# CSE8B - Exam 1 - WI22

Due: Saturday, January 22nd at 1 pm Pacific Time

Hello everyone! Welcome to your first exam for CSE 8B WI22.

You will have over 24 hours to complete this exam, but it should only take one to two hours. The exam will be released on Friday, January 21st at 10AM Pacific Time and will be due by Saturday, January 22nd at 1PM Pacific Time.

This page details a take-home exam that you will complete over the next few days. You can't communicate with anyone about the content of the assignment until you receive your grade. You can message us privately on Piazza, but the course staff will not give programming advice or answer most questions about the problems. If you have technical trouble creating a screencast (detailed below) feel free to reach out for assistance.

Do not use any online service other than Piazza to ask questions about the assignment. Do not search for, solicit, or use solutions to the problems that you find elsewhere for the exam. These are all violations of academic integrity that students have committed on exams like this in the past.

You can make use of any course notes, online resources about Java and its libraries, Java tools, and so on to complete the exam, including reusing code from class notes.

## Outline of the Exam

The exam will be worth 100 points and will be split up into 3 tasks as shown below:

Task 1 - Coding (35 points)
Task 2 - Coding (35 points)
Task 3 - Video (30 points)

## Submission Checklist

- `Point.java` containing all methods for Task 1
- `CovidCondition.java` containing all methods for Task 2
- `explanation.mp4` (or another compatible video extension) that has your recording for Task 3

# Tasks

For Exam 1, you will have 3 tasks to complete.

**NOTE: You do not need to worry about style guidelines for Exam 1.**

## Task 1

For Task 1, you will be making an extension to the Point class that was discussed in the Lecture.

Please download the starter code from this Google Drive link. Ensure that there are two files: `Point.java` and `PointTester.java`. **You will be implementing several methods within `Point.java`, and you will be testing those methods in `PointTester.java`.** For Task 1, you ONLY need to submit `Point.java`.

### `PointTester.java`

Within `PointTester.java` are a few test cases for the Point class. **As you are implementing Point.java, you should test thoroughly to make sure that your program is functioning as expected.** We will not be collecting `PointTester.java`.

### `Point.java`

Within `Point.java`, there are 3 *private* member variables given to you:

- `private int x` - the x-coordinate of the Point (as an integer)
- `private int y` - the y-coordinate of the Point (as an integer)
- `private String type` - the type of the Point (*e.g.* `"Home"`, `"School"`, etc.)

Take note of their declared types. Do NOT change any of their types.

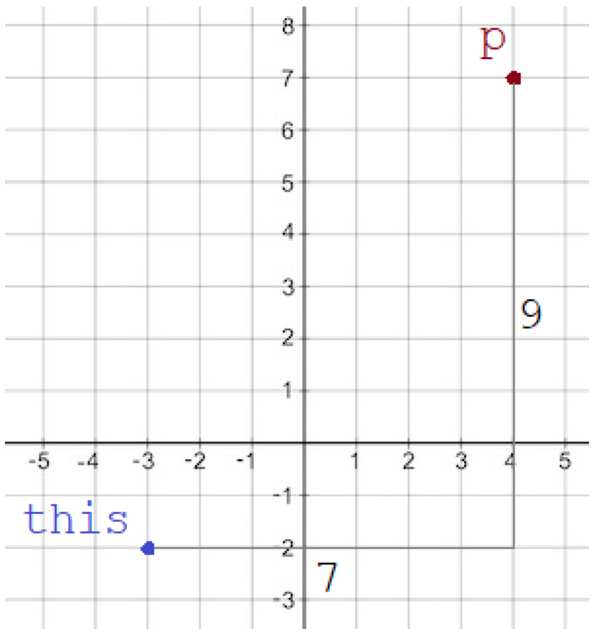
We also provide you several getter methods. Please read and understand them.

**For the rest of `Point.java`, you will need to implement the 4 following methods. DO NOT MODIFY ANY METHOD DECLARATIONS:**

1. `public Point(int x, int y, String type)`
   Within the constructor, you simply need to assign the 3 parameters to their respective member variables (*e.g.* assign the parameter `x` to the member variable `x`, and so on). You can assume that `type` will be a valid String.

2. `public int manhattanDistance(Point p)`
   This method should return the Manhattan distance from `this` point to the parameter Point `p`. You can assume that `p` will be a valid Point.

In a 2D plane with `p₁` at `(x₁, y₁)` and `p₂` at `(x₂, y₂)`, the calculation for the Manhattan distance is $|x_1 - x_2| + |y_1 - y_2|$. For example, if `this` was at `(-3, -2)`, and `p` was at `(4, 7)`, then the Manhattan distance would be $|-3 - 4| + |-2 - 7| = 7 + 9 = 16$. See the image below for a visual example:



*The Manhattan distance is* $7 + 9 = 16$.

**IMPORTANT: For `manhattanDistance` (and for any other methods that require an absolute value calculation), you are NOT allowed to use any packages or classes such as `Math` to perform the absolute value calculation.** We want you to be clever to figure out how to calculate the absolute value without using any other packages or classes!

3. `public boolean checkSameType(Point p)`
   This method should `return true` if `this` point has the same `type` as the parameter Point `p`. Otherwise, this method should `return false`. For example, if `this` has the `type "Store"`, and `p` also has the `type "Store"`, then this method should `return true`. You can assume that `p` will be a valid Point.

4. `public boolean checkIfNearby(Point p, int distance)`
   This method should `return true` if the Manhattan distance between `this` point and the parameter Point `p` is less than or equal to the parameter `distance`. Otherwise, the method should `return false`. For example, if `this` was at `(-3, -2)`, `p` was at `(4, 7)`, and `distance` was `15`, then this method should `return false`. You can assume that `p` will be a valid Point, and that `distance` will be a non-negative integer.

## Task 2

For Task 2, you will be creating a Class to represent the current condition of a State with respect to the COVID situation in that area.

Please download the starter code from this Google Drive link. Ensure that there are two files: `CovidCondition.java` and `CovidConditionTester.java`. **You will be implementing several methods within `CovidCondition.java`, and you will be testing those methods in `CovidConditionTester.java`.** Similar to the previous task, for Task 2, you ONLY need to submit `CovidCondition.java`.

### `CovidConditionTester.java`

Within `CovidConditionTester.java` are a few test cases for the CovidCondition class. **As you are implementing `CovidCondition.java`, you should test thoroughly to make sure that your program is functioning as expected.** We will not be collecting `CovidConditionTester.java`.

### `CovidCondition.java`

Within `CovidCondition.java`, there are 2 *private* member variables given to you:

- **`private String stateName`** - Represents the name of a US state
- **`private int avgCases`** - Represents the average number of cases per day (in thousands)

Take note of their declared types. Do NOT change any of their types.

**For the rest of `CovidCondition.java`, you will need to implement the 4 following methods. DO NOT MODIFY ANY METHOD DECLARATIONS:**

1. `public CovidCondition(String stateName, int avgCases)`
   Within the constructor, you simply need to assign the 2 parameters to their respective member variables (*e.g.* assign the parameter `stateName` to the member variable `stateName`, and so on). NOTE: `avgCases` does NOT have to be strictly non-negative. Keep that in mind when testing.

2. `public String getDetails()`
   This method should return a single String that reads `"[stateName] currently has a daily average of [avgCases] thousand cases"`. **Make sure to replace `[stateName]` and `[avgCases]` with the respective values of the member variables.** For example, if `stateName` was `California`, and `avgCases` was `20`, then the String should read: `"California currently has a daily average of 20 thousand cases"`.

3. `public void updateAvgCases(int numOfCases)`

This method should update the value of `avgCases` to the value of parameter `numOfCases`. HINT: To properly test this method, you should make use of `getDetails()` - this implies that you should ensure that `getDetails()` is implemented correctly!

4. `public String checkTier()`
   This method should `return` the string:
   `"Yellow"` - `if` the value of `avgCases` is strictly less than `10`.
   `"Orange"` - `if` the value of `avgCases` is between `10` and `50` (inclusive).
   `"Red"` - `if` the value of `avgCases` is between `51` and `100` (inclusive).
   `"Purple"` - `if` the value of `avgCases` is strictly greater than `100`.

   For example, if `avgCases` was 23, then `checkTier()` should return `"Orange"`

   *** Please note that the values stated above are not accurate in terms of real-world data. They are only given for the purpose of this exam.**

## Task 3

**For Task 3, you will record a short video of *no more than 5 minutes*.** Please read ALL of the instructions and guidelines below before recording your video.

Consider the following 2 lines of code within the `main` method of `CovidConditionTester.java`:

```
Line 1: CovidCondition state1 = new CovidCondition("California",
120);
Line 2: CovidCondition state2 = new CovidCondition("Utah", 10);
```

In your short video, answer the following 3 questions:

1. If implemented correctly, explain what happens when `state1.getDetails()` is called.
   a. For this question, you should show and trace your code. To be more specific, you should first start inside the `main` method of your tester with the method call (`state1.getDetails()`), show the details of what happens within `getDetails()`, then discuss what `state1.getDetails()` returns.
2. If implemented correctly, explain what `state2.checkTier()` is expected to return.
   a. For this question, you should show and trace your code. To be more specific, you should first start inside the `main` method of your tester with the method call (`state2.checkTier()`), show the details of what happens within `checkTier()`, then discuss what `state2.checkTier()` returns.

3. For the rest of the video, show a memory model diagram that depicts everything in memory *AFTER* the 2 lines of code have finished running (*i.e.* after `Line 1` and `Line 2` finished running). Read below for an example and instructions for creating a memory model diagram. Using your diagram, answer the following questions as descriptive as you can.
    a. For **`Line 1`** above, how and where is memory allocated in the stack and in the heap?
    b. For **`Line 2`** above, how and where is memory allocated in the stack and in the heap?

An example of a memory model diagram can be found at the 30:30 timestamp of A00's Lecture 5 Recording or on slide 6 of B00's Lecture 4. **You can draw the memory model diagram in whatever fashion you would like -** you can draw the diagram on a tablet, draw the diagram on paper then take a picture of the drawing, draw the diagram with a (free) drawing program like Google Drawings or GIMP, or you could just use Google Slides to create shapes and texts. **Whatever you do, we just need to see it in the video.**

Here are the required items to include in your video:

1. **For the first few seconds of the video, show ONLY your face and a picture ID** (your UCSD Student ID is highly preferred, but any picture ID with your full name will suffice). Afterwards, you do not have to be on camera, although it is fine to leave it on. This is simply to confirm that it is you recording your video and doing your work. If you do NOT have a webcam, then take a picture of yourself and your picture ID with your phone, and display that picture at the start of your recording.
2. For the rest of the video, answer the 3 questions above *in that exact order*.

Here are some notes in regards to Task 3:

1. If you do not provide a picture ID, then you may get a 0 on Exam 1 until you prove to us that it was actually you who recorded the video.
2. Please be sure to stand as still as possible when showing yourself and your picture ID. **If we cannot verify that it is you, then you may get a 0 on the exam until you prove to us that it was actually you who recorded the video.**
3. Video must have sound! Furthermore, while explaining your memory model diagram, make sure to enunciate to the best of your ability. We must hear you clearly explain your diagram!

Here is a short tutorial demonstrating how to make a screencast with Zoom: Screencast Tutorial

## Submission Instructions

You will be submitting your answers directly to Gradescope under the assignment **"Exam 1"**. The files required for submission are:

- `Point.java` containing all methods for Task 1

- `CovidCondition.java` containing all methods for Task 2
- `explanation.mp4` (or another compatible video extension) that has your recording for Task 3

**If you cannot upload your video to Gradescope:**
Gradescope has a max file size of 100MB. Any files larger than that will be rejected. All files must be submitted to Gradescope; we will **not** accept videos (or code) through email or Piazza.

If your video is larger than 100MB, then here are some tips to ensure that your video can be uploaded to Gradescope:

- Use Zoom:
  - A 100MB video is about a 20 minute video on Zoom (1620x1080 video resolution).
  - If you are not using Zoom, then convert your file to an MP4. There are web tools that can do this. MP4 videos have smaller video sizes than other video formats.
- Use a smaller screen size for your computer, or only record a smaller portion of the screen.
- Reduce background clutter (*i.e.* desktop icons). Background clutter reduces compression, making larger file sizes. You can hide the background by maximizing your text editor's window.
- Keep your video short (*i.e.* under 5 minutes):
  - 5 minutes is the max, and it is certainly possible to cover everything in less time.
  - We recommend creating a script of what you are going to say.
  - Pause the video as you switch to the code for the next question (make sure to tell us which question you are showing the code for when you switch).
- Use a 3rd party video editor to reduce the width/height of the video, but make sure your code is not blurry when you play it on Gradescope as we cannot grade blurry videos.