

CS 235 - Homework 2

Deadline:

Problem 1 -- completing the CS 235 Required Syllabus Confirmation and Reading Questions on Moodle -- is due by **11:59 pm on Friday, September 4, 2015.**

The rest of the problems are due by **11:59 pm on Wednesday, September 9, 2015.**

How to submit:

For Problem 1, complete the CS 235 Required Syllabus Confirmation and Reading Questions on the course Moodle site.

For the rest of the problems, submit your files using `~st10/235submit` on nrs-projects, with a homework number of 2, by the deadline shown above.

Reminder: Instructions for using the tool `~st10/235submit`:

- If you did not create your files (using, for example, `nano`) on nrs-projects, then first transfer your files to be submitted to a directory on nrs-projects.
 - On Linux or Mac OS X, you can use `sftp` (secure file transfer) to connect to `nrs-projects.humboldt.edu`, and then transfer them
 - In a Windows lab on campus, you can use the **WinSCP** program, which is a graphical `sftp` program.
 - The free program **FileZilla** is a graphical `sftp` program that works on Windows and Mac OS X (and maybe also Linux?)
- Once your files to be submitted are in a directory on nrs-projects, then use `ssh` (or PuTTY in a Windows lab on campus) to connect to `nrs-projects.humboldt.edu`.
- Use `cd` to change to the directory containing the files to be submitted -- for example,

```
cd 235hw02
```

- Type the command:

```
~st10/235submit
```

...and give the number of the homework being submitted (or whatever number you have been asked to do for lab-related files) when asked, and answer that, `y`, you do want to submit all of the files with of-interest-to-235 suffixes in the current directory. (Note that I don't mind if a few extraneous files get submitted as well -- I'd rather receive too many files than too few, and typing in all of the file names for some assignment is just too error-prone...)

- You are expected to carefully check the list of files that the tool believes have been submitted, and make sure all of the files you hoped to submit were indeed submitted! (The most common error is to try to run `~st10/235submit` while in a different directory than where your files are...)

Purpose

To make sure you have carefully read the course syllabus, and to practice some Java language features in command-line applications.

Important notes:

- Follow the initial class Java coding standards mentioned in class -- some of these include:
 - Follow the Java naming standards that have been discussed in class.
 - Attempt "javadoc-style" comments for **each** Java class and method, in the same style as you see in posted in-class examples.
 - Everything inside a set of { } must be indented by AT LEAST 3 SPACES -- and the beginnings of statements that are sequential should be indented the SAME NUMBER of spaces. (That is, sequential statements should line up.)
 - { and } should each go on their OWN line, with { lined up evenly with the preceding line, with the { }'s contents indented by at least 3 spaces, and with } lined up with the opening {. That is, handle the curly braces as you see in all posted class examples!
- ASK ME if any of these are unclear to you!

Problem 1 - 25 points - due by 11:59 pm FRIDAY SEPTEMBER 4!

Read over the course syllabus until you are comfortable saying that you have read and that you understand it. (Be sure to ask me any questions you have as you are reading it!) Note that, in addition to the paper copy of the syllabus that was passed out in class, an electronic version of the syllabus is also available from the public course web site.

Then, go to the course Moodle site. In the "Miscellany section", you will see a link to a questionnaire, "CS 235 Required Syllabus Confirmation and Reading Questions". Answer these questions in this questionnaire by **11:59 pm on Friday, September 4**.

Problem 2 - 15 points

Recall our discussion in class about Java applications: any `public` Java class that contains a method named `main` with the following method header:

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

...is a Java **application**. Recall that such an application can be run with the `java` command, followed by the name of the class (thus, with NO suffix) -- that is, for a class `TryMe` that contains such a `main` method, if you were to type:

```
java TryMe
```

...then `TryMe`'s `main` method would be executed. (Do you see how this is a little like the `main` function in a C++ program, where execution "starts" when that C++ executable program is run?)

And recall that `args`, `main`'s parameter, is an array of `String` instances -- the command line

arguments given after the class name when the application is executed. That is, if one were to execute:

```
java TryMe moo 3 "and how"
```

...then, for that call of `main`,

`args[0]` would be `"moo"`

`args[1]` would be `"3"` (and yes, I DO mean the `String` `"3"`, not the integer 3)

`args[2]` would be `"and how"`

and `args.length` would be 3, since you can find the number of elements in ANY Java array via its `length` data field.

Finally, note that the example application `TryMe.java` from the Week 2 Lab, that prints its command-line arguments to standard output, is posted on the public course web site, under "In-class examples", and is also posted along with this homework handout.

As a warm-up, modify `TryMe.java` so that it **also**:

- first prints your name to the screen within a message of your choice
- next prints a message to the screen noting how many command line arguments were given in this call
- (then prints those arguments to the screen, one per line)
- finally prints a concluding message of your choice to the screen *after* it outputs the command-line arguments
- Make sure you add a second `@author` comment, and modify the `@version` date, also, in `TryMe`'s opening javadoc comment.
- (NOTE -- we played around with both `for`-loop versions in the posted version of `TryMe.java` -- feel free to edit out the commented version you do not choose to use!)

Submit your resulting `TryMe.java`.

Problem 3 - 20 points

The purpose of this is to practice writing some Java logic, as well as to work with `String` instances a little bit.

Write a command-line Java application class `Contradict` which behaves as follows:

- if NO command-line arguments have been given, it should complain in a message to standard output that at least one is needed and exit.
- if the first command-line argument is `"yes"`, it should print NO (in the case of your choice) to standard output.
 - OPTIONAL variation: have it print NO for a first command-line argument of `"yes"` written in ANY case -- for example, `"Yes"` or `"YES"` or `"yEs"` or... etc.
- if the first command-line argument is `"no"`, it should print YES (in the case of your choice) to standard output.

- OPTIONAL variation: have it print YES for a first command-line argument of "no" written in ANY case -- for example, "No" or "NO" or "nO" or... etc.
- if the first command-line argument is anything else, it should print PERHAPS (in the case of your choice) to standard output.
- it may happily ignore any other command-line arguments given
- Hint: to make this easier, look at the methods for class `String` in package `java.lang` in the Java 8 API.

For example:

```
java Contradict
```

could cause this to be printed to standard output:

```
Contradict requires at least one argument -- Bye!
```

And, as another example:

```
java Contradict yes
```

could cause something like this to be printed to standard output:

```
NO
```

And, as another example:

```
java Contradict no
```

could cause something like this to be printed to standard output:

```
YES
```

And, as another example:

```
java Contradict moo
```

could cause something like this to be printed to standard output:

```
PERHAPS
```

Submit your resulting `Contradict.java`.

Problem 4 - 20 points

Make a copy of your `GameDie` class from Homework 1 in the same directory as this homework's Java files. Then, any class within this directory can declare and use `GameDie` instances. (Consider `GameDieTest.java`, which did this!)

Write a command-line Java application `RollForThem` which can be called with any number of command-line arguments (even 0).

- If none are given, it simply prints a message to standard output including that there are no arguments to roll for.
- Otherwise, it creates a game die instance with a **named-constant** number of sides of your choice,
 - and it rolls this game die once for each command-line argument,

- printing to standard output a message including both the command-line argument and the roll for that command line argument.

For example:

```
java RollForThem
```

could cause this to be printed to standard output:

```
Hmm, there are NO arguments to roll for!
```

And, as another example:

```
java RollForThem Mona Ed Lisa
```

could cause something like this to be printed to standard output:

```
for Mona: rolled 3  
for Ed: rolled 1  
for Lisa: rolled 6
```

Submit your resulting `RollForThem.java`.

Problem 5 - 20 points

You read about `String` methods in the Java API for Homework 1 - Problem 3. Decide on at least one `String` method BESIDES `equals` that you would like to "play" with.

Then, for some more `String` method practice and for a little lightweight `Scanner` practice, write a command-line Java application `StringPlay` that:

- expects NO command-line arguments (and cheerfully ignores any it is given)
- ASKS the user to enter at least one string of their choice, and reads in what they enter
- calls your chosen `String` method (or methods) on the (or each) entered string, printing the result (or each result) to standard output in a suitably descriptive message.

You need to do this for at least one `String` method for at least one user-entered-when-prompted string -- you MAY do this for MORE user-entered-when-prompted strings and MORE `String` methods IF you would like!

Submit your resulting `StringPlay.java`.