# Homework 05 - Cheese

## Purpose

To complete this assignment, you will apply your knowledge of constructor chaining, the `this` keyword, encapsulation, method overloading, object equality, and `null`. As we get further into the course, we are starting to learn more elements of Object Oriented Programming. As such, you will create three classes - `Cheese`, `CheeseConnoisseur`, and `CheeseDriver` - that simulate how cheese connoisseurs might behave in the real world. Remember to use good design! This is also the first assignment where you will write javadocs, so be sure to read the "Checkstyle and Javadocs" section for more information.

## Directions

For this assignment, you will be creating three classes to simulate how cheese connoisseurs might behave in the real world: one class representing the cheese (`Cheese.java`), one class representing the cheese connoisseurs (`CheeseConnoisseur.java`), and one class to run the simulation (`CheeseDriver.java`). You will also javadoc your code for this assignment, so be sure to read the "Checkstyle and Javadocs" section.

`Cheese.java`

This Java file represents some cheese from a grocery store. Once a cheese has been traded 3 times, it becomes sentient. (Sentient means able to perceive or feel things).

- Instance Variables
  Pick appropriate variable types and names. All fields should follow the rules of encapsulation.
  - Holds the type of the cheese (examples: "Pepperjack", "Cheddar", etc.)
  - Holds the price of the cheese
  - Holds whether or not this cheese is sentient
  - Holds a counter of how many times this specific `Cheese` object has been traded.
  - Holds how many cheese trades have happened across all cheeses
    - This variable should be the same across all instances of `Cheese`
- Constructors
  - A constructor that takes in the price, type, and number of trades for that specific `Cheese` instance.
    - A cheese should not be sentient at instantiation unless it has already been traded 3 times.
  - A constructor that takes in price, type.
    - A cheese should not be sentient at instantiation.
    - By default, a `Cheese` object has been traded 0 times.
- Methods
  - A `toString` method that converts a `Cheese` object to a `String`
    - If the cheese is not sentient, the `String` should be: "This is a slice of [type] cheese that has been traded [number of trades] times."
    - If the cheese is sentient, the `String` should be: "I am a slice of [type] cheese that has been traded [number of trades] times."
  - An `equals` method (as learned in class) dependent on type, price, and sentience.
    - We're looking for an overloaded `equals` method here, not an overridden `Object equals` method. (If you don't know what that means, don't worry! We'll be covering it in Week 7).
  - A method for when a cheese is traded
    - Increment the appropriate variables
    - A cheese is sentient if it has been traded at least 3 times.
      - When cheese becomes sentient, it doubles in price.
      - When cheese becomes sentient, it should print "I'm ALIIIIIVE!"
  - getters/setters as needed

`CheeseConnoisseur.java`

This Java file represents a cheese connoisseur who owns a singular cheese.

- Instance Variables
  Pick appropriate variable types and names. All fields should follow the rules of encapsulation.
  - Holds the name of the connoisseur.
  - Holds one cheese owned by the connoisseur.
  - Holds how much money the connoisseur has.
- Constructors
  - One constructor that takes in name, money, and cheese
  - One constructor that takes in name, money
    - Should give the connoisseur a `null` `Cheese` that can be filled later.
    - This should use constructor chaining.
  - One constructor taking name.
    - The connoisseur should have no money.
    - The connoisseur should have no cheese at the moment.
    - This should use constructor chaining.
  - One constructor with no arguments.
    - The connoisseur's name should be "Louis Pasteur"
    - The connoisseur should own no cheese at the moment.
    - The connoisseur should have $20.00
- Methods
  - One method that allows a `CheeseConnoisseur` to make a trade. It will take in a `Cheese` object.
    - The connoisseur should replace their current cheese with the cheese that is passed in to the method.
    - Both cheese trade variables should increase by 1 (instance and object).
    - If a cheese has been traded 3 times, it should become sentient.
  - Another method that allows a CheeseConnoisseur to make a trade. It will take in a `CheeseConnoisseur`. This method should overload the previous method.
    - If one of the cheese connoisseurs has a `null` cheese, the trade will not happen.
    - If the two cheeses are equal, the connoisseurs do not trade `Cheese` objects.
    - If they are not equal, the connoisseur who gives away the less expensive cheese should pay the difference, then they trade cheese. If they do not have enough money to pay the difference, the connoisseurs will not trade `Cheese` objects.
      - Both cheese trade variables should increase by 1 (instance and object).
      - If a cheese has been traded 3 times, it should become sentient.
  - Getters/setters as needed

`CheeseDriver.java`

This Java file is a driver, meaning it will run the simulation. You can also use it to test your code. We require the following in your `CheeseDriver`:

- Create at least 4 `Cheese` objects
- Create at least 2 `CheeseConnoisseur` objects.
  - One of the `CheeseConnoisseur` should be initialized with no cheese
- Execute trades until at least 3 `Cheese` objects are sentient.

Again, this is the minimum. You are encouraged to test your code more. Our requirements for `CheeseDriver` are not comprehensive.

### Tips and Tricks

- Create methods and variables as necessary. The ones listed above are just the ones we require.

- Test your code yourself! Neither the autograder nor our requirements for `CheeseDriver` are comprehensive.

*Note: Make sure your classes are all in separate files and that all of your files are in the same directory.*

## Allowed Imports

To prevent trivialization of the assignment, no imports are allowed.

## Checkstyle and Javadocs

You must run checkstyle on your submission. The checkstyle cap for this assignment is **20** points. If you don't have checkstyle yet, download it from Canvas -> Files/Resources. Place it in the same folder as the files you want checkstyled. Run checkstyle on your code like so:

```
$ java -jar checkstyle-8.28.jar yourFileName.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off (limited by the checkstyle cap mentioned above). The Java source files we provide contain no Checkstyle errors. In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

Additionally, you must javadoc your code.

Run the following to only check your javadocs.

```
$ java -jar checkstyle-8.28.jar -j yourFileName.java
```

Run the following to check both javadocs and checkstyle.

```
$ java -jar checkstyle-8.28.jar -a yourFileName.java
```

## Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or breaks our auto grader. For that reason, do not use any of the following in your final submission: * var (the reserved keyword) * System.exit * Runtime.getRuntime.halt * Runtime.getRuntime.exit

## Collaboration

### 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 `//`.

## Turn-In Procedure

### Submission

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

- `Cheese.java`
- `CheeseConnoisseur.java`
- `CheeseDriver.java`

Make sure you see the message stating "HW05 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. 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 usage validation (e.g. forbidden imports, reserved keywords, etc)

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.

### Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Do not submit `.class` files.
- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- Check on Piazza for all official clarifications

---