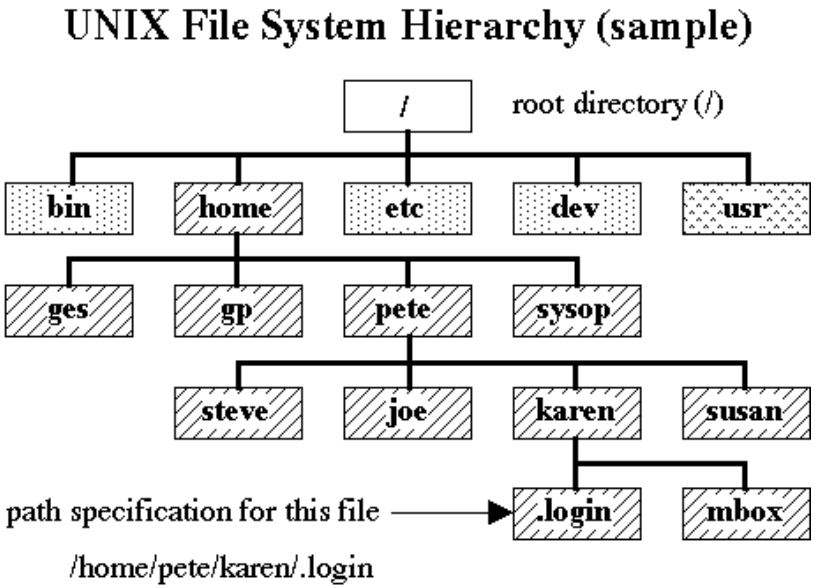
Working With Files

- In short:
  - We often need to write the data to files to store it
  - We often need to read the data from files

- We will cover some basics about files and directories in Windows / Linux & Mac OS first

Files and Directories

- Files are stored in directories or folders in a tree structure
- A directory can contain one or more files and/or directories
- The root directory in Windows is the drive name (C: or D: , don't miss the :)
- The root directory in Unix/ Linux/MacOS is /



Files and Directories: Path to File

- A file is identified by its path through the file system, beginning from the root node
  - Linux/Unix: e.g., /home/yihong/Music
  - MacOS: e.g., /Users/yihong/Music
  - Windows: e.g., C:\Users\yihong\Music

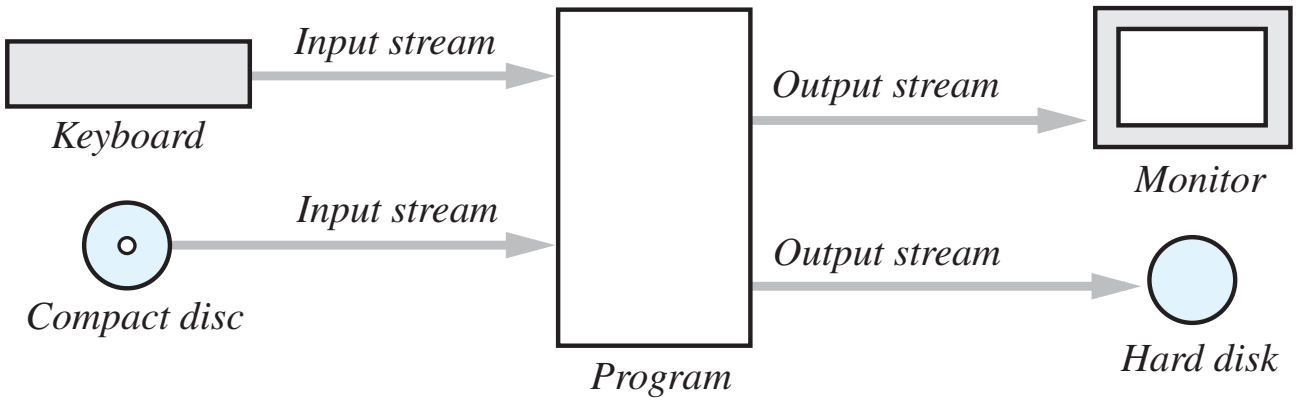
- The character used to separate the directory names (also called the *delimiter*) is forward slash (/) in Linux/Unix/MacOS, and backslash slash (\) in Windows.

Relative and Absolute Path

- A path is either *relative* or *absolute*
  - An absolute path always contains the root element and the complete directory list required to locate the file
  - e.g.: /Users/yihong/Music
- A relative path needs to be combined with another path in order to access a file
  - e.g. yihong/Music is a relative path
  - Without more information, a program cannot reliably locate the yihong/Music directory in the file system
- In java, when you write a relative path, it's relative to the working directory

Java's Input/Output Mechanism

- A stream is a flow of data into or out of a program



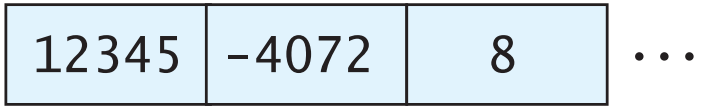
- Very complicated design based on "streams"
- Here, we focus on how to use input and out streams

Text Files v.s. Binary Files

- Text file: a sequence of characters
- Binary file: pack values into binary representation

A text file

A binary file



- We only cover text file I/O in this course

Creating a Text File

- Opening a file connects it to a stream
- The class PrintWriter in the package java.io is for writing to a text file

```
String fileName = "out.txt";//Could read file name from user
PrintWriter outputStream = null;
try
{
    outputStream = new PrintWriter(fileName);
}
catch(FileNotFoundException e)
{
    System.out.println("Error opening the file " + fileName);
    System.exit(0);
}
```

Creating a Text File

- After we connect the file to the stream, we can write data to it
  - outputStream.println(“This is line 1.”);
  - outputStream.println(“Here is line 2.”);
- Closing a file disconnects it from a stream
  - outputStream.close();

Creating a Text File

- Syntax

```
// Open the file PrintWriter
Output_Stream_Name = null;
try
{
    Output_Stream_Name = new PrintWriter(File_Name);
}
catch(FileNotFoundException e)
{
    Statements_Dealing_With_The_Exception
}

// Write the file using statements of either or both of the
// following forms:
Output_Stream_Name.println(...);
Output_Stream_Name.print(...);
// Close the file
Output_Stream_Name.close();
```

Example

```
String fileName = "out.txt";
PrintWriter outputStream = null;

try
{
    outputStream = new PrintWriter(fileName);
}
catch(FileNotFoundException e)
{
    System.out.println("Error opening the file " + fileName);
    System.exit(0);
}

System.out.println("Enter three lines of text: ");
Scanner keyboard = new Scanner(System.in);
for(int count = 1; count <= 3; count++)
{
    String line = keyboard.nextLine();
    outputStream.println(count + " " + line);
}
outputStream.close();
System.out.println("Those lines were written to " + fileName);

keyboard.close();
```

Appending to a Text File

- Adding data to the end of a file
- Syntax

```
PrintWriter Output_Stream_Name = new PrintWriter(new
FileOutputStream(File_Name, true));
```

- Example

```
PrintWriter outputStream = new PrintWriter(new
FileOutputStream("out.txt", true));
```

Reading From a Text File

- Use Scanner to open a text file for input

```
Scanner Stream_Name = new Scanner(new File(File_Name));
```

  - E.g.: Scanner inputStream = new Scanner(new File("out.txt"));

- Use the method hasNextLine to read

```
while (inputStream.hasNextLine())
{
    String line = inputStream.nextLine();
    System.out.println(line);
}
```

Reading From a Text File

- Syntax

```
// Open the file
Scanner Input_Stream_Name = null;
try
{
    Input_Stream_Name = new Scanner(new File(File_Name));
}
catch(FileNotFoundException e)
{
    Statements_Dealing_With_The_Exception
}
// Read the file using statements of the form:
Input_Stream_Name.Scanner_Method();
// Close the file
Input_Stream_Name.close();
```

Example

```
Scanner inputStream = null;
try
{
    inputStream = new Scanner(new File(fileName));
}
catch(FileNotFoundException e)
{
    System.out.println("Error opening the file " + fileName);
    System.exit(0);
}
while(inputStream.hasNextLine())
{
    String line = inputStream.nextLine();
    System.out.println(line);
}
inputStream.close();
```

Other Techniques

- The class File provides a way to represent file names in a general way
  - E.g.: new File(“out.txt”) – Create a File object represents the name of a file
- Let the user enter the file name at the keyboard
  - E.g.: String fileName = keyboard.next();
- Use Path Names
  - A path name specifies the folder containing a file
  - E.g.: Scanner inputStream = new Scanner(new File(“/User/yihong/out.txt”));

Help on Homework 4

Next Class

- Lab 8