Indiana University
CSCI-C212: Introduction to Software Systems (F24) Lab 08
Joshua Crotts

Object-Oriented Programming

What To Do:

Follow each step carefully. As you complete the lab, submit the source files (`.java`) problems to the autograder. After finishing, please let one of the AIs know.

For this lab, since we are now working with classes that have meaningful names, please name the files and testers as we specify.

Overview

In this lab, we will be introducing to you object-oriented programming using lines and planes from Calculus III (or multi-variable calculus). Unfortunately, you can't escape math, but don't worry, as most of the code and equations will be provided for you! We understand that most of you have probably only taken pre/calculus. With that being said, I hope that this lab inspires to not only take more math classes but also investigate concepts like three-dimensional graphics and computer vision, which utilize multi-variable calculus. In fact, object-oriented programming applications involving 3-D graphics, computer vision, and virtual reality, multi-variable calculus is the mathematical backbone for modeling and manipulating geometric shapes, surfaces, transformations, projections, and lighting effects. Therefore, understanding concepts like parametric equations, surface integrals, and volume integrals is crucial for developing realistic and immersive visual experiences. However, as stated previously we will only be diving into the basics.

What is a Vector?

The vector is probably the most important concept in applied mathematics. Unfortunately, most lecturers explain this concept poorly. The formal definition of a *vector* is a "quantity having direction as well as magnitude" but what does that even mean? Basically, a vector is an "mathematical object" that points in a specific direction with a specific length (or magnitude). Think of it like this: when you throw a ball, the force you apply to it (the throw) acts as a vector. A vector has both magnitude (how hard you throw it) and direction (the angle at which you throw it). Gravity, on the other hand, is also represented as a vector with direction (vertically downward) and magnitude (force of attraction). To broaden your view even more, in engineering, vectors represent electric and magnetic fields, fluid flow, and structural forces. In machine learning and data science, vectors represent the mean position of all data points in all directions, know as a *centroid*. So, as you can see, vectors are foundational for any field you choose to go down in the world of science and technology.

Problem 1

(a) A vector, denoted as v, is calculated using only two points in space. A *point* is (x, y, z). To calculate a vector, let's define two points $A = (x_1, y_1, z_1)$ and $B = (x_2, y_2, z_2)$. Thus,

$$v = \langle x_2 - x_1, y_2 - y_1, z_2 - z_1 \rangle = \langle a, b, c \rangle$$

Using this logic, design the constructors for both the `Point` and `Vector` classes along with their respective instance variables. The `Vector` class should have a two constructors:

- `Vector(Point A, Point B)` receives two `Point` objects and constructs the vector from those.
- `Vector(double x, double y, double z)` receives the three components x, y, z and directly creates the vector from those.

(b) Design the respective accessor methods in both classes. Also, design the `List<Double> getVectorComponents()` accessor, which returns the vector components as a list.

(c) In the `Vector` class, design the `Vector unitVector()` instance method, which creates a *new* vector that is the corresponding unit vector to `this`. A *unit vector* is like a direction arrow that always has a fixed length of one. Imagine you're standing at a point in space and you want to move in a certain direction. Instead of just pointing in that direction, a unit vector is like a normalized arrow that points in the direction you want to go, but its length is always exactly one unit. To calculate the unit vector we take a vector and divide each of its components a, b, c by the magnitude of the vector, which is represented by $\sqrt{a^2 + b^2 + c^2}$. So the equation of a unit vector is:

$$\hat{\mathbf{u}} = \left\langle \frac{a}{\sqrt{a^2 + b^2 + c^2}}, \frac{b}{\sqrt{a^2 + b^2 + c^2}}, \frac{c}{\sqrt{a^2 + b^2 + c^2}} \right\rangle$$

(d) Design the `double dotProduct(Vector v)` instance method in the `Vector` class, which receives a vector v and computes the dot product between `this` vector and v. The *dot product* is the measure of how closely two vectors align in terms of direction. If the dot product is zero, this means that our two vectors are perpendicular (or orthogonal) to one another. To calculate the dot product, we use the following formula:

$$\mathbf{a} \cdot \mathbf{b} = \langle a_1, a_2, a_3 \rangle \cdot \langle b_1, b_2, b_3 \rangle = a_1 b_1 + a_2 b_2 + a_3 b_3$$

(e) Design the `Vector crossProduct(Vector v)` instance method in the `Vector` class, which receives a vector v and computes the cross product between `this` vector and v. The *cross product* between two vectors is an operation that results in a new vector that is perpendicular to the two original vectors. If the cross product between two vectors is the zero vector (i.e., a vector whose components are all zero), then the two vectors are parallel. If you compute a zero vector, then return the zero vector. As you can see in Figure 1, $\mathbf{a} \times \mathbf{b}$ is perpendicular to **a** and **b**.

$$\mathbf{a} \times \mathbf{b} = \langle a_1, a_2, a_3 \rangle \times \langle b_1, b_2, b_3 \rangle = \langle a_2 b_3 - a_3 b_2,\ a_3 b_1 - a_1 b_3,\ a_1 b_2 - a_2 b_1 \rangle$$

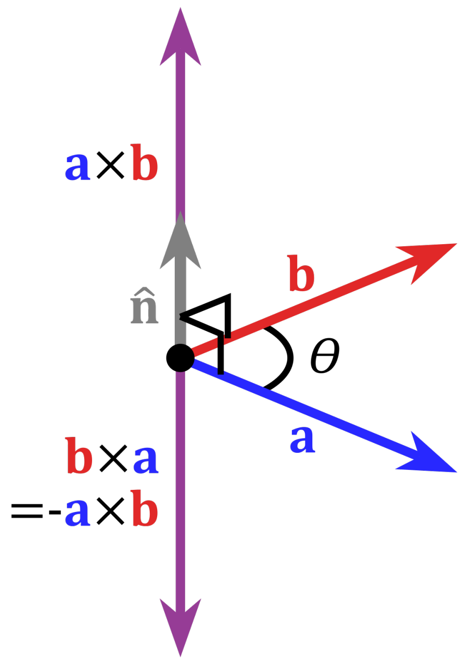

Figure 1: Cross Product

What is a Line?

In mathematics, a *line* is a straight path that extends infinitely in both directions in space. To represent a line in three-dimensional space, we need two pieces of information: a direction vector (which indicates the line's orientation) and a point in space through which the line passes. This point is often referred to as the "initial point" or "starting point" of the line. To represent a line, we use *parametric equations*. Suppose I had a point $A = (x_0, y_0, z_0)$ and vector v. The parametric equations for the line would be:

$$
\begin{aligned}
x &= x_0 + at \\
y &= y_0 + bt \\
z &= z_0 + ct
\end{aligned}
$$

The parameter t represents a point's position along the line and can be thought of as representing a moment in time. By varying the value of t, we can precisely calculate points along the line, represented by the coordinates (x, y, z).

Problem 2

(a) Design the `Line` class, whose constructor receives a `Point` and a direction `Vector`. Store these as respective instance variables in the class.

(b) Using what we now know about parametric equations, design the following instance methods `String paramX()`, `String paramY()`, and `String paramZ()`, which create "parameterized strings."

For example, if I have a line l_1 whose point is $p = (5, 0, -9)$ and whose direction vector is $v = \langle 4, 7, -7 \rangle$, then l_1`.paramX()` would return `"x = 5 + 4t"`, l_1`.paramY()` would return `"y = 7t"`, and l_1`.paramZ()` would return `"z = -9 - 7t"`. Hint: keep track of your "+" and "−" operators. It wouldn't make sense to return a string like `"z = 9 +-4t"`, right? If our vector component is a negative number, then change the operator so it returns `"z = 9 - 4t"`. The spacing matters in this problem!

(c) `Design the double[][] parameterize()` instance method in the `Line` class that converts a `Line` into a 3×2 matrix (array) representing its parametric equations. The method should have the following structure:

$$\begin{bmatrix} x_0 & a \\ y_0 & b \\ z_0 & c \end{bmatrix}$$