Organised documentation and tutorials
Comments in the code itself
HOWTO.txt
A StackOverflow Q&A post
Forum post 5yrs ago "nvm fixed it"

CS 2112 Lab 2: Javadoc and I/O

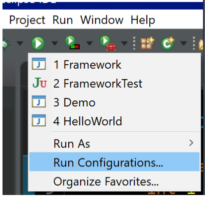
September 9 / 11, 2019

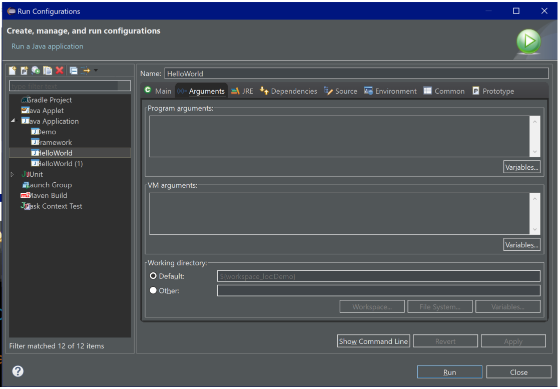
Importing A2

- Create a new Eclipse project
- Navigate to your Eclipse workspace on your hard drive
- Go into the project folder, and then into the src folder
- Copy the contents of the src folder from the release code into the src folder of your project
- From the package explorer, right click and choose Refresh

Run Configurations

You can pass arguments into project by clicking the down arrow next to the Run button and choosing Run Config.
Then, in the dialog, enter the arguments into the "Program Arguments" field.




Run Configurations

Create, manage, and run configurations

Run a Java application

Name: HelloWorld

Program arguments:

VM arguments:

Working directory:

Default: ${workspace_loc:Demo}

Other:

Filter matched 12 of 12 items

Javadoc Overview

- Javadoc is a tool for creating html documentation
- The documentation is generated from comments
- It produces actual html web pages
- Helps keep documentation consistent with the code

Doc Comments

- Doc comments start with /** and end with */.
- The /** and */ should be on their own lines
- Additional lines need to start with *
- Comments can have html tags

```
    /**
     * This is a javadoc comment
     */
```

Writing Good Docs

- First sentence: high level overview (no fluff)
  - Bad: This method is a method that computes the square root of a number
  - Good: Computes the square root
- Go into detail if necessary after first sentence
- Don't repeat things in the method declaration
  - Bad: The first argument is an integer
- Cover Edge Cases!

Javadoc Tags

- Use tags to help Javadoc parse your comments
- Tags start with a @ and are case sensitive
- It must be at the beginning of the line

```java
/**
 * Prints the kth element of the list
 *
 * @author Alexander Lee
 * @param list The list whose element is to be printed
 * @param k    The index of the item to be printed
 */
 public void printK(List list, int k) {

 }
```

Important Tags

Some important tags

- @author: describes the author
- @param: describes a specific parameter
- @return: describes the return value
- @throws: describes exceptions it throws
- @deprecated: describes the reason for deprecation
- @see package.class#member

Javadoc Links

- @link package.class#member label
- This tag inserts a link that points to the documentation of the specified class
- label represents the text that shows up
- The curly brackets indicate that it is an inline tag
- Can be placed anywhere in the comments, doesn't have to be at the beginning

```
/**
 * Get the names of an object
 *
 * @deprecated This function should not be used
 * Use {@link #getFirstName()} and
 * {@link al91.Person#getLastName()} instead.
 */
```

Javadoc and Eclipse

- To generate the doc, `Project > Generate Javadoc...`
- Eclipse automatically generates Javadoc comments if the method signature is already written
- Can configure Eclipse to complain about missing Javadoc comments: `Window (or Eclipse) > Preferences... > Java > Compiler > Javadoc`

I/O

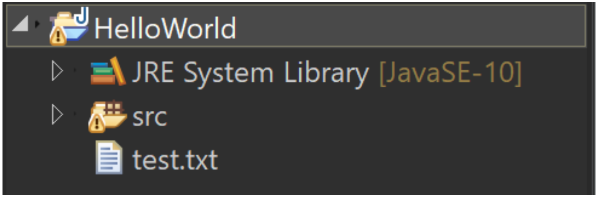
Everything on the following slides can be referenced later on the I/O handout:

https://courses.cs.cornell.edu/cs2112/2019fa/handouts/IO.pdf

File System

In Eclipse, by default, file paths are relative to the project directory (ie: outside the src/ folder).

```java
try {
        InputStream in =
            new FileInputStream("test.txt");
} catch (FileNotFoundException e) {
        e.printStackTrace();
}
```



Streams vs. Readers

- InputStream and OutputStream reads data one byte at a time.
- Reader and Writer reads data one character at a time.
- Remember to close your resource after using it.

Try making a text file with some text inside your project directory, then reading from it with a FileInputStream.

Try switching the FileInputStream for a Reader.

```
/* TODO Solution code
   Exercise left to the reader */
```

Streams vs. Readers

```java
try {
    FileReader in = new FileReader("document.txt");
    char[] array = new char[1];
    in.read(array);
    System.out.println(array[0]);
    in.close();
} catch (Exception e) {
    e.printStackTrace();
}

try {
    FileInputStream in =
        new FileInputStream("document.txt");
    byte[] array = new byte[1];
    in.read(array);
    System.out.println(array[0]);
    in.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

Character Encoding

Character encoding defines how characters we recognize get stored to disk as individual bytes.

For this class, use Unicode UTF-8.

Exercise

Find out what words are shared by two files, and return the number of unique words in common. Output the words you find to a different file. Write the Javadoc for the method before you begin to write any code. Remember to explain all the edge cases!

```java
long wordsInCommon(File file1, File file2) {
    // TODO implement
}
```