PIC 20A: Homework 2 (due 2/1 at 5pm)

Submitting your homework

The zip file you extracted to find this pdf includes a file called `MathVector.java`.
In this assignment, you will edit this file and submit it to Gradescope.

- Upload `MathVector.java` to **Gradescope** before the deadline.
- **Name** the file exactly as just stated.
- Do **not** enclose `MathVector.java` in a folder or zip it.

  Do **not** submit `HW2_Tester.java`.

  You should be submitting exactly **one file** and it should have the **extension** `.java`.
- Be sure that your code **compiles and runs** with `HW2_Tester.java` using **Adoptium's Temurin Version 11 (LTS)**.

Tasks

1. Correct the definitions of the constructors.
   (a) `MathVector(int)` should construct a vector of zeros of the specified dimension.
   (b) `MathVector(double...)` should construct a vector using the specified `doubles`.
   You should make sure to copy the values (to help avoid confusing a user who constructs a `MathVector` using a pre-existing array of doubles).
   (c) `MathVector(MathVector)` should copy a vector.

2. Correct the definitions of `toString` and `print`.
   (a) `String toString()` can use `java.util.Arrays.toString`.
   (b) `void print()` can apply `System.out.println` to the implicit parameter.

3. We have used encapsulation, marking both fields as `private`. Users are not allowed to access the field `dim` directly, but they can ask for its value. Users will also not have direct access to `values`, but they can ask for specific values using `1`-based indexing and update specific values similarly. Correct the definitions of the getters and setters.

   (a) `int getDimension()` should return the value of `dim`.

   (b) `double getValue(int)` should return the value at the index specified.
   In math, people frequently index starting at `1`, not `0` (like in computer science).

   (c) `setValue(int, double)` should update the value at the given index with the value specified. Again, you should use `1`-indexing.

   In part (b) and (c), you do **not** need to worry about what to do when a user indexes out of range. Later on, after learning about exceptions, you could handle this situation better.

4. Correct the definitions of the factory methods `vec2DfromPolar` and `vec3DfromPolar`.

   (a) `vec2DfromPolar` should return (a reference to) a newly created 2-dimensional `MathVector`: r and θ determine the vector $(r\cos\theta,\ r\sin\theta)$.

   (b) `vec3DfromPolar` should return (a reference to) a newly created 3-dimensional `MathVector`: r, θ, and φ determine the vector $(r\sin\theta\cos\varphi,\ r\sin\theta\sin\varphi,\ r\cos\theta)$.

5. Correct the definitions of the math operations.

   (a) The following functions are closely related.

   - `double dotWith(MathVector)` should calculate the dot product...

     $$(x_1, x_2, x_3, \ldots, x_N) \cdot (y_1, y_2, y_3, \ldots, y_N) = x_1y_1 + x_2y_2 + x_3y_3 + \ldots + x_Ny_N$$

     You do **not** need to worry about a user attempting to find the dot product of vectors of different dimensions.

   - `double getMagnitude()` should calculate the magnitude of the vector using Pythagoras...

     $$\Big\|(x_1, x_2, x_3, \ldots, x_N)\Big\| = \sqrt{x_1^2 + x_2^2 + x_3^2 + \ldots + x_N^2}$$

(b) The following functions are the same mathematically, but very different in terms of their impact on the relevant `MathVector`. The scalar product is defined as follows.

$$\lambda \cdot (x_1, x_2, x_3, \ldots, x_N) = (\lambda x_1, \lambda x_2, \lambda x_3, \ldots, \lambda x_N)$$

- `void scaleBy(double)` should change the implicit parameter by multiplying each of its values by the specified double.
- `static MathVector multiply(double, MathVector)` should leave the input vector unchanged and return (a reference to) a newly created `MathVector` whose values are obtained by performing the relevant scalar product.

(c) The following functions are the same mathematically, but very different in terms of their impact on the relevant `MathVector`. The addition of vectors is defined as follows.

$$(x_1, x_2, x_3, \ldots, x_N) + (y_1, y_2, y_3, \ldots, y_N) = (x_1 + y_1, x_2 + y_2, x_3 + y_3, \ldots, x_N + y_N)$$

- `void shiftBy(MathVector)` should change the implicit parameter by adding to each of its values the corresponding value of the other vector.
- `static MathVector add(MathVector, MathVector)` should leave the input vector unchanged and return (a reference to) a newly created `MathVector` whose values are obtained by performing the addition of vectors.

You do **not** need to worry about a user attempting to add vectors of different dimensions.

(d) `boolean equals(MathVector other)` should test for equality between two vectors.
It should allow the comparison of vectors of different dimensions, returning `false` in this case.

(e) `static MathVector add(MathVector...)` should behave like
`static MathVector add(MathVector, MathVector)`
except it should allow the addition of an arbitrary number of vectors.

- It is simplest to start with a vector of zeros, and keep shifting by the vectors you're supposed to be adding.
- The sum of no vectors can be `new MathVector(0)` although I won't test this case.

Comments

- `HW2_Tester.java` provides some essential test cases. You may want to check others.
- The relationship between `scaleBy` and `multiply` is similar to that between `*=` and `*`.
- The relationship between `shiftBy` and `add` is similar to that between `+=` and `+`.
- Those who learned good style choices while operator overloading in PIC 10B will know that `*` is often defined in terms of `*=` and `+` is often defined in terms of `+=`. One can mimic that to some degree here. One cannot write the analogous statements to `return v *= scalar` and `return v1 += v2` because `scaleBy` and `shiftBy` do not return anything. However, you can still follow the protocol of copying `v` and `v1`, respectively, applying the relevant asymmetric method to the copy, and then returning the edited copy.
- In case the math formulae were introduced too briefly...

Vectors in math

An n-dimensional vector sounds scary, but it's surprisingly easy. What is an n-dimensional vector? A list of n numbers.

Examples

1. (1) is a 1-dimensional vector.
2. $(1, 2)$ is a 2-dimensional vector.
3. $(1, 2, 3)$ is a 3-dimensional vector.
4. $(1, 2, 3, 4)$ is a 4-dimensional vector.
5. $(1, 2, 3, 4, 5)$ is a 5-dimensional vector.
6. $(1, 2, 3, \ldots, n)$ is an n-dimensional vector.

In math we index starting at 1, so that 4 is the 4th component of $(1, 2, 3, 4, 5)$ (as opposed to the 3rd component).

Equality

Two vectors of the same dimension are said to be equal if and only if all of their components are equal. $(1, 2, 3)$ is equal to $(1, 2, 3)$, but not equal to $(1, 2, \pi)$ because they differ in the 3rd component.

Adding vectors

One can view the addition of vectors geometrically, but for programming purposes it is more relevant to think algebraically. You add vectors component by component.

$$(x_1, x_2, x_3, \ldots, x_N) + (y_1, y_2, y_3, \ldots, y_N) = (x_1 + y_1, x_2 + y_2, x_3 + y_3, \ldots, x_N + y_N)$$

For example, $(1, 0, 28) + (0, 7, -10) = (1, 7, 18)$.

Multiplying vectors by scalars

One can view the scalar multiplication of vectors geometrically, but for programming purposes it is more relevant to think algebraically. You multiply each component by the scalar.

$$\lambda \cdot (x_1, x_2, x_3, \ldots, x_N) = (\lambda x_1, \lambda x_2, \lambda x_3, \ldots, \lambda x_N)$$

For example, $8(1, 11, 111) = (8, 88, 888)$.

The magnitude of a vector

The magnitude of a vector is its length (from the origin).

(Nerd comment: I put the last part in parentheses because people may agree or disagree about whether it's required due to their preference for $\mathbb{R}^n$ or its tangent space at the origin. Really, it is a happy fluke that there's a natural isomorphsim between $\mathbb{R}^n$ and its tangent space at the origin, and this is not true for other manifolds.

A less nerdy whay of saying the same thing: some peope think of a vector as describing a point; some think of it as describing an arrow; some people drift between the two perspectives.)

A vector's length is calculated using Pythagoras' theorem:

$$\left\| (x_1, x_2, x_3, \ldots, x_N) \right\| = \sqrt{x_1^2 + x_2^2 + x_3^2 + \ldots + x_N^2}.$$

Again, n may be intimidating, but if we give a two-dimensional example, hopefully things feel easier. $\|(3, 4)\| = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5$. The length of $(3, 4)$ is 5, because if you draw an arrow from $(0, 0)$ to $(3, 4)$ in the Cartesian plane, you have the hypotenuse of a 3-4-5 right-angled triangle.

The dot product of two vectors

The dot product of two vectors is calculated as follows.

$$(x_1, x_2, x_3, \ldots, x_N) \cdot (y_1, y_2, y_3, \ldots, y_N) = x_1 y_1 + x_2 y_2 + x_3 y_3 + \ldots + x_N y_N$$

For example, $(3, 4, 0) \cdot (-4, 3, 0) = 3 \cdot (-4) + 4 \cdot 3 + 0 \cdot 0 = -12 + 12 + 0 = 0$. The dot product is very useful. For example, the fact that $(3, 4, 0) \cdot (-4, 3, 0) = 0$ tells us that $(3, 4, 0)$ and $(-4, 3, 0)$ are perpendicular to one another.