Project 0: Extending A Class

In this project you will write a `Student` class that subclasses the `Human` class we discussed in lecture.

Goals:

- Work with someone else's class
- Learn about inheritance and virtual methods
- Read program arguments from the command line

Below is the skeleton of a simple Java class that extends the `Human` class. Class `Student` is declared in package `edu.pdx.cs.joy.`*your-login-id*. In order for `Student` to access its superclass, `Human`, it must `import` the `edu.pdx.cs.joy.lang` package. Note how the `Student` class *overrides* the `says` and `toString` methods.

```
package edu.pdx.cs.joy.<your-login-id>;

import edu.pdx.cs.joy.lang.*;    // Lets us use Human
import java.util.*;              // Lets us use ArrayList

/**
 * This class is represents a <code>Student</code>.
 */
public class Student extends Human {

  /**
   * Creates a new <code>Student</code>
   *
   * @param name
   *        The student's name
   * @param classes
   *        The names of the classes the student is taking.  A student
   *        may take zero or more classes.
   * @param gpa
   *        The student's grade point average
   * @param gender
   *        The student's gender ("male", "female", or "other", case insensitive)
   */
  public Student(String name, ArrayList<String> classes, double gpa, String gender) {
    super(name);
  }
  /**
   * All students say "This class is too much work"
   */
  public String says() {
    throw new UnsupportedOperationException("Not implemented yet");
  }

  /**
   * Returns a <code>String</code> that describes this
   * <code>Student</code>.
   */
  public String toString() {
    throw new UnsupportedOperationException("Not implemented yet");
  }

  /**
   * Main program that parses the command line, creates a
   * <code>Student</code>, and prints a description of the student to
   * standard out by invoking its <code>toString</code> method.
   */
  public static void main(String[] args) {
    System.err.println("Missing command line arguments");
    System.exit(1);
  }
}
```

The easiest way to get started with this project is to following the instructions in this GitHub repository.

```
    https://github.com/DavidWhitlock/PortlandStateJavaGettingStarted
```

The generated Maven project contains unit and integration tests to get you started with test-driven development. You can build the project and run its tests by invoking the `verify` phase.

```
$ ./mvnw verify
```

Building the Maven project will create an "executable jar" file that will invoke the `main` method the `Student` class.

```
$ java -jar target/student-1.0.0.jar
```

Initially, running the executable jar will simply print "Missing command line arguments" to standard error. The goal of this assignment is to parse the input from the command line, create a new `Student` object, and invoke its `toString` method such that running this command result in the output below:

```
$ java -jar target/student-1.0.0.jar \
  Dave male 3.64 Algorithms "Operating Systems" Java
Dave has a GPA of 3.64 and is taking 3 classes: Algorithms, Operating
Systems, and Java.  He says "This class is too much work".
```

Error handling: Your program should exit "gracefully"[1] with a user-friendly error message under all reasonable error conditions. For instance, if the command line does not contain enough entries, then your program should issue an error like `Missing command line arguments!` It would be even better if it stated which arguments were missing.

A note from Dave on gender and this project

At one point in time, this project modeled gender as a binary choice between "female" and "male". A student of mine, Casper Rutz, reached out to engage me in a conversation about the impacts of a representing gender in the computing sciences. They had this to say on the subject:

> Confining options of things like gender, sexuality, and race to boxes to be checked or unchecked is problematic as it automatically limits the autonomy of our users, but is often necessary for the sake of user statistics and database entry. The goal we should have as programmers is to allow our users as much freedom to choose how to define themselves in our system as we realistically can, given the constraints of whatever we are building.
>
> In a real student record system (such as the one used by PSU) there are many gender options for students to choose from; however, for ease of programming, this assignment is going to limit gender and pronouns to three options: male (he/him), female (she/her), and other (they/them).

In our dialog, Casper helped me realize something that I think is powerful: When a project like this one restricts gender to a binary choice, people who do not identify within that binary cannot use themselves as a test case. As a result, their identity is not represented and, in a way, is erased. That runs counter to a software engineer's goal of writing software that is welcoming and usable to a wide customer base.

Casper and I talked about how best to evolve this project to be more inclusive (and realistic). Modeling a multi-valued data attribute is part of the project and something that I want students to learn how to do. We agreed that it would be appropriate to include a third gender option of "other" to model a `Student` who does not identify as "male" or "female".

I admire the courage that Casper showed to bring these issues to my attention and greatly appreciate the education they have brought to me. That education has helped me improve this project.

[1]For example, the user shouldn't see any evidence of an exception being thrown.