Overview

This week, we are going to help you understand classes, objects, static methods, and the concepts of overriding and shadowing.

Classes and objects

In lecture, you've seen classes Person and Student, among others.

You can think of a class as describing a bunch of various related attributes plus some methods that do things to those attributes.

For example, a Person might have a name attribute and a changeName method – both of these things belong strictly to a Person and you couldn't conceive of a Person without these attributes.

Honorifics and titles

Mr., Mrs., Ms., and Mx. are all titles that are used by adults in Canadian culture. Some occupations come with honorifics that alter a person’s title. The current President of the United States is addressed as “Mister President”, whereas a member of the Canadian house of parliament might be called “The Right Honourable Sophia”.

If you wanted to have a getHonorific() method in class Person, you might implement it like this:

```java
public String getHonorific() {
    if (getOccupation().equals("Member of the House of Commons")) {
        return "The Right Honourable";
    } else if (getOccupation().equals("Lieutenant")
                && getCountry().equals("United States of America") {
        return "Lieutenant";
    } else {...}
}
```

You can see that such a method would quickly become hundreds of lines, and if any new titles appeared you would need to constantly modify your Person object – this is clearly unmaintainable.

It would be much better if each specific type of person reported their own title.

You can override methods to say “use my behavior instead of my superclass’ behavior”.

For example, a Lieutenant class might extend Person and override the getHonorific method to return "Lieutenant" regardless of the person’s title.

Overriding is as simple as declaring your own version of a method, with the same return type, name, and parameter types. Java calls your method instead of your parents’:

```java
Person p = new Lieutenant("Paul", "Gries", "Mr.");
System.out.println(p.getHonorific()); // prints Lieutenant
```

Java, at compile time, sees "hey! p is declared as a Person. does Person have a getHonorific() method? Yup!"

Java, at run time, says "What is the actual value of p? Oh, it’s a Lieutenant. Does that have a getHonorific() method?"

If Lieutenant didn’t declare getHonorific(), Java would check if its parent, Person, had one.

This is the essence of overriding. You know that Person’s getHonorific will never get called, just yours (as you have declared yours "on top" of Person’s).
Your task

0.1 Person

Create a Person class. Start fresh: don’t base yours on the posted lecture notes.

1. Use this header for your Person class’ constructor:

   public Person(String firstName, String lastName, String title)

   You might make a new person like this: new Person("Paul", "Gries", "Mr.").

2. Add three fields to your Person class and complete the constructor. (Which fields make sense in the context of that constructor?)

3. Create a public String getHonorific() method that returns the Person’s title.

4. Create a public String getName() method that returns the Person’s full name.

0.2 Representative

Create a Representative class that extends Person and overrides the getHonorific() method to return "The Right Honourable". Calling this should print "The Right Honourable":

   System.out.println((new Representative("Paul", "Gries", "Mr.")).getHonorific())

0.3 Judge

Create a Judge class that extends Person and overrides the getHonorific() method to return "The Honourable".

0.4 President of the United States

If you were to use our getHonorific() method right now, we would have to do something like System.out.println(p.getHonorific() + " " + p.getName()) for a formal address such as “Mr. Paul Gries”.

The president of the United States, however, is always formally addressed as “Mister President” or “Madam President” regardless of their name.

The solution we propose to fix this is to change the getHonorific() method into a getHonorificName() method that will return “Mr. Paul Gries” for a generic person, “The Right Honourable Paul Gries” for a Representative, and “Mister President” for a President.

Ensure this method exists in Person, Representative, and Judge, and then create a President class that overrides the getHonorificName() method to return "Mister President".

Test your code by printing the formal address of President, Representative, Judge, and Person objects with your name in a public static void main(String[] args) method.

0.5 President of the United States #45

Amend President so that it keeps track of the the number of presidents, and has a getJobDescription method that includes that information. For example, the first president object you create should have the following behavior:

   President p1 = new President("David", "Jorjani", "Mr.");
   System.out.println(p1.getJobDescription()); // Prints President of the United States #1
   President p2 = new President("Lindsey", "Shorser", "Dr.");
   System.out.println(p2.getJobDescription()); // Prints President of the United States #2

To achieve such a behavior, you need to create a static presidentCount variable that keeps track of the number of presidents seen so far.

   static variables are shared across the Class instead of the Object. In President’s constructor, you can increment this static variable.

Show your work to your TA – you might not completely finish this lab in your tutorial section, but if that’s the case, you should work it through if you are unsure of overriding or static. They are fundamental properties of Java and you will definitely be tested on your knowledge of them, either directly or indirectly.