

Lecture Set #17: Command Line Running Issues & Comments

1. Command-line Java
2. Arguments to Main Method
3. Commenting Review



CMSC 131 Fall 2008
Jan Plane (adapted from Bonnie Dorr)

Command-Line Java

- Source File:
 - an ASCII text file named appropriately for the class it contains (i.e. `ClassDemo.java`)
 - created by an editor (emacs, vi, edit, notepad...)
- Translate to byte code:
 - `javac ClassDemo.java`
 - creates a file named `ClassDemo.class`
- Run (and interpret the Byte Code):
 - `java ClassDemo`
 - runs the java application defined in the the `ClassDemo` file
- Finding Things
 - For the System to be able to find `javac` and `java`
 - Command Search Path must be set it is named `PATH`
 - For the System to be able to find the class named as the argument to the `java` command
 - The Class Search Path must be set it is named `CLASS PATH`
 - Both of these are set in Windows using the Control Panel/System/Advanced/Environment Variables
- In Class Demonstration



CMSC 131 Fall 2008
Jan Plane (adapted from Bonnie Dorr)

1



Arguments to main

- Recall prototype of main method

```
public static void main (String[] args);
```

 - args is array of Strings
 - args come from operating system
 - When user runs executable ...
 - ... s/he can provide arguments
- Demonstration from Command Line
 - simple arguments
 - treating arguments as integers
- Demonstration from within Eclipse



Types of Documentation

- Two kinds of code commentary in Java
 - Implementation comments
 - Interface comments
- JavaDoc
 - from inside Eclipse
 - from outside of Eclipse
- Common Errors in Commenting
 - Too many comments:** This can obscure the flow of your program.
 - Too few comments:** Your intent may not be understood (code is never “self-documenting”!).
 - Comments that repeat the code:**

```
int total = 0; // initialize integer total to 0
```
 - Using comments to conceal unclear code:** ...just rewrite the code

```
double a = h*w; // set the area (a) to the height (h)
// times width (w)
double area = height * width;
```
 - Uninformative comments:** “What the heck does that mean?”

```
double d = processValue(); // Change later (legacy)
```
 - Misleading / erroneous comment:** These are dangerous

```
for ( int i = 0; i < a.length-1; i++) // run through the
//whole array
```



Javadoc Documentation

- **Class comments:** Immediately prior to each public class, add a javadoc comment that **explains what the class does**. You can also add the following special “**tags**”, which javadoc recognizes and provides special formatting for:

@author – the author of the class
@version – the current software version number
@see – refer the reader to related classes

- **Example:** In Rational.java

Sample javadoc class comment

```
/**  
 * This class implements a rational number object,  
 * and provides methods for performing arithmetic  
 * on rational numbers.  
 * @see java.lang.Math  
 * @author Schultzie von Wienerschnitzel III  
 * @version 3.14159  
 */  
public class Rational { ... }
```

CMSC 131 Fall 2008
Jan Plane (adapted from Bonnie Dorr)

4



Javadoc Documentation

- **Method comments:** Immediately prior to each public method, add a javadoc comment **explaining what the method does**, the meanings of the **parameters**, the **return value**, and any **errors**. The following tags are recognized:

@param – give the name and description of each parameter. There should be one for each parameter.
@return – describe the return value (unless it is void)
@throws – (Later: we will discuss error exceptions later this semester)
@deprecated – (Usually for system use: indicates that a method should be avoided, since better alternatives exist)

- **Example:**

```
/**  
 * Multiplies two rational numbers and returns their the product.  
 * @param q The first operand.  
 * @param r The second operand.  
 * @return A reference to a newly created Rational with the sum.  
 */  
public static Rational multiply( Rational q, Rational r) { ... }
```

CMSC 131 Fall 2008
Jan Plane (adapted from Bonnie Dorr)

5