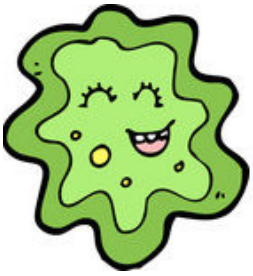
CS161: Introduction to Computer Science
Homework Assignment 6
due 3/15 by 11:59pm

This homework assignment is designed to give you more practice writing Java classes. The first part of this assignment asks you to write an `Organism` class that uses the `Stomach` and `Brain` class you wrote in lab. If you did not finish these classes in lab, you will need to finish them before you can move on. The second part of this assignment asks you to write a class that simulates a playing card.

Note that there is starter code for this assignment that you should download from the course webpage.

Part 1: Organism



Create a new Java class named `Organism`. An organism should have a brain, a stomach, and a name. These represent the instance variables for the organism. Add a constructor to initialize your instance variables. You should not copy-and-paste the code from the `Brain` and `Stomach` class inside the organism. This defeats the purpose of encapsulation. Instead, you should *use* the `Brain` and `Stomach` classes inside the organism.

Next, add the following methods to your class. Most of these methods will be one-line methods that simply call the methods in the `Stomach` and `Brain` class.

- `public void eat(int amount)` – this method should cause the organism to eat the specified amount of food
- `public void digest()` – this method should cause the organism to digest a random amount of food
- `public void think(String newThought)` – this method should cause the organism to think a thought
- `public void remember()` – this method should cause the organism's current thought to be stored in its memory
- `public void recall()` – this method should cause the organism's memory to be loaded back into its current thought
- `public void wakeUp()` – this method should cause the organism to wake up
- `public void sleep()` – this method should cause the organism to go to sleep
- `public String toString()` – this method should return a string representing the state of the organism
- `public boolean equals(Organism other)` – this method returns true if both organisms have the same name and false otherwise

Reproduction

Now we'll add the ability for our organisms to reproduce. In reality, if our organism could not reproduce, it would quickly become extinct.

1. Start by adding asexual reproduction to the `Organism` class. Create a method inside the `Organism` class called `asexuallyReproduce()` that *returns* a new organism. The name of the new organism should somehow be related to the name of the original organism. For example, if your organism is named "ozzy" then the new organism might be named "ozzy 1".

2. Continuing with the previous idea, modify your `Organism` class so that it keeps track of the number of children produced and numbers them accordingly. I.e., the first time the `asexuallyReproduce()` method is called, the new organism has the name "ozzy 1". The next time the method is called, the new organism has the name "ozzy 2". Then "ozzy 3", "ozzy 4", etc.

3. Continuing with this idea, how could you implement sexual reproduction? Create a method called `sexuallyReproduce()` that takes as input *another* organism and *returns* a new organism whose name is a combination of both of its parents. This method is similar to the `equals()` method in that it takes as input another organism and accesses that organisms' instance variables.

Once your `Organism` class is written and compiles, you can start creating actual organisms. Create a new class called `OrganismController` that has a `main()` method. Inside the `main()` method, ask the user to type in a name and create an organism with that name. Have the organism perform at least 5 different actions.

Next, call both `assexuallyReproduce()` and `sexuallyReproduce()` to create children and grandchildren for your original organism. Add enough print statements so that when I execute your `main()` method, it is clear what is happening.

Playing Cards

The next class you will be writing represents a playing card. A playing card has the following attributes:

- A face value which is an integer ranging from 1 to 13. A face value of 1 corresponds to an Ace. A face value of 11, 12, or 13 corresponds to a Jack, Queen, or King respectively. These three are known as *face* cards because they usually have a face drawn on them.
- A suit which is either: `diamond`, `heart`, `spade`, or `club`. Diamonds and hearts are red in color. Spades and clubs are black in color.

The starter code for this assignment has a `Card` class that I have already started for you. Add the appropriate instance variables to represent the card's face value and suit. Next, add the following methods:

- A constructor that takes a suit and a face value as input arguments.
- A constructor that takes no inputs. This constructor should randomly generate the card's face value and suit
- Getter methods for the suit and face value
- `isBlack()` – this method returns whether the card is black
- `isRed()` – this method returns whether the card is red
- `isFaceCard()` – this method returns whether the card is a face card
- `hasSameFaceValue(Card other)` – this method returns whether this card and the other card have the same face value
- `hasSameSuit(Card other)` – this method returns whether this card and the other card have the same suit
- `equals(Card other)` – this method returns whether this card and the other card are equal
- `outRanks(Card other)` – this method returns true if this card has a strictly greater face value than the other card. The only exception is a face value of 1 (i.e. an Ace) which outranks all other face values.

- A `toString()` method. Your `toString()` method *must return a string in this precise format*:

  `[suit, faceValue]`

  where a face value of 1 is converted to an "A", an 11 to "J", a 12 to "Q" and a 13 to "K". For example, an ace of spades should return `[spade, A]` and a three of clubs should return `[club, 3]`. (My tester code relies upon your `toString()` method returning this exact format.)

When you are done, write a `CardController` class that creates 2 cards and calls at least 5 methods on them. Add enough print statements so that when I run your code, it is clear what is happening.

Style Guide

Be sure to go through the list below carefully and make sure your code adheres to the style guide:

- Delete any unused instance variables
- If an instance variable isn't used in multiple methods, then consider putting it inside of the method where it is used.
- All non-final instance variables should be initialized in the constructor
- You have a Javadoc comment at the top of the class with a brief description (written in full English sentences), you and your partner's name, and the date.
- Each method you write has a Javadoc comment with appropriate `@param` and `@return` statements
- All variable names are lower cased (remember, only classes are capitalized in Java)
- Use `final` when appropriate
- Use inline comments (`//`) to explain any complicated code

Submitting your assignment

Please make sure to rename your folder before zipping. You should rename your folder using both of your first and last names. For example, `hw6_John_Doe_Jane_Doe`.

Submit your zipped folder via Moodle.