CS 61B Fall 2025

Java Basics

Discussion 01: September 1, 2025

1 Class Creation

(a) On the first midterm for this class, you'll be asked to write code from scratch. Below, fill in the code to create a class called Planet. Each planet should have an x position, a y position, and a name. The positions should be doubles, and the name should be a String. There should be one constructor for the planet that sets these values. There should also be a method called **`distanceTo`** that takes in another planet **`other`** and returns the Euclidean distance between **`this`** and **`other`**. As an example, the Euclidean distance between (1, 2) and (3, 4) is

$$\sqrt{(1-3)^2 + (2-4)^2} = \sqrt{8} \approx 2.828$$

```
public class Planet {

  private double x;
  private double y;
  private String name;

  public Planet(double x, double y, String name) {
    this.x = x;
    this.y = y;
    this.name = name;
  }

  public double distanceTo(Planet other) {
    return Math.sqrt(Math.pow(other.x - this.x, 2) + Math.pow(other.y - this.y, 2));
  }

}
```

(b) If the **`distanceTo`** method were made static, would the code still compile and work? If not, how would you change it to make it work?

If the **`distanceTo`** method were made static in its current state, the code would **not** compile. The "**`this`**" keyword cannot be referenced from a static context, because static methods are called on a class rather than an object. The compiler has no way of knowing what "**`this`**" is!

One way to fix the method would be to add the first **`Planet`** to the function's parameters, and access its position that way.

(c) Above, add a new method called **`allPlanetsCreated()`** that returns a **`List<Planet>`** of all planets that have ever been instantiated. That is, if the constructor is used to create a planet called "Mars", and then the constructor is used to create a planet called "Earth", **`allPlanetsCreated()`** should return a list containing these planets named "Mars" and "Earth". You will need to modify other parts of your **`Planet`** class to make this work.

Modified `Planet` class:

```
public class Planet {

  private double x;
  private double y;
  private String name;

  private static List<Planet> allPlanets = new ArrayList<>();

  public Planet(double x, double y, String name) {
    this.x = x;
    this.y = y;
    this.name = name;

    allPlanets.add(this);
  }

  public double distanceTo(Planet other) {
    return Math.sqrt(Math.pow(other.x - this.x, 2) + Math.pow(other.y - this.y, 2));
  }

  public static List<Planet> allPlanetsCreated() {
    return allPlanets;
  }
}
```

2 Fill-in-the-Blank

On the first midterm for this class, you'll often be asked to fill in the blanks. Below, fill in the **maxDistance** method which computes the maximum distance between any pair of planets.

```
public static double maxDistance(List<Planet> planets) {

    double max;
    for (Planet p : planets) {

        for (Planet other : planets) {

            if (p.distanceTo(other) > max) {

                max = p.distanceTo(other);
            }
        }
    }

    return max;
}
```

3 PlanetScanner

Fill in the code below. The PlanetScanner class should have a constructor that takes in a array of planets and the number of planets to scan. It should have two methods:

- **`scannedPlanets`** returns the names of the planets that have been scanned.
- **`scanMorePlanets`** scans more planets.

For example, if we create a **`PlanetScanner`** for an array of 10 planets, with numToScan equal to 4, then **`scannedPlanets()`** should return an array of the names of the first 4 planets. If we then call **`scanMorePlanets(3)`**, then **`scannedPlanets()`** should return an array of all 7 planets scanned so far.

As with midterm problems, you may not need all available space.

This problem has many possible solutions. This is the cleanest one Dawn could think of.

```
public class PlanetScanner {

    private Planet[] allPlanets;

    private Planet[] scannedPlanets;

    public PlanetScanner(Planet[] planets, int numToScan) {
        this.allPlanets = planets;

        this.scannedPlanets = new int[0];;

        this.scanMorePlanets(numToScan);
    }

    public Planet[] scannedPlanets() {
        return this.scannedPlanets;
    }

    public void scanMorePlanets(int numToScan) {
        int newSize = this.scannedPlanets.length + numToScan;

        Planet[] newScan = new Planet[newSize];


        for (int i = 0; (i < newSize) && (i < allPlanets.length); i++) {
            newScan[i] = this.allPlanets[i];
        }


        this.scannedPlanets = newScan;
    }
}
```

4 Helpful LLM Use

On HW2, you had a chance to play around with large language models for solving the **`starTriangle`** problem. You've probably also used LLMs in prior programming classes.

What are some ways that you've used LLMs to help your learning? What are some ays that you've used LLMs that actually hindered your learning? Did you find it helpful to see what an LLM came up with for **`starTriangle`**?