Homework 05 – The Dessert Conundrum

Authors: Nathan, Chloe, Jack, Melanie, Chelsea, Niki
Topics: ArrayList with generics, Comparable, asymptotics, searching, sorting

Problem Description

Bob's favorite food are desserts (as it should be for you too!). Unfortunately, Bob is also a very picky eater and needs help figuring out which dessert to eat today. Help Bob pick the best dessert to complete his meal!

Solution Description

You will need to complete and turn in 5 classes: Dessert.java, Cake.java, IceCream.java, Store.java, Bob.java.

Notes:

1. All variables should be inaccessible from other classes and require an instance to be accessed through, unless specified otherwise.
2. All methods should be accessible from everywhere and must require an instance to be accessed through, unless specified otherwise.
3. You may assume that inputs to all constructors and methods will be valid (i.e., primitives are reasonable values, and references are non-null).
4. Floating-point values in Strings should be displayed to two decimal places with rounding.
5. Use constructor chaining and reuse methods whenever possible! Ensure that your constructor chaining helps you **maximize code reuse**!
6. Make sure to add Javadoc comments to your methods and classes!

Dessert.java

This class is the superclass for other types of dessert and defines the basic behaviors of desserts. It should never be instantiated. This class should implement the Comparable interface using generics so that it is comparable only to other Desserts.

Variables:

- `String flavor` – the flavor of this cake
- `double sweetness` – the sweetness of this cake

Constructors:

- A constructor that takes in `flavor` and `sweetness` of the dessert.
- A no-argument constructor that sets `flavor` to "vanilla" and `sweetness` to 25.0.

Methods:

- `toString`
  - Overrides the `toString` method from Object.
  - Returns a String in the following format (single line, without curly braces, without quotation marks):

```
"{flavor} dessert with a sweetness of {sweetness}."
```

THIS GRADED ASSESSMENT IS NOT FOR DISTRIBUTION.
ANY DUPLICATION OUTSIDE OF GEORGIA TECH'S LMS IS UNAUTHORIZED.

- `equals`
  - Overrides the `equals` method from Object.
  - The implementation must be symmetric.
  - Two desserts are equal if they have the same `flavor` and `sweetness`.
- `compareTo`
  - Overrides the `compareTo` method from the Comparable interface.
  - Desserts should be compared based on `sweetness` first, and then `flavor`. That is:
    - A dessert is considered greater than the other if it has greater `sweetness`.
    - If both desserts have equal `sweetness`, then the greater dessert is the one with the lexicographically greater `flavor`.
      - **Hint:** Look in the String API for a method that *compares* Strings lexicographically!
  - **Note:** Since this implementation only compares the fields of the Dessert class and not those of subclasses, it should only be used to determine a natural ordering between desserts. It should not be used to determine object equality.
- Getters and setters as necessary.

Cake.java

This class is a subclass of Dessert and represents a certain kind of dessert that Bob can pick: cake!

Variables:

- `String frosting` – the type of frosting of this cake

Constructors:

- A constructor that takes in `flavor`, `sweetness`, and `frosting`.
- A constructor that takes in `flavor` and sets `sweetness` to 45.0 and `frosting` to "vanilla".

Methods:

- `toString`
  - Overrides the `toString` method from Dessert.
  - Returns a String in the following format (single line, without curly braces, without quotation marks):

```
"{flavor} cake with a {frosting} frosting and has a
sweetness of {sweetness}."
```

- `equals`
  - Overrides the `equals` method from Dessert.
  - The implementation must be symmetric.
  - Two cakes are equal if they have the same `flavor`, `sweetness`, and `frosting`.
- No getters or setters.

IceCream.java

This class is a subclass of Dessert and represents a certain kind of dessert that Bob can pick: ice cream!

Variables:

- `int scoops` – the number of scoops of this ice cream
- `boolean cone` – whether this ice cream has a cone

Constructors:

- A constructor that takes in `flavor`, `sweetness`, `scoops`, and `cone`.
- A constructor that takes in `scoops` and `cone` and sets `flavor` to "vanilla" and `sweetness` to 45.0.
- A no-argument constructor that sets flavor to "vanilla", sweetness to 45.0, scoops to 1, and cone to false.

Methods:

- `toString`
  - This method should properly override Dessert's `toString` method.
  - It should return a String in the following format (single line, without curly braces, without quotation marks):

    `"{flavor} ice cream with {scoops} scoops and {has/does not have} a cone."`

    - The "has/does not have" depends on the value of the `cone` field.
- `equals`
  - Overrides the `equals` method from Dessert.
  - The implementation must be symmetric.
  - Two ice creams are equal if they have the same `flavor`, `sweetness`, `scoops`, and `cone`.
- Getters and setters as necessary.

Store.java

This class represents a store that sells all kinds of dessert.

Variables:

- `String name` – the name of this store
- `ArrayList<Dessert> desserts` – the desserts this store sells
  - This list should always be initially empty.

Constructors:

- A constructor that takes in `name`.

Methods:

- `addDessert`
  - Takes in a `Dessert` object and adds it to the back of `desserts`.
  - Returns nothing.
  - Runs in $O(1)$ time.

- `removeDessert`
  - Takes in a `Dessert` object and removes the first occurrence of an equal dessert in `desserts`.
  - Returns the removed `Dessert`.
    - If the dessert is not found, return null.
    - **Note:** If a dessert is removed, you should return the dessert that was removed from the store rather than the inputted dessert item.
  - Use the dessert item's `equals` method to identify the first occurrence of the inputted dessert item in the store.
  - Runs in $O(n)$ time.
- `findDessert`
  - Takes in a `Dessert` object and finds and returns the `Dessert` that has the same `sweetness` and `flavor`.
    - **Note:** You should only consider the sweetness and flavor, not any other fields.
    - **Note:** If a dessert is found, you should return the dessert that was found in the store rather than the inputted dessert item.
  - You should make the following assumptions:
    - If the dessert can be found in the store, then that dessert in the store will be equal to the inputted dessert according to the `equals` method.
    - The elements of `desserts` are **unique** in sweetness and flavor and is **sorted** in ascending order according to their natural ordering.
  - If a dessert with the same sweetness and flavor is not found, return null.
  - Runs in $O(log\ n)$ time.
- `sortStore`
  - Takes no parameters, does not return anything.
  - Sort `desserts` in **ascending** order based on their sweetness first, then flavor.
    - **Hint:** What method compares two desserts based on sweetness first, then flavor?
  - Runs in $O(n^2)$ time.
- `countGreaterDesserts`
  - Takes in a valid `Dessert` object and returns the number of desserts in the store that is considered greater than or equal to the dessert passed in.
    - Equivalently, return the number of desserts in the store that are *not* lesser than the inputted dessert.
    - Dessert comparison should be based on sweetness first, then flavor.
  - Runs in $O(n)$ time.
- `showMenu`
  - Prints the store's menu, including the store's name and desserts available in the store.
  - Print a string in the following format (single line, without curly braces, without quotation marks):

    ```
    "{name}'s Menu of the Day:"
    ```

  - Display the string representation of each dessert on their own line, starting from index 0.
  - Runs in $O(n)$ time.
- Getters and setters as necessary.

Bob.java

This class defines Bob's behaviors. All methods should be static. Notice that you have already implemented most of the code necessary for Bob's behaviors, so reuse code as much as possible!

Methods:

- `compareStores`
  - Takes in two `Store` objects, `store1` and `store2`.
  - Returns true if all desserts in `store1` are found in `store2`. Returns false otherwise.
  - You should make the following assumptions:
    - If `store2` has a dessert with the same sweetness and flavor as the dessert from `store1`, then these desserts will be equal according to the `equals` method.
    - The desserts in `store2`'s inventory are **unique** in sweetness and flavor and are **sorted** in ascending order according to their natural ordering.
  - **Hint**: What method can you reuse to search for a dessert in a sorted store?
  - Runs in $O(n \log n)$ time.
- `shop`
  - Takes in a `Store` object and a `Dessert` object.
  - The store's desserts should become sorted in ascending order according to their natural ordering to help Bob find the dessert Bob is looking for.
  - Returns true if Bob is able to find a dessert with equal sweetness and flavor, even if the desserts are of different types. Returns false otherwise.
    - **Note:** You should only consider the sweetness and flavor, not any other fields.
  - **Hint**: What method can you reuse to search for a dessert in a sorted store?
  - Runs in $O(n^2)$ time.
- `findAvailableStores`
  - Takes in an `ArrayList` of `Store` objects, where `desserts` of each store is **sorted** in ascending order according to their natural ordering, and a target `Dessert` that Bob is looking for.
    - **Note**: Bob will be satisfied with any dessert from the store that has the same sweetness and flavor as the input dessert, even if they are different types.
  - Returns an `ArrayList` of `Store` objects containing only the stores that have the dessert Bob is looking for.
  - Runs in $O(n \log m)$ time, where n is the number of stores and m is the number of desserts in each store.
    - You may assume that all stores have the same number of desserts (i.e., m).

Checkstyle

You must run Checkstyle on your submission (to learn more about Checkstyle, check out cs1331-style-guide.pdf under the Checkstyle Resources module on Canvas). **The Checkstyle cap for this assignment is 30 points.** This means there is a maximum point deduction of 30. If you don't have Checkstyle yet, download it from Canvas → Modules → Checkstyle Resources → checkstyle-8.28.jar. Place it in the same folder as the files you want to run Checkstyle on. 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 number of points we would take off (limited by the Checkstyle cap). In future assignments 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
```

For additional help with Checkstyle see the CS 1331 Style Guide.

Turn-In Procedure

Submission

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

- `Dessert.java`
- `Cake.java`
- `IceCream.java`
- `Store.java`
- `Bob.java`

Make sure you see the message stating the assignment was submitted successfully. From this point, Gradescope will run a basic autograder on your submission as discussed in the next section. **Any autograder tests are provided as a courtesy to help "sanity check" your work and you may not see all the test cases used to grade your work.** You are responsible for thoroughly testing your submission on your own to ensure you have fulfilled the requirements of this assignment. If you have questions about the requirements given, reach out to a TA or Professor via the class forum for clarification.

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

Gradescope Autograder

If an autograder is enabled for this assignment, you may be able to see the results of a few basic test cases on your code. Typically, tests will correspond 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. **We reserve the right to hide any or all test cases, so you should make sure to test your code thoroughly against the assignment's requirements.**

The Gradescope tests serve two main purposes:

- Prevent upload mistakes (e.g., non-compiling code)
- 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.

Burden of Testing

You are responsible for thoroughly testing your submission against the written requirements to ensure you have fulfilled the requirements of this assignment.

Be **very careful** to note the way in which text output is formatted and spelled. Minor discrepancies could result in failed autograder cases.

If you have questions about the requirements given, reach out to a TA or Professor via the class forum for clarification.

Allowed Imports

- `java.util.ArrayList`

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our autograder. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`
- `System.arraycopy`

Collaboration

Only discussion of the assignment at a conceptual high level is allowed. You can discuss course concepts and assignments broadly; that is, at a conceptual level to increase your understanding. If you find yourself dropping to a level where specific Java code is being discussed, that is going too far. Those discussions should be reserved for the instructor and TAs. To be clear, you should never exchange code related to an assignment with anyone other than the instructor and TAs.

The only code you may share are test cases written in a Driver class. If you choose to share your Driver class, they should be posted to the assignment discussion thread on the course discussion forum. We encourage you to write test cases and share them with your classmates, but we will not verify their correctness (i.e., use them at your own risk).

You **MAY NOT** use code generation tools to complete this assignment. This includes generative AI tools such as ChatGPT.

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 Ed Discussion for a note containing all official clarifications and sample outputs.**

It is expected that everyone will follow the Student-Faculty Expectations document and the Student Code of Conduct. The professor expects a **positive, respectful, and engaged academic environment** inside the classroom, outside the classroom, in all electronic communications, on all file submissions, and on any document submitted throughout the duration of the course. **No inappropriate language is to be used, and any assignment deemed by the professor to contain inappropriate offensive language or threats will get a zero.** You are to use professionalism in your work. Violations of this conduct policy will be turned over to the Office of Student Integrity for misconduct.

THIS GRADED ASSESSMENT IS NOT FOR DISTRIBUTION.
ANY DUPLICATION OUTSIDE OF GEORGIA TECH'S LMS IS UNAUTHORIZED.