CS410J 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.cs410J.`*your-login-id*. In order for `Student` to access its superclass, `Human`, it must `import` the `edu.pdx.cs410J.lang` package. Note how the `Student` class *overrides* the `says` and `toString` methods.

```
package edu.pdx.cs410J.<your-login-id>;

import edu.pdx.cs410J.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" or "female", 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);
  }
}
```

To get you started with the project, there is a Maven archetype for this Student project.

```
$ mvn archetype:generate \
  -DarchetypeCatalog=https://dl.bintray.com/davidwhitlock/maven/ \
  -DarchetypeGroupId=edu.pdx.cs410J \
  -DarchetypeArtifactId=student-archetype
Define value for groupId: : edu.pdx.cs410J.<login-id>
Define value for artifactId: : student
Define value for version:  1.0-SNAPSHOT: :
Define value for package:  edu.pdx.cs410J.<login-id>: :
Confirm properties configuration:
groupId: edu.pdx.cs410J.<login-id>
artifactId: student
version: 1.0-SNAPSHOT
package: edu.pdx.cs410J.<login-id>
 Y: : Y
```

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.

```
$ mvn verify
```

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

```
$ mvn package
$ java -jar target/student-1.0-SNAPSHOT.jar \
  Dave male 3.64 Algorithms "Operating Systems" Java
```

Running with the above command line arguments should result in the following output:

```
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.

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