Homework 10 - Enterprise

Authors: Andrew Chafos, Jack Kelly, Vince Li, Davis Williams, Hussain Miyaziwala, Shishir Bhat

Problem Description

Since your bizarre time traveling incident from a few weeks ago, you've successfully launched your own forum site, helped the owner of a local puppy shelter modernize their business, explored the curious properties of Conway's Game of Life, and have created a dueling simulator for Professor Dumbledore at Hogwarts. Needless to say, things have been looking up for being trapped in 1998! However, much to your dismay your forum site has had very little activity; apparently no one else has experienced similar time traveling incidents.

However, today that is about to change! Instead of waking up in your bed this morning, you wake up in a futuristic cell, and in a much more comfortable bed. You are greeted by a bizarre humanoid creature with yellow-green, scaly skin. It approaches you, and says, "Good morning, human. I apologize for the previous mishap in our time travel device; it has been acting up due to recent events. My name is Silik, and I am the leader of the Suliban Cabal. You are currently hundreds of light-years away from Earth on my ship, and it is the Earth year 2745. Species around the universe are currently engaged in a Temporal Cold War, which has caused the timeline to become seriously disrupted. My men have taken heavy damage in an attempt to fix it, and we need your help to restore the timeline. We currently lack sufficient technical manpower to restore our database on Starfleet and the history of your species. Because of your great technical ability, you have been selected to create the infrastructure that will restore it. You will start with data representations of humans, androids, and Vulcans; we believe events related to these three species particularly disrupted the timeline. Once your work is complete, we will send you back to your own time so you can resume your studies, but only if you are successful. We have much Java documentation to guide your work, and we will provide an IDE that you are comfortable with. Remember, the timeline depends on your success, so we can't afford any mishaps. Good luck!"

Solution Description

For this assignment, you will be dealing with 3 concrete classes, an abstract class, an enumeration, and 2 interfaces.

The following files are provided to you for your reference. You will have to implement methods declared in them, but you will not have to modify any code in the files themselves:

- `Alien.java`
- `Logic.java`
- `Rank.java`

In these files, the following methods have been provided:

Alien.java

- This file contains the Alien *abstract class.*
- It contains a singular method, **`String getHomePlanet();`**
  - This method will need to be implemented in **Vulcan.java**, which extends it.

Logic.java

- This file contains the Logic *interface.*

- It contains a singular method, **`boolean isPrime(int num);`**
  - This method will need to be implemented in both **Vulcan.java** and **Android.java**.
  - The Vulcan species is well-known for being calculating and logical, rather than emotional and instinctive like say, humans or Andorians, which is why they implement this interface. Similarly, androids are part human, part machine, so internally they use much precise logic to function!
  - Since both Vulcans and androids are "calculating" and logical, it would make sense that they would need mathematical capabilities to perform their decisions.
  - See the file itself for implementation details.

Rank.java

- This file contains the Rank *enumeration* type.
- It contains a list of Rank enums to be used in the next 4 listed files. See details below for their specific use cases.

You will need to make changes to the following files, and you will be submitting them to Gradescope:

- `Officer.java`
- `Human.java`
- `Vulcan.java`
- `Android.java`

Officer.java

- This file contains the **Officer** *interface.*
- It contains 2 *abstract* methods, **`String getName();`** and **`Rank getRank();`**.
  - These methods will need to be implemented in **Vulcan.java**, **Android.java**, and **Human.java**. Since the Cabal's database only deals with members of these entities that are part of Starfleet, we know that all instances of them will be Starfleet officers, which is why this interface is used.
  - See the file itself for implementation details.
- Additionally, there is one method in this file that you have to implement: **`default String rankString();`**
  - See the file itself for implementation details.
- Lastly, there is an implemented static String method provided to you, **`private String capitalizeEachWord(String rank);`**, that will be used in the implementation of **`default String rankString();`**. No need to modify this method.

Human.java

- This file contains the **Human** *concrete class.*
- This class will implement the **Officer** interface. See **Officer.java** for its requirements.
  - For any method overridden from this interface make sure to use the **`@Override`** annotation directly above each method header.
- It will have 2 instance variables, **`String name`** and **`Rank rank`**
  - **`name`** should never be able to be changed once initialized in the constructor. What keyword would allow us to do this?
  - **`rank`** is allowed to be modified after initialization
- It should also have exactly one constructor, which takes in **`String name`** and **`int rank`**. It should assign **`String name`** accordingly.
  - However, we must get the Rank enum value corresponding to int rank. A `rank` of 0 corresponds to `Rank.ENSIGN`, `rank` of 1 to `Rank.LIEUTENANT`, and so on. Note that we can only obtain a valid rank if 0 <= `rank` <= 5.

    - So, if `int rank` is negative or greater than 5, initialize `Rank rank` to `null`. Otherwise, obtain the corresponding enum rank using `Rank.values()`, and set `Rank rank` equal to it.
- This class will also implement the overridden methods **`public String toString()`** and **`public boolean equals(Object other)`** from **`java.lang.Object`**
    - For any method overridden from **`java.lang.Object`** make sure to use the **`@Override`** annotation directly above each method header.
    - See the file for implementation details.

Vulcan.java

- This file contains the **Vulcan** *concrete class.*
- This class will implement the **Officer** interface. See **Officer.java** for its requirements.
    - For any method overridden from this interface make sure to use the **`@Override`** annotation directly above each method header.
- This class will also implement the **Logic** interface.
    - For any method overridden from this interface make sure to use the **`@Override`** annotation directly above each method header.
    - See **`Vulcan.java`** for implementation details.
- This class will also extend the **Alien** abstract class. See **Alien.java** for its requirements.
    - Note that the home planet of a Vulcan is Vulcan!
    - For any method overridden from this abstract class make sure to use the **`@Override`** annotation directly above each method header.
- It will have 2 instance variables, **`String name`** and **`Rank rank`**
    - **`name`** should never be able to be changed once initialized in the constructor. What keyword would allow us to do this?
    - **`rank`** is allowed to be modified after initialization
- It should also have exactly one constructor, which takes in **`String name`** and **`int rank`**. It should assign **`String name`** accordingly.
    - However, we must get the Rank enum value corresponding to int rank. A `rank` of 0 corresponds to `Rank.ENSIGN`, `rank` of 1 to `Rank.LIEUTENANT`, and so on. Note that we can only obtain a valid rank if 0 <= `rank` <= 5.
    - So, if `int rank` is negative or greater than 5, initialize `Rank rank` to `null`. Otherwise, obtain the corresponding enum rank using `Rank.values()`, and set `Rank rank` equal to it.
- You should create an additional overloaded method **`boolean isPrime(int num1, int num2);`**
    - This method overloads the method **`boolean isPrime(int num);`** from the **Logic** interface.
    - See **`Vulcan.java`** for its implementation details.
- This class will also implement the overridden methods **`public String toString()`** and **`public boolean equals(Object other)`** from **`java.lang.Object`**
    - For any method overridden from **`java.lang.Object`** make sure to use the **`@Override`** annotation directly above each method header.
    - See the file for implementation details.

Android.java

- This file contains the **Android** *concrete class.*
- This class will extend the **Human** concrete class. However, no methods will be overridden from that class.
- This class will also implement the **Logic** interface.
    - For any method overridden from this interface make sure to use the **`@Override`** annotation directly above each method header.
    - See **`Android.java`** for implementation details.
- This class will **not** have any instance variables! Instead, it will pass **`String name`** and **`int rank`** to the constructor of its superclass, **Human**

  - Implement a constructor that takes in **`String name`** and **`int rank`**. This constructor should call the superclass constructor in **Human**, which takes in the same parameters. This can be accomplishes using the **`super`** keyword.
  - It should have a second, no-arg constructor that uses constructor chaining with the `this` keyword to call the other constructor with parameters `"Data"` and `2`.
- You should create an additional overloaded method **`boolean isPrime(int num1, int num2, int num3);`**
  - This method overloads the method **`boolean isPrime(int num);`** from the **Logic** interface.
  - See **`Android.java`** for its implementation details.
- This class will also implement the overridden method **`public String toString()`**.
  - For any method overridden from **`java.lang.Object`** make sure to use the **`@Override`** annotation directly above each method header.
  - See the file for implementation details.

Important: Make sure each constructor that takes in `String name` and `int rank` takes them in exactly in that order!

Testing your code:

We have provided a driver class that you can use to test your code:

- `Enterprise.java`

You should create your own tests in the `main` method! Here is what the expected output of the `main` method we have provided is:

```
Captain Kirk
Captain Picard
Admiral Forrest
Android Lieutenant Commander Data
Android Lieutenant Commander Data
Commander Spock from Vulcan
Commander T'Pol from Vulcan
Captain Picard
Unranked Daniels

false
true
false
true

false
true
false
false
```

Rubric

- [12] `Officer.java`
  - [12] `String rankString()` implementation
- [25] `Human.java`
  - [5] Correct Constructor and Variables
    * [1] Correct placing of final keyword
    * [1] Correct variable access modifiers

        - [3] Correct Constructor behavior
    - [5] Implements `Officer` methods
        - [1] @Override annotation present for both
        - [4] Correct behavior for both
    - [5] toString() method
        - [1] @Override annotation present
        - [4] Correct behavior
    - [10] equals() method
        - [1] @Override annotation present
        - [2] Correct behavior
- [38] `Vulcan.java`
    - [5] Correct Constructor and Variables
        - [1] Correct placing of final keyword
        - [1] Correct variable access modifiers
        - [3] Correct Constructor behavior
    - [5] Implements `Officer` methods
        - [1] @Override annotation present for both
        - [4] Correct behavior for both
    - [3] Correctly implements `Alien` method
    - [6] Implements `Logic` method
        - [1] @Override annotation present
        - [1] Correct behavior
    - [4] Overloads `Logic` method
        - [4] Correct behavior
    - [5] toString() method
        - [1] @Override annotation present
        - [4] Correct behavior
    - [10] equals() method
        - [1] @Override annotation present
        - [9] Correct behavior
- [25] `Android.java`
    - [10] Constructors
        - [5] 2-argument Constructor
        - [2] Uses `super` keyword
        - [3] Correct behavior
        - [5] no-argument Constructor
        - [2] Uses `this` keyword
        - [3] Correct behavior
    - [6] Implements `Logic` method
        - [1] @Override annotation present
        - [3] Correct behavior
    - [4] Overloads `Logic` method
        - [4] Correct behavior
    - [5] toString() method
        - [1] @Override annotation present
        - [4] Correct behavior

Allowed Imports

- There are no allowed imports for this assignment.

Javadocs

For this assignment, you will be commenting your code with Javadocs. Javadocs are a clean and useful way to document your code's functionality. For more information on what Javadocs are and why they are awesome, the online overview for them is extremely detailed and helpful.

You can generate the javadocs for your code using the command below, which will put all the files into a folder called javadoc:

```
$ javadoc *.java -d javadoc
```

The relevant tags that you need to include are `@author`, `@version`, `@param`, and `@return`. Here is an example of a properly Javadoc'd class:

```
import java.util.Scanner;

/**
 * This class represents a Dog object.
* @author George P. Burdell
* @version 1.0
*/

public class Dog {

    /**
     * Creates an awesome dog (NOT a dawg!)
     */

    public Dog() {
    ...
    }

    /**
     * This method takes in two ints and returns their sum
     * @param a first number
     * @param b second number
     * @return sum of a and b
     */

    public int add(int a, int b) {
    ...
    }
}
```

A more thorough tutorial for Javadocs can be found here. Take note of a few things:

1. Javadocs are begun with `/**` and ended with `*/`.
2. Every class you write must be Javadoc'd and the `@author` and `@verion` tag included. The comments for a class should start with a brief description of the role of the class in your program.
3. Every non-private method you write must be Javadoc'd and the `@param` tag included for every method parameter. The format for an `@param` tag is `@param <name of parameter as written in method header> <description of parameter>`. If the method has a non-void return type, include the `@return` tag which should have a simple description of what the method returns, semantically.

Checkstyle can check for Javadocs using the -a flag, as described in the next section.

Checkstyle

For this assignment, we will be enforcing style guidelines with Checkstyle, a program we use to check Java style. Checkstyle can be downloaded here. To run Checkstyle, put the `jar` file in the same folder as your homework files and run

```
java -jar checkstyle-6.2.2.jar -a *.java
```

The Checkstyle cap for this assignment is **90 points**. This means that up to 90 points can be lost from Checkstyle errors.

Collaboration Statement

To ensure that you acknowledge collaboration and give credit where credit is due, **we require that you place a collaboration statement as a comment AT THE TOP of at least one java file that you submit**. That collaboration statement should say either:

I worked on the homework assignment alone, using only course materials.

or

In order to help learn course concepts, I worked on the homework with [give the names of the people you worked with], discussed homework topics and issues with [provide names of people], and/or consulted related material that can be found at [cite any other materials not provided as course materials for CS 1331 that assisted your learning].

Recall that comments are special lines in Java that begin with `//`.

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `Officer.java`
- `Human.java`
- `Vulcan.java`
- `Android.java`

Make sure you see the message stating "HW10 submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your last submission: be sure to **submit every file each time you resubmit**.

Gradescope Autograder

For each submission, you will be able to see the results of a few basic test cases on your code. Each test typically corresponds to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

1. Prevent upload mistakes (e.g. forgetting checkstyle, non-compiling code)
2. Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or checkstyle your code; you can do that locally on your machine.

Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.