

CS 235 - Homework 1

Deadline:

Due by **11:59 pm** on **WEDNESDAY, September 2, 2015**.

How to submit:

Submit your files using `~st10/235submit` on nrs-projects, with a homework number of 1, 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 235hw01
```

- 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 start practicing some simple Java programming, to start learning to use the Java API documentation, and to lightly reflect on Chapter 1 of the "Core Java" text.

Important notes:

Here is the initial set of this class's Java style standards (ASK ME if any are unclear to you!):

- 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!

Problem 1

- You will find the source code for the Week 1 Lab versions of the classes `GameDie.java` and `GameDieTest.java` on the CS 235 public course web page, both posted along with this homework handout, and under "In-class Examples", in the Week 1 Lab section. Create copies of these classes that you will modify for this homework.
 - Include your name in a third `@author` line in each of their opening comments, and
 - change the `@version` comment to have today's date.
- Compile and correct as necessary until your classes successfully compile, and `GameDieTest` successfully runs.
- In your `GameDieTest.java`:
 - declare a third `GameDie` instance with a different number of sides from those already declared,
 - and also print to the screen the result of rolling that new `GameDie` instance at least once (including printing the name of the die being rolled, as was done for the original two dice).
- Again compile and correct as necessary until your classes successfully compile, and `GameDieTest` successfully runs.
- At this point, demonstrate that your `GameDieTest` was working by running the following on nrs-projects:

```
java GameDieTest > hw1-1-demo.txt
```

 - (the `>` above is called output redirection -- it causes whatever would be printed to the screen to instead be printed in a file whose name is given after the `>`)
 - (would you like to see the contents of that file? The `more` command lets you view the contents of a file one screen at a time: `more hw1-1-demo.txt`)
- Submit your resulting files `hw1-1-demo.txt`, `GameDie.java`, and `GameDieTest.java`.

Problem 2

- Consider how the methods and data fields are declared in the `GameDie` example.
- We decide we'd like a `top` data field, storing what is currently on top of a `GameDie` instance -- this is either the value of the latest roll, OR if the die has never been rolled, we are arbitrarily deciding that it will start out with 1 on top.
 - So -- in `GameDie.java`, declare this private data field appropriately,
 - initialize it properly in `GameDie`'s constructor,
 - modify the `roll` method so it resets `top` each time a roll occurs, and
 - add an accessor method `getTop` that returns the current top of the calling `GameDie`.
- Compile and correct as necessary until your modified class successfully compiles.
- In your `GameDieTest.java`:
 - print to the screen the result of calling your new `getTop` method at least twice:
 - ...once before one of the `GameDie` instances has been rolled,
 - ...and once immediately after one of the `GameDie` instances has been rolled.
 - (you hope to see it return 1 for the not-yet-rolled die, and whatever was just rolled for the just-rolled die)
 - (do you want to do this for all three of your `GameDie` instances? Or after each roll of cube? That's fine, if you would like!)
- Again compile and correct as necessary until your classes successfully compile, and `GameDieTest` successfully runs.
- At this point, demonstrate that your `GameDieTest` was working by running the following on nrs-projects:

```
java GameDieTest > hw1-2-demo.txt
```
- Submit your resulting files `hw1-2-demo.txt`, `GameDie.java`, and `GameDieTest.java`.

Problem 3

- From time to time during the semester, you will be given problems whose purpose is to encourage you to explore the Java Abstract Programming Interface (API) documentation.
 - There is a link to Java 8 API on the public course web site, in the **References** section.
- Here, I want you to look up the `String` class in the package `java.lang`, and skim through its available methods.
- Then, in a file name `hw1-3.txt` (a plain text file), put your name, and give the names of at least 3 `String` methods that interest you. After **each** of these 3 `String` methods:
 - explain why it interests you
- Note that your answers may be posted anonymously to the course Moodle site.
- Submit your resulting file `hw1-3.txt`

Problem 4

- Consider Chapter 1 in Core Java. Decide on one of the 11 Java "buzzwords" discussed that you find interesting. Then, in a file name `hw1-4.txt` (a plain text file), put your name, and in the form of a **paragraph** of at least 3 sentences, (NOT as a bulleted list), include:
 - the buzzword you chose
 - why you chose it
 - paraphrase why it is claimed that Java can fairly be described by that buzzword
 - explain either why you think that buzzword really does apply to Java, or why you think that it really does not.
- Note that your answer may be posted anonymously to the course Moodle site.
- Submit your resulting file `hw1-4.txt`