CMSC 131

Object-Oriented Programming I

Command Line Arguments, For-Each Loops

Dept of Computer Science
University of Maryland College Park

This material is based on material provided by Ben Bederson, Bonnie Dorr, Fawzi Emad, David Mount, Jan Plane

Overview

- Command Line Arguments
- For-each loops

Command-Line Arguments

- **Dissecting main**: Recall the main method declaration:

  public static void main(String[ ] args) ...

  String[ ] args: is called with an array of String command-line arguments. What are these arguments?

- **Command-Line Arguments**: On Unix- and DOS-style systems, when a program is run from the command prompt, options are included on the command line. For example, the Unix command:

  % emacs fooBar.java

- runs the **emacs** program on the file **fooBar.java**. The string "**fooBar.java**" is a command-line argument to the program

Command-Line Arguments

- **Command-Line Arguments**: Provide a way for the user running your program to pass in run-time information
  - **Run-time options**: affect how the program runs (e.g., run the program in "debug mode" with additional diagnostic output.)
  - **I/O file names**: provide the names of files used for input and/or output
  - **Special definitions**: define special values (e.g., paper size for a word processor)
- **Java Command Arguments**:

  If you run Java directly from the **command prompt**, these arguments are typed right after the name of your Java program:

  javac CommandArgTest.java **(compile your program)**

  java CommandArgTest –f foo –b bar **(run it)**

  Here "**–f**", "**foo**", "**–b**", "**bar**" are the (4) command-line arguments

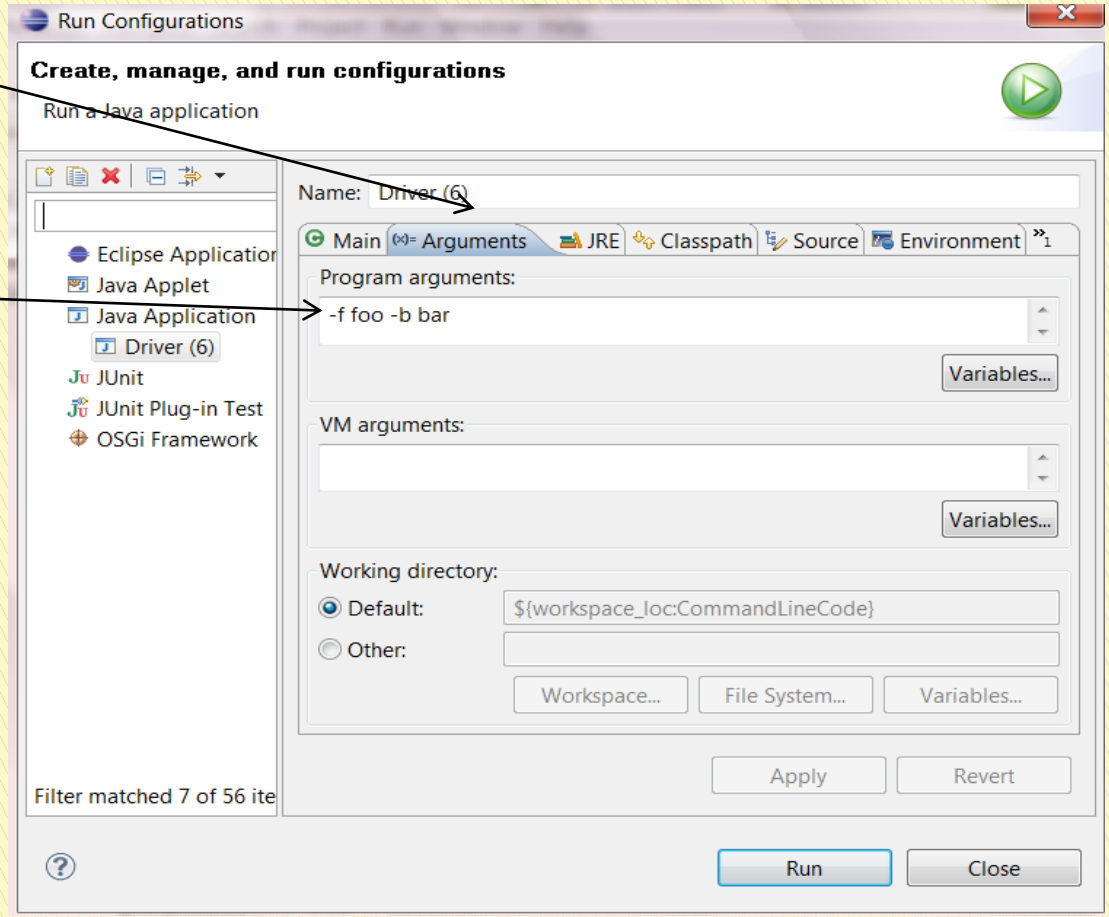
Command-Line Arguments: Eclipse

- **Java Command Arguments**:

  If you run Java from **Eclipse**, these arguments can be specified when you select "**Run Configurations…**" (rather than "**Run As**")

Select the "(x)= Arguments" tab

Enter your command-line arguments here



Command-Line Arguments: Example

File: CommandArgTest.java

```
public class CommandArgTest {
    public static void main(String[ ] args) {
       System.out.println( "Command line arguments:" );
       for ( int i = 0; i < args.length; i++)
          System.out.println( "  Argument[" + i + "] = " + args[i] );
    }
}
```

Output of:
java CommandArgTest –f foo –b bar

```
Command line arguments:
    Argument[0] = -f
    Argument[1] = foo
    Argument[2] = -b
    Argument[3] = bar
```

For-Each Loops

- Enhanced for loop
- Works for arrays and any class implementing the **Iterable** interface (e.g., **ArrayList**)
  - Behind the scenes there is an **iterator** object
- Iterating over **array**

```
String[ ] roster = {“Laura", “Mary", “Mike", “John"};
for (String student : roster)
    System.out.println(student);
```

For-Each Loops

- Iterating over **ArrayList**

```
ArrayList<String> roster = new ArrayList<String>( );
roster.add("Laura");
roster.add("Mike");

for (String student : roster)
    System.out.println(student);
```

For-Each Loops

- Limitations:
  - Modifications limited
    - Can't add items to the list being iterated over
    - Can't remove items from the list being iterated over
    - Can't replace items in the list being iterated over
  - Access only one
    - Only a single collection can be traversed at a time
    - Can't access the one before or the one after on this iteration
  - Limited to forward and one at a time
    - Can't traverse the list in the reverse order
    - Can't go to every other element or any variation