Due Saturday by 11:59pm **Points** 2 **Available** after Sep 18 at 6am

Module 3 — Representing More Complex Forms of Data

Home

Modules

Lab 2: Methods, Exceptions, Equality & Class vs. Instance

Position and Velocity

1.1 Introduction

The purpose of this lab is to give you practice with the concepts of methods (design, overloading, etc.), exception handling, equality, and instance features vs. class features (e.g. instance methods vs. class methods).

1.2 What to do

Remember your physics classes in high school? For this lab, you will design and implement the basic elements for us to track and compute simple displacement vectors in 3-D space. Then, using classical mechanics we'll create a specific service that provides a class method to calculate (average) velocity, given the displacement and a time value.

In case you need it, our handy distance equation
(from varsity tutors at **https://www.varsitytutors.com/hotmath/hotmath_help/topics/distance-formula-in-3d (https://www.varsitytutors.com/hotmath/hotmath_help/topics/distance-formula-in-3d)**) is:

> In three-dimensional Cartesian space, points have three coordinates each. To find the distance between $A\ (x_1, y_1, z_1)$ and $B\ (x_2, y_2, z_2)$, use the formula:
>
> $$AB = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Important Note: All of the work done in this lab should be placed in a package called distance.

Create a class called **`Point3D`** that represents a point in 3-D cartesian space. You may name your instance variables whatever you want, but the protocol/interface for Point3D must include:

- A no-parameter constructor (aka the "default" constructor)
- A constructor that take 3 integers, representing x, y, z in that order

- Three "getter" methods called: `getX, getY, getZ`

- A method that calculates and answers (returns) the distance between the current point and another point. Consider the "other" point is the destination (or "B" in the equation above). It should have the signature:
  `double distanceTo(Point3D other)`

- An `equals` method that allows us to compare a `Point3D` instance to another `Point3D` object. Note that you should typically write a hashCode method since you're providing equals, but we will not be testing that and you will not lose points on this lab if you don't provide one.

Write JUnit tests to validate and verify your `Point3D` class. After you've done that:

Create a class called `Physics`. This class is a "utility service" class and provides a single class method (not instance method) that calculates a unitless velocity given two `Point3D` objects and an elapsed time. Average velocity is the change in distance over change in time. Consider that our point instances know how to calculate the distance for us and if we supply a time delta (called elapsedTime), we can calculate average velocity.
The signature for this class method is:
`double velocity(Point3D one, Point3D two, double elapsedTime)`

`velocity` throws an `IllegalArgumentException` if the time value provided is zero or negative.

Write appropriate JUnit tests to validate the `Physics` utility class.

After developing and testing your `Physics` class, implement a static main method (it can be in a separate class if you wish, or in the `Physics` class - your choice). Enter the code shown below and run it. Ensure the code you developed runs without issue. Include a plaintext file in your lab submission and describe the output on the console when you run the code.

```
public static void main(String [] args) {
  try {
    Point3D one = new Point3D();
    Point3D two = new Point3D( x: 1,  y: 1,  z: 1);
    System.out.println("Displacement = " + one.distanceTo(two));
    double velocity = Physics.velocity(one, two,  elapsedTime: 5);
    System.out.println("Prof. Keith is on the move! His Velocity =" + velocity);
    velocity = Physics.velocity(one, two,  elapsedTime: 0);
    System.out.println("Velocity =" + velocity);
  }
  catch(IllegalArgumentException e) {
    System.out.println("Encountered an error: " + e.getMessage());
  }
}
```

Next, swap the **`elapsedTime`** values in the method calls such that 5 becomes 0 and the 0 becomes 5. Run the code again. Describe the output in your text file submission.

Finally, create a UML class diagram that describes the system you constructed.

To Turn In

As usual, Gradesope is the place to turn in your work. Submit your zip containing your src and test directories (remember your code assets should be in the distance package), your UML diagram (pdf, png, jpg) and your plaintext file describing what you observed when you ran the main() method.

Lab 2 Rubric

Full Credit - Pass	Partial Credit - Pass	No Credit
2 pts	**1 pt**	**0 pt**
Point3D and Physics Solution passes all automated tests. Good OO theme. Exception handling implemented, documented (including UML) and reasonable number of tests provided.	Solution passes most automated tests. Good OO theme. Exception handling implemented. Some documentation (including UML) and/or tests partially implemented. Misunderstanding of class vs. instance methods.	Solution passes few tests, problems with OO themes. Missing documentation, missing tests. Or assignment not submitted