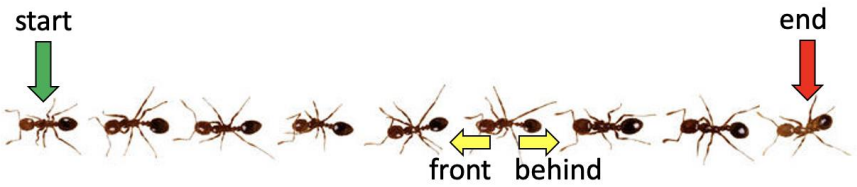
CSC 232: Object-Oriented Software Development
Homework #2: A Chain of Ants
Due: Tuesday, October 20th by 11:59pm (Greencastle time)

This homework is intended for you to practice creating/using multiple classes and objects in your Java programs. More importantly, this homework is designed for you to practice using **non-primitive (object) variables** in your methods. You may work on this homework with up to 1 partner (no more) but you must email me your partner's name by the end of this week – be sure to CC your partner on the email that you send. If you do not send me an email but work with someone and both of you submit the same code, unfortunately I will have to treat it as an Academic Dishonesty Violation and file a report to the University.

In nature, ants are incredible organisms that have the ability to cooperate with one another and form a long chain (*see example image of an ant chain below*).



In this homework, you will create an Ant class as well as an AntChain class that supports various methods (i.e., capabilities) to manipulate the AntChain object.

An Ant Class

- Begin by creating an **Ant** class
- Each Ant object should store the following:
  - Its name as a String (example: "Antoine")
  - A reference ("pointer") to the Ant object that is in **front** of the ant
  - A reference ("pointer") to the Ant object that is **behind** the ant
- Your Ant class should have a **constructor** that takes a single String as a formal parameter to initialize a single, lonely ant's name variable. Recall: A non-primitive variable (i.e., **front** and **behind**) will be initialized to their default value (i.e., **null**) since the ant is not part of a chain yet)
- Write appropriate "getters" for each of the three member variables above
- Write appropriate "setters" for each of the three member vairables above

An AntChain Class

- Next, create an **AntChain** class
- An AntChain object should store/maintain the following:
  - A reference ("pointer") to the Ant object at the **start** of the chain
  - A reference ("pointer") to the Ant object at the **end** (rear) of the chain
  - A count of the number of Ant objects that are currently in the chain
- The default constructor that Java provides automatically is sufficient, therefore, we do not need to implement any constructors since start and end will be initialized to their default value (**null**) and numAnts will be intiailized to its default value (**0**)
  Important: After calling **any** of the methods described below, the AntChain should always uphold the following:
  1. The start pointer should always point at the Ant object that is at the start of the chain
  2. The end pointer should always point at the Ant object that is at the end of the chain
  3. Each Ant object's front pointer should always point at the correct Ant object that is in front of it
  4. Each Ant object's behind pointer should always point at the correct Ant object that is behind it
  5. If an Ant object does not have an Ant object in front of it, then it should point at null (i.e., no ant)

6. If an Ant object does not have an Ant object behind it, then it should point at null (i.e., no ant)
7. The count of the number of Ant objects in the chain should always be up-to-date

- A method named **rollCallFromStart** that does not take any formal parameters and that prints the names of the ants (*in order*) from the start of the chain to the end (rear) of the chain. If there are no ants in the chain, then this method should not print anything.
- A method named **rollCallFromEnd** that does not take any formal parameters and that prints the names of the ants (*in order*) from the end (rear) of the chain to the start of the chain. If there are no ants in the chain, then this method should not print anything.
- A method named **getNumberOfAnts** that does not take any formal parameters and returns how many ants are currently in the chain
- A method named **addToEnd** that takes an ant's name as a String formal parameter and creates a new Ant object with the provided name and adds/links this ant object appropriately to the end (rear) of the AntChain
- A method named **addToStart** that takes an ant's name as a String formal parameter and creates a new Ant object with the provided name and adds/links this ant object appropriately to the start of the AntChain
- A method named **addAt** that takes two formal parameters: (1) an ant's name as a String and (2) an index as an integer. This method should add a new Ant object with the provided name at the specified index (position) in the chain. For example, addAt("Jane", 3) should result in a new Ant object with the name "Jane" being added as the 3rd ant in the chain. Be sure to handle illegal indices that the user might accidentally provide (i.e., too small or too large) by simply printing out "Can't add an Ant"
- A method named **positionOf** that takes an ant's name as a String formal parameter and returns the index (i.e., position) of the Ant in the chain. For example, the 1st ant is at position 0, the 2nd ant is at position 1, the 3rd ant is at position 2, etc. If there is not an ant in the chain with that name, then this method should return **–1**. If there are multiple ants in the chain with the same name, it should return the lowest index (position) of the matching ant.
- A method named **kickout** that takes an ant's name as a String formal parameter and removes the ant in the chain whose name matches. If there are multiple ants in the chain with the same name, it should remove ("kickout") **only** the **first** matching ant that it finds.
- A method named **connectToChain** that takes another AntChain object as a formal parameter. This method should connect the end of the AntChain that the method was called on to the beginning of the AntChain that was passed as a formal parameter. Be sure to test your method on different possible AntChains (e.g., empty chains, chains where the size is smaller/bigger than the other, etc.)
- A method named **isSame** that takes another AntChain object as a formal parameter. This method should return true if the AntChain the method was called on has the **same Ant names in the same order** as the AntChain object that was passed as a parameter. Be sure to test your method on different possible AntChains (e.g., empty chains, chains where the size is smaller/bigger than the other, etc.)

Tips:

- You should use your Driver class to test your methods **thoroughly** to ensure that they are working correctly, as this is a critical skill for a programmer to know how to do. Specifically, you should print your AntChain's names from both directions (front-to-back and back-to-front) **each time that you implement a new method** to ensure that the pointers are connected correctly.
- Be sure to incorporate good object-oriented design principles ("rules of thumb") that we have discussed so far this semester
- I will not grade your Driver file – the only code that I will grade will be what you put in your Ant.java and AntChain.java classes
- Start early – do not wait until a few days before the deadline to start this homework (**no extensions will be granted – see late penalty policy in syllabus**)
- If you have questions relating to what I am asking for, I am more than happy to clarify. However, I will not look at your project's source code nor will I give "hints" about whether you are on the right track or

how to solve the problem. This is a graded assignment and I am assessing your ability to design and implement a solution – not mine.

Submission:

- You should develop your Homework 02 project in Visual Studio Code
- After you have thoroughly tested your methods to ensure they are working correctly and are ready to submit your code, you should first ensure all changes have been saved to your project's files (recall: if there is a solid circle next to the name of your file on its Visual Studio Code tab, then it has not been fully saved yet). Then, **right-click** on your project's folder in your Windows/Mac file system. If you are a a Windows user, then select **Send To …** >> **Compressed (zipped) folder**. A new file with "zipped folder" icon should appear that has your project's same name. If you are a Mac user, then selec the **Compress …** option and a new file with .zip extension should appear alongside your project's folder. Open up a web browser, navigate to our DePauw Moodle page. In our course's Moodle page, you should see a link for **Homework 02**. When you select this item, it will present you with an option to upload your submission. Upload the compressed (zipped) folder that you created in the step above and press Submit.