# Executing programs from the "command line" and command line arguments

March 18, 2005

Most of you are familiar with an operating system such as Windows or MacOS. These operating systems have a "Graphical User Interface," this is, the way the user interacts with the computer is through graphical elements such as windows, menus, dialog boxes, buttons, scrollbars, etc.

Most operating systems usually provide an alternative, non-graphical way to interact with the user. These "text-based user interfaces" are less sophisticated, but allow the user to do everything by typing commands on the so-called "command line" or "command prompt."

## 1 The command prompt

On Windows you can access the command prompt through the "Accessories" sub-menu from the "Start" menu. It is sometimes called the "Console" or the "MS-DOS prompt."

Once you start it, you will see a black window with a prompt that looks like this:

```
C:\Documents and Settings\User Name> _
```

This is the "prompt" where you can type commands.

The "`C:`" refers to the hard disk where you are currently located. The "`\Documents and Settings\User Name`" refers to the subdirectory where you are. The "`_`" is the cursor where you type your commands.

A disk (such as a hard disk) is divided into "folders" or directories. Each folder or directory can contain files, or other folders and directories. A *path* is a description of a directory or file in a hard drive. "`C:\Documents and Settings\User Name`" is a path, where "`User Name`" is a subdirectory of "`Documents and Settings`". This means that directories are organized in a "tree." The "root" directory is "\". For example, a typical directory structure is shown in Figure 1. In this example, the directory `User1` contains two subdirectories "`My Documents`" and "`My Programs`" which in turn contain files and directories.

Other operating systems have different conventions to describe file and directory paths. On Unix/Linux, you do not write the name of the disk drive

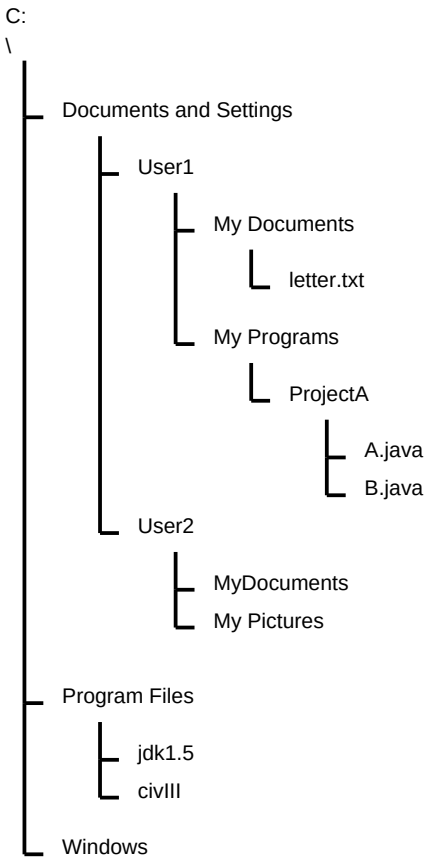


Figure 1: A typical directory tree

"C:" in front of the path, and the backslash "\" is replaced by a normal slash character "/".

# 2 Commands

There are many commands that you can type at a command prompt, but they depend on the operating system that you have. Here we describe a few commands available under Windows.

## 2.1 File system commands

- `dir`: displays a list of files and directories in the current directory
- `cd` *path*: changes the current directory to path
- `copy` *source target*: copies the file source into the path specified by target
- `del` *file*: deletes a file

For example:

```
cd C:\docs\text
```

or

```
cd "C:\My Documents\My Spreadsheets\Taxes"
```

Note: if the path contains space characters it must be enclosed in double quotes.

A path that starts with "C:\" or just with "\" is called an *absolute path*, and it describes the location of a file or directory with respect to the root directory. A *relative path* is one that does not begin with "\" or "C:\". A relative path refers to the location of a file or directory with respect to the current directory.

Suppose that you have a directory structure as picture in the previous section. Figure 2 shows a "session" where the user types commands on the prompt and the results are shown.

You may have noticed that each directory has to entries "." and ".." on top. These entries refer to the current directory, and the *parent* directory. They could be used in any path description, and in particular they can be used to move quickly to the parent directory. For example, in the same directory structure, we could have the following:

```
C:\Documents and Settings\User1\My Documents> cd ..
C:\Documents and Settings\User1> cd ..
C:\Documents and Settings> cd "User2\My Pictures"
C:\Documents and Settings\User2\My Pyctures> cd "..\..\User1\My Programs"
C:\Documents and Settings\User1\My Programs> _
```

```
C:\Documents and Settings\User1> dir
  Volume in drive C has no label.
  Volume Serial Number is 1643-0CD7

  Directory of C:\Documents and Settings\User1

07/10/2004  08:06 PM  <DIR>          .
07/10/2004  08:06 PM  <DIR>          ..
07/10/2004  08:06 PM  <DIR>      512 My Documents
07/10/2004  08:06 PM  <DIR>      512 My Programs
               0 File(s)           0 bytes
               2 Dir(s)         1024 bytes

C:\Documents and Settings\User1> cd "My Programs\ProjectA"
C:\Documents and Settings\User1\My Programs\ProjectA> dir
  Volume in drive C has no label.
  Volume Serial Number is 1643-0CD7

  Directory of C:\Documents and Settings\User1\My Programs\ProjectA

07/10/2004  08:06 PM  <DIR>          .
07/10/2004  08:06 PM  <DIR>          ..
02/10/2005  09:09 PM           2,049 A.java
02/10/2005  09:09 PM           4,097 B.java
               2 File(s)       6,146 bytes
               0 Dir(s)            0 bytes

C:\Documents and Settings\User1\My Programs\ProjectA> cd "\Program Files\jdk1.5"
C:\Program Files\jdk1.5> _
```

Figure 2: A typical command line session

In other operating systems such commands have different names. The following table shows a correspondance between these commands in Windows and Unix/Linux:

| Windows commands | Unix/Linux commands |
|---|---|
| `dir` | `ls` |
| `cd` | `cd` |
| `copy` | `cp` |
| `del` | `rm` |

## 2.2 Executing programs

Aside from commands to navigate through the directory structure and manipulate files, you can also execute any program from the command line. Under windows, programs are "executable files" which end in "`.exe`". To execute a program you can do it in different ways:

- First go to the directory where it is located, and then executing it by typing the name of the executable file, or
- Give the absolute path to the program.

For example. Suppose that there is a program called "`civ.exe`" stored in the directory "`C:\Program Files\civIII`". To execute it you could type

```
cd "C:\Program Files\civIII"
civ.exe
```

or

```
C:\Program Files\civIII\civ.exe
```

or simply

```
C:\Program Files\civIII\civ
```

The extension "`.exe`" is optional when you want to execute a program.

## 2.3 The `PATH` environment variable

When you execute a program, the operating system will search for the program's executable file in the directory where you are currently located. If the executable file is not there, the operating system will look for it in other directories specified by the so-called "`PATH`" environment variable. If the program is not found in any of those directories, a message saying that the command was not found will be displayed.

For example, suppose that you are in the directory "`C:\Program Files\civIII`", and it contains the executable file "`civ.exe`". Then the following will execute the program civ:

```
C:\Program Files\civIII> civ
```

Now, suppose that you are in "`C:\Documents and Settings\User1`", and you try to execute "`civ.exe`" but "`C:\Program Files\civIII`" is not in the `PATH`. Then you will get an error message:

```
C:\Documents and Settings\User1> civ
Command not found
```

If you want to be able to execute the program from that directory you could write its full path:

```
C:\Documents and Settings\User1> \Program Files\civIII\civ
```

But if you use this program very often and you want to be able to execute it from any directory, writing the full path can become annoying. Instead, you could add the "`C:\Program Files\civIII`" directory to the `PATH` environment variable:

```
C:\Documents and Settings\User1> set PATH = %PATH%;"C:\Program Files\civIII"
```

This should append the directory to the current `PATH`, so now you can execute the file from any directory:

```
C:\Documents and Settings\User1> civ
```

However, modifying the `PATH` in such way will affect the current session only, and it will be forgotten the next time you start a different session. The simplest way of making sure that the `PATH` is always set correctly on Windows 2000 and Windows XP is to do the following:

1. From the desktop right click the "My Computer" icon, and click "Properties".
2. In the "System Properties" window click on the "Advanced" tab.
3. In the "Advanced" section click the "Environment Variables" button.
4. Finally, in the "Environment Variables" window highlight the `PATH` variable in the "Systems Variables" section and click "Edit". Add or modify the path lines with the paths you wish. Each different directory is separated with a semicolon (;).
5. Click "OK".

# 3 Compiling and executing Java programs from the command line

The Java compiler and the Java runtime system (the Virtual Machine that executes Java programs) are themselves executable programs, called "`javac.exe`" and "`java.exe`" respectively. They are located in the "`bin`" subdirectory of the directory where you installed Java. For instance, if you installed Java on "`C:\Program Files\jdk1.5`" then "`javac`" and "`java`" are located in "`C:\Program Files\jdk1.5\bin`".

Using these, you can compile and run a file from the command line.

For instance, if your program is called `MyProgram.java` and is located in "`C:\My Files\My Programs\ProjectA`" then you can do the following:

1. Change the directory to the location of your project:

   ```
   cd "C:\My Files\My Programs\ProjectA"
   ```

2. Then compile each java file

   ```
   javac MyProgram.java
   ```

   If you get a message saying that javac was not found, then type the full path of `javac`:

   ```
   C:\Program Files\jdk1.5\bin\javac MyProgram.java
   ```

   or add the Java "`bin`" directory the the `PATH` environment variable as described in section 2.3.

   You should compile each source file of your program. But this can be done in one step by typing:

   ```
   javac *.java
   ```

   which compiles all source files in that directory. This generates the bytecode files (.class files) which you can now run

3. Run the program: if the main method was in class `MyProgram`, then you can type

   ```
   java MyProgram
   ```

   (no need for .class) or

   ```
   C:\Program Files\jdk1.5\bin\java MyProgram
   ```

```
class MyProgram
{
  public static void main(String[] args)
  {
    int i = 0;
    while (i < args.length)
    {
      System.out.println(args[i]);
      i++;
    }
  }
}
```

Figure 3: Processing command-line arguments

# 4 Program arguments

Programs executed on the command line can receive arguments. For example:

```
java MyProgram arg1 arg2 arg3 etc
```

In Java those arguments are received by the main method. The main method has a parameter which is an array of strings. Each element of this array will be assigned each argument from left to right.

The program in Figure 4 prints all the arguments (if any) passed to the program.

So when we execute

```
java MyProgram arg1 arg2 arg3 etc
```

it will print

```
arg1
arg2
arg3
etc
```

This can be used to make your program do things depending on the arguments received on the command line. An example is shown in Figure 4.

If we execute the program we could see the results:

```
C:\Documents and Settings\User1\My Programs> java MyProgram2 -version
MyProgram2 v1.0
```

```
class MyProgram2
{
  public static void main(String[] args)
  {
    if (args.length > 0)
    {
      if (args[0].equals(“-version”))
      {
        System.out.println(“MyProgram2 v1.0”);
      }
      else if (args[0].equals(“-help”))
      {
        System.out.println(“Usage: java MyProgram2 [OPTION]”);
        System.out.println(“Where [OPTION] is one of:”);
        System.out.println(“  -version: prints the version number”);
        System.out.println(“  -help:    prints this message”);
      }
    }
  }
}
```

Figure 4: Processing command-line arguments

```
C:\Documents and Settings\User1\My Programs> java MyProgram2 -help
Usage: java MyProgram2 [OPTION]
Where [OPTION] is one of:
  -version: prints the version number
  -help:    prints this message
C:\Documents and Settings\User1\My Programs> _
```