CS 235 - Homework 2

Deadline:

11:59 pm on **Friday, September 10, 2021**.

Purpose

To check your understanding of some more Java basics, and to practice some Java language features in command-line applications.

How to submit:

You complete **Problem 1** on the course Canvas site (short-answer questions related to more Java basics).

For **Problems 2 onward**, you will create the specified `.java` files, and then submit those to the course Canvas site. (You'll be creating `.class` files, also, but you do not submit those.)

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 - 12 points

Problem 1 is correctly answering the "HW 2 - Problem 1 - short-answer questions on more Java basics" on the course Canvas site.

Problem 2 - 18 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.

As a warm-up, create a public application class `TryMe` in file `TryMe.java` so that it **at least**:

- 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
- (it can do anything else that you like after this, between the above and the following:)
- finally prints a concluding message of your choice to the screen *after* it outputs the command-line arguments
- Make sure you include `@author` and `@version` parts in `TryMe`'s opening javadoc comment.

Submit your resulting `TryMe.java` .

Problem 3 - 23 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` that 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
  - BUT -- as an OPTIONAL variation -- it *may* do something with any additional arguments, IF you would like.
- Hint: to make this easier, search for "Java 16 String class" on the Web and read over the available methods for class `String` in Java JDK 16.

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 - 24 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 - 23 points

Important Aside #1 - `import java.util.*;`

As we found out in the Week 2 Lab Exercise, classes in Java package `java.util` (such as `Scanner`) are NOT automatically visible to all Java classes.

- So (as we'll discuss further in class) you can make them available to a class by importing them, adding this line as the first line of a Java source code file:

  ```
  import java.util.*;
  ```

Important Aside #2 - awkwardness when call `Scanner` method `nextLine` after something like `nextInt`

It appears that the Java `Scanner` class's `nextLine` method suffers from a similar issue as the C++ `getline` function...!

- That is, in C++, if you do interactive input using `cin >>` and then want to use `getline` to read in an entered line of input as a string, you need to add an additional `getline` to "get past" the previous `cin >>`'s enter/newline.
- It appears that, in Java, using a `Scanner` object with `System.in`, if you use something like one of

the `Scanner` methods `nextInt` or `nextDouble` and then want to use method `nextLine` to read in an entered line of input as a string, you likewise need to add an additional call of method `nextLine` to "get past" the previous entry's enter/newline!

Now, as in C++, if you don't HAVE any calls to methods such as `nextInt` or `nextDouble` before calls to `nextLine`, there is no such issue, and the interactive input of lines of input goes like you'd expect!

Adding an extra call to `Scanner` method `getline` *should* solve the issue we ran into in the Week 2 Lab Exercise.

And, such an extra call should *not* be necessary on this problem, since this method will only use the `Scanner` class method `nextLine`.

your task for Problem 5:

Search for "Java 16 String class" on the Web and read over the available methods for class `String` in Java JDK 16.

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` .