



OOP with Java

11. Command Line Arguments

Thomas Weise · 汤卫思

tweise@hfu.edu.cn · <http://iao.hfu.edu.cn>

Hefei University, South Campus 2
Faculty of Computer Science and Technology
Institute of Applied Optimization
230601 Shushan District, Hefei, Anhui, China
Econ. & Tech. Devel. Zone, Jinxiu Dadao 99

合肥学院 南艳湖校区/南2区
计算机科学与技术系
应用优化研究所
中国 安徽省 合肥市 蜀山区 230601
经济技术开发区 锦绣大道99号

- 1 Introduction
- 2 Processing Command Line Arguments
- 3 Specifying Command Line Arguments in Eclipse
- 4 Summary



website

- Usually, a console program takes some arguments

- Usually, a console program takes some arguments
- Under linux, `echo bla` runs program `echo` and passes argument `bla` to it, causing it to print `bla` to its stdout

- Usually, a console program takes some arguments
- Under linux, `echo bla` runs program `echo` and passes argument `bla` to it, causing it to print `bla` to its stdout
- `java -version` runs the program `java` and passes to it the argument `-version`

- Usually, a console program takes some arguments
- Under linux, `echo bla` runs program `echo` and passes argument `bla` to it, causing it to print `bla` to its stdout
- `java -version` runs the program `java` and passes to it the argument `-version`
- `java PrintCommandLineArguments` runs the program `java` and passes to it the argument `PrintCommandLineArguments` (which tells it to start the JVM and executes program `PrintCommandLineArguments`)

- Usually, a console program takes some arguments
- Under linux, `echo bla` runs program `echo` and passes argument `bla` to it, causing it to print `bla` to its stdout
- `java -version` runs the program `java` and passes to it the argument `-version`
- `java PrintCommandLineArguments` runs the program `java` and passes to it the argument `PrintCommandLineArguments` (which tells it to start the JVM and executes program `PrintCommandLineArguments`)
- `java PrintCommandLineArguments A B` runs the program `java` and passes to it the arguments `PrintCommandLineArguments` , `A` , and `B` (which it to start the JVM and executes program `PrintCommandLineArguments` with command line arguments `A` and `B`)

- Usually, a console program takes some arguments
- Under linux, `echo bla` runs program `echo` and passes argument `bla` to it, causing it to print `bla` to its stdout
- `java -version` runs the program `java` and passes to it the argument `-version`
- `java PrintCommandLineArguments` runs the program `java` and passes to it the argument `PrintCommandLineArguments` (which tells it to start the JVM and executes program `PrintCommandLineArguments`)
- `java PrintCommandLineArguments A B` runs the program `java` and passes to it the arguments `PrintCommandLineArguments` , `A` , and `B` (which it to start the JVM and executes program `PrintCommandLineArguments` with command line arguments `A` and `B`)
- Besides via `stdin`, we now have a second option to pass in data to a program

- Usually, a console program takes some arguments
- Under linux, `echo bla` runs program `echo` and passes argument `bla` to it, causing it to print `bla` to its stdout
- `java -version` runs the program `java` and passes to it the argument `-version`
- `java PrintCommandLineArguments` runs the program `java` and passes to it the argument `PrintCommandLineArguments` (which tells it to start the JVM and executes program `PrintCommandLineArguments`)
- `java PrintCommandLineArguments A B` runs the program `java` and passes to it the arguments `PrintCommandLineArguments` , `A` , and `B` (which it to start the JVM and executes program `PrintCommandLineArguments` with command line arguments `A` and `B`)
- Besides via `stdin`, we now have a second option to pass in data to a program
- Command line arguments are used to set parameters and pass in data of small size, `stdin` can be used for arbitrary size unstructured data

- the `main` method of any Java program takes one parameter `args` which is a String array

- the `main` method of any Java program takes one parameter `args` which is a String array
- this parameter contains the command line arguments passed to the program, each one as a String

Listing: Printing Command Line Arguments.

```
/** Example for printing the command line arguments<br/>
 * Eclipse: "Run As" -> "Run Configurations..." -> Java Application
 *         -> new -> (x)= arguments -> program arguments
 *         (type arguments space-separated in that field) */
public class PrintCommandLineArgs {

    /**The main routine
     * @param args
     *     the command line arguments of the program (no longer ignored `~`) */
    public static final void main(String[] args) {
        System.err.println("There were " + //$NON-NLS-1$
            args.length + // the number of command line arguments
            " command line arguments."); //$NON-NLS-1$

        for (String arg : args) { // iterate over the command line arguments
            System.out.println(arg); // print the current iteration element
        }
    }
}
```

Listing: Greetings Printer Revised: Now with Command Line Arguments.

```
/** Examples for using command line arguments (and if-then-else) */
public class HelloSwitchCaseArgs {

    /** The main routine
     * @param args
     *         the command line arguments of the program (no longer ignored ^_^) */
    public static final void main(String[] args) {

        if (args.length < 2) { // check if there are at least two arguments, if not, print help and exit
            System.out.println("Please specify two arguments: your family name and your gender."); //NON-NLS-1$
        } else { // OK, we have at least two command line arguments
            System.out.print("Hello "); // just print hello //NON-NLS-1$
            switch (args[1]) { // choose what to do based on gender provided as second command line argument
                case "f": //NON-NLS-1$
                    case "F": { // we will get here if gender is either "f" or "F" //NON-NLS-1$
                        System.out.print("Mrs. "); //NON-NLS-1$
                        break;
                    }

                case "m": //NON-NLS-1$
                    case "M": { // we will get here if gender is either "m" or "M" //NON-NLS-1$
                        System.out.print("Mr. "); //NON-NLS-1$
                        break;
                    }

                default: { // we will get here if the gender is neither "f", "F", "m", "M"
                    System.out.print(args[1]);
                    System.out.print(' ');
                    break;
                }
            }
            System.out.println(args[0]); // print family name, the first command line arg
        }
    }
}
```

How to Specify Command Line Args in Eclipse

A screenshot of the Eclipse IDE interface. The Package Explorer on the left shows a project named '10_command_line_args' with a sub-package 'javaExample'. Inside 'javaExample', a class named 'PrintCommandLineArguments.java' is selected, and a context menu is open over it. The menu items include 'New', 'Open', 'Open With', 'Open Type Hierarchy', 'Show In', 'Copy', 'Copy Qualified Name', 'Paste', 'Delete', 'Remove From Context', 'Build Path', 'Source', 'Refactor', 'Import...', 'Export...', 'References', 'Declarations', 'Refresh', 'Assign Working Sets...', 'Run As', 'Debug As', 'Validate', 'Restore from Local History...', 'Team', 'Compare With', 'Replace With', and 'Properties'. The 'Run As' option is highlighted, and a sub-menu is visible with 'Java Application' and 'Run Configurations...' options. The 'Run Configurations...' option is also highlighted. The main editor window shows the source code of 'PrintCommandLineArguments.java'. The code includes a package declaration, imports, a class declaration, and a main method that prints the command line arguments. The console at the bottom shows the output of the program: 'PrintCommandLineArguments [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bin/java (Feb 3, 2017, 6:01:46 AM) line arguments.'

```
1 /** Example for printing the command line arguments-  
2  * Eclipse: "Run As" -> "Run Configurations..." -> Java Application  
3  * -> new -> (x)= arguments -> program arguments  
4  * (type arguments space-separated in that field) */  
5  public class PrintCommandLineArguments {  
6  /**  
7  * The main routine  
8  */  
9  public static void main(String[] args) {  
10     // Print the command line arguments of the program (no longer ignored ^_^)  
11     final void main(String[] args) {  
12     println("There were " + //NON-NLS-15  
13     length + // the number of command line arguments  
14     " command line arguments."); //NON-NLS-15  
15     for (String arg : args) { // iterate over the command line arguments  
16         println(arg); // print the current iteration element  
17     }  
18 }  
19 }  
20 }
```

How to Specify Command Line Args in Eclipse



The screenshot shows the Eclipse IDE's 'Run Configurations' dialog box. The title bar reads 'Run Configurations'. Below the title bar, the text 'Create, manage, and run configurations' is displayed, followed by 'Run a Java application' and a green play button icon. The main area is divided into two panes. The left pane contains a list of configuration types: 'Gradle Project', 'Java Applet', 'Java Application' (highlighted in orange), 'JUnit', 'Maven Build', and 'Task Context Test'. Above this list is a search filter field with a magnifying glass icon and a 'CE' button. Below the list, it says 'Filter matched 6 of 56 items'. The right pane is titled 'Configure launch settings from this dialog:' and contains a list of instructions: '- Press the 'New' button to create a configuration of the selected type.', '- Press the 'Duplicate' button to copy the selected configuration.', '- Press the 'Delete' button to remove the selected configuration.', '- Press the 'Filter' button to configure filtering options.', and '- Edit or view an existing configuration by selecting it.'. Below this list, it says 'Configure launch perspective settings from the 'Perspectives' preference page.'. At the bottom of the dialog, there are two buttons: 'Close' and 'Run'. A help icon (?) is located in the bottom left corner.

How to Specify Command Line Args in Eclipse



The screenshot shows the Eclipse IDE's "Run Configurations" dialog. The title bar reads "Run Configurations". Below the title bar, it says "Create, manage, and run configurations" and "Run a Java application" with a green play button icon.

The left sidebar contains a tree view of configuration types: "Gradle Project", "Java Applet", "Java Application" (expanded), "PrintCommandLineArguments" (selected), "JUnit", "Maven Build", and "Task Context Test". A search filter "type filter text" is at the top of the sidebar. At the bottom of the sidebar, it says "Filter matched 7 of 57 items".

The main area is titled "Name: PrintCommandLineArguments". It has several tabs: "Main" (selected), "Arguments", "JRE", "Classpath", "Source", "Environment", and "Common".

Under the "Main" tab, there are three sections:

- Program arguments:** A text area containing "Hello World!". To the right is a "Variables..." button.
- VM arguments:** An empty text area. To the right is a "Variables..." button.
- Working directory:** Two radio buttons: "Default:" (selected) and "Other:". The "Default:" option has a text field containing "\${workspace_loc:10_command_line_args}". Below these are three buttons: "Workspace...", "File System...", and "Variables...".

At the bottom right of the dialog are four buttons: "Revert", "Apply", "Close", and "Run".

How to Specify Command Line Args in Eclipse

A screenshot of the Eclipse IDE interface. The Package Explorer on the left shows a project named '10_command_line_args' with a source folder 'src' containing a package '(default package)' and a class 'PrintCommandLineArguments'. The main editor window displays the source code for 'PrintCommandLineArguments.java'. The code includes a comment explaining the purpose, a class declaration, and a main method that prints the number of command-line arguments and then iterates over them to print each one. The console at the bottom shows the output of running the program: 'There were 2 command line arguments.', 'Hello', and 'World!'.

```
1 /** Example for printing the command line arguments<br/>
2  * Eclipse: "Run As" -> "Run Configurations..." -> Java Application
3  *   -> new -> (x)= arguments -> program arguments
4  *   (type arguments space-separated in that field) */
5  public class PrintCommandLineArguments {
6  /**
7   * The main routine
8   *
9   * @param args
10  *   the command line arguments of the program (no longer ignored ^_^)
11  */
12  public static final void main(String[] args) {
13      System.err.println("There were " + //$NON-NLS-1$
14          args.length + // the number of command line arguments
15          " command line arguments."); //$NON-NLS-1$
16
17      for (String arg : args) { // iterate over the command line arguments
18          System.out.println(arg);// print the current iteration element
19      }
20  }
21  }
22  }
```

<terminated> PrintCommandLineArguments [Java Application] /usr/lib/jvm/java-8-openjdk-amd64/bin/java
There were 2 command line arguments.
Hello
World!

- We have learned what command line arguments are: A second way to pass data to a program (the first is `stdin`)
- We have learned how to receive them in a Java program: via `String[]` parameter of `main` method
- We have seen how to specify them in Eclipse if we want to run a program

谢谢

Thank you

Thomas Weise [汤卫思]
tweise@hfu.edu.cn
<http://iao.hfu.edu.cn>

Hefei University, South Campus 2
Institute of Applied Optimization
Shushan District, Hefei, Anhui,
China



Caspar David Friedrich, "Der Wanderer über dem Nebelmeer", 1818
http://en.wikipedia.org/wiki/Wanderer_above_the_Sea_of_Fog