# Programming Exercise 05

## Problem Description

Hello and welcome to another exciting programming exercise! The goal of this PE is to become more familiar with the general structure and components of classes in Java. Throughout the PE, you will be seeing and implementing concepts relating to constructors, instance data, instance methods, and object equality.

## Solution Description

Close your eyes (figuratively, of course, because you have more to read). The year is 2020 and you are in a deep and peaceful sleep. You are dreaming about your next lecture in the best CS course that there is, CS 1331, when suddenly the GT servers containing all of the students' data becomes corrupted. You awake to an email letting you know that GT OIT was able to recover most of the lost information, but now they want to reorganize the information using Java objects and need your expertise to do it. Your task is to create a Java class that keeps track of all the students' basic information.

Your program class must be named `Student.java`. This class should only contain instance data and methods. For testing purposes, we recommend you create another Java file with a `main` method, but you do not need to turn in your test file. The specifications for the program are as follows:

### *Instance Data*

a. `name`: The name of the student. Should be represented as a **constant** String value since a student's name should not change after being set.
b. `gtid`: This should be a numeric whole number that represents the student's GT Id (i.e. 903xxxxxx).
c. `classYear`: This should be a whole number value between [1, 4] that represents the student's year in college.
d. `gpa`: This should be a decimal value between [0.0, 4.0] that represents the student's grade point average.
e. `diningDollar`: Should be a decimal value that represents the student's current amount of dining dollars
f. `buzzfund`: Should be a decimal value that represents the student's current amount of buzzfunds
g. All of the above values should be private

### *Constructors*

a. It must be possible to create a student with the following combinations of initial data provided as parameters on construction
    i. `name, gtid, classYear, gpa, diningDollar, buzzfund`
    ii. `name, gtid, classYear`
    iii. `name, diningDollar, buzzfund`
    iv. `name`
b. A `name` for the student must always be provided on construction
c. The default value for `gtid` must be –1 when not provided on construction.
d. The default value for `classYear` must be 1 when not provided on construction or if the provided parameter is invalid (not [1,4])

e. The default value for `diningDollar` and `buzzfund` must both be 0.0 when not provided on construction or if the provided parameter is negative.
f. The default `gpa` must be 4.0 when not provided on construction or if the provided parameter is invalid (not [0.0, 4.0])
g. **Note: Constructor chaining MUST be used where possible.**

### *Instance Methods*

Below you are given the method signatures and specification for the Student class. Make sure these are instance methods and the signatures match exactly.

a. `purchaseMeal(double cost)`
    i. Reports whether there are enough funds to purchase the meal and updates the student's funds accordingly. The cost should be deducted from the object's `diningDollar` first and from `buzzfund` only when the dining dollars are no longer available. If a student has 5.0 dining dollars and 5.0 buzzfunds, and attempts to purchase a meal costing 7.0, they should end up with 0 dining dollars and 3.0 buzzfunds.
    ii. Prints out when the cost is successfully deducted by the appropriate fund variable: "{name} has successfully purchased the meal"
        1. There is a newline character at the end of the String
        2. {name} should be the name of the student
    iii. If there are not enough funds to purchase the meal, prints out: "{name} does not have enough funds to purchase the meal"
        1. There is a newline character at the end of the String
        2. {name} should be the name of the student
b. `addDiningDollars(double diningDollarVal)`
    i. Adds the passed in amount of `diningDollar` to the instance field `diningDollar`
c. `addBuzzfund(double buzzfundVal)`
    i. Adds the passed in amount of `buzzfund` to the instance field `buzzfund`
d. `increaseClassYear()`
    i. Increments the current `classYear` of the `Student` object by 1, if the `classYear` is 4, then it should remain as 4
e. `toString()`
    i. Returns a string that is in the format of: "Name: {name}, GTID: {gtid}, GPA: {gpa}, Class Year: {classYear}, Funds: {buzzfund + diningDollar}"
        1. Do not include {}
        2. There is no newline character
f. `equals(Student other)`
    i. Two `Student` objects are considered to be equal if they have the same `gpa` and are of the same `classYear`, and have the same `names`
    **ii.** Return true if the above criteria is met, or false if it is not.
g. **Note: Use the exact method signatures provided above**

## Checkstyle

You must run checkstyle on your submission. If you don't have checkstyle yet, download it from Canvas -> Files -> Resources in the A/B/GR section and in Modules -> General Information in the D section . 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 in the Rubric section). In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

## Rubric

Below is a guide for how your assignment will be graded, however this may not be inclusive, you must follow the specification given in the prompt above.

[100] **Student.java**

- Instance Data
  - Instance fields are properly named and initialized
  - Instance fields are all private
- Constructor
  - Uses constructor chaining
  - Contains all necessary constructors
- Instance Methods
  - `purchaseMeal` is output is correct
  - `purchaseMeal` properly deducts from instance field
  - `addDiningDollars` properly adds to instance field
  - `addBuzzfund` properly adds to instance field
  - `toString` method returns correct value
  - Correct method headers
- Equality
  - Equality check is correct

***Checkstyle will be required on this homework, with a maximum point deduction of 10.***

## Allowed Imports

To prevent trivialization of the assignment, you may only import `java.util.Random`.

## Feature Restrictions

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

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

# Collaboration

## *Collaboration Statement*

To ensure that you acknowledge a 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].*

## *Allowed Collaboration*

When completing homeworks and programming exercises for CS1331 you may talk with other students about:

- What general strategies or algorithms you used to solve problems in the homeworks or programming exercises
- Parts of the homework you are unsure of and need more explanation
- Online resources that helped you find a solution
- Key course concepts and Java language features used in your solution

You may **not** discuss, show, or share by other means the specifics of your code, including screenshots, file sharing, or showing someone else the code on your computer, or use code shared by others.

Examples of approved/disapproved collaboration:

- **approved**: "Hey, I'm really confused on how we are supposed to implement this part of the homework. What strategies/resources did you use to solve it?"
- **disapproved**: "Hey, it's 10:40 on Thursday... Can I see your code? I won't copy it directly I promise"

In addition to the above rules, note that it is not allowed to upload your code to any sort of public repository. This could be considered an Honor Code violation, even if it is after the homework is due.

# Turn-In Procedure

## *Submission*

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

- `Student.java`

Make sure you see the message stating "PE03 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. Each test typically corresponds 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.

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 or run 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 a note containing all official clarifications