COS126 Princeton University Programming Exam 2 Fall '23

Before the exam. Read this page of instructions before the exam begins. Do not start the exam (or read the next page) until instructed to do so.

Duration. Once the exam begins, you have **75** minutes to complete it.

Submission. Submit your solutions on TigerFile using the link from the Exams page. You may submit multiple times (but only the last version will be graded).

Check Submitted Files. You may click the ***Check Submitted Files*** button to receive partial feedback on your submission. We will attempt to provide this feature during the exam, but you should not rely upon it.

Grading. Your program will be graded *primarily* on correctness. However, efficiency and clarity will also be considered. You will receive partial credit for a program that implements some of the required functionality. You will receive a substantial penalty for a program that does not compile.

Allowed resources. During the exam you may use **only** the following resources: course textbook, companion booksite, lectures, course website (which includes past exams and solutions), course Ed Discussion and Ed Lessons, your course notes, jshell / jshell-introcs, and your code from the programming assignments and precepts. You may not use outside sources such as StackOverflow, Google, ChatGPT, etc.

No collaboration or communication. Collaboration and communication during this exam are prohibited, except with course staff. A staff member will be outside the exam room to answer clarification questions.

No electronic devices or software. Software and computational/communication devices (phones, ipads, etc.) are prohibited, except to the extent needed for taking this exam (such as a laptop, browser, and IntelliJ). For example, you must close all unnecessary applications and browser tabs; disable notifications; and turn off your cell phone.

Honor Code pledge. Write and sign the Honor Code pledge by typing the text below in the file **`acknowledgments.txt`**. Submit to TigerFile.

> *I pledge my honor that I have not violated the Honor Code during this examination.*
>
> Electronically sign it by typing **`/s/`** followed by your name.

After the exam. Discussing or communicating the contents of this exam before solutions have been posted is a violation of the Honor Code.

Deliverables. Write and submit one Java program, **`Pedometer.java`**, along with an **`acknowledgments.txt`** file. Submit both files to TigerFile.

Overview. A pedometer is a device worn by a person that keeps track of its current position and the total distance traveled as the person moves. In this problem, a person may only move east, west, north or south in a two-dimensional (x, y) grid. Write a *mutable* abstract data type **`Pedometer.java`**:

Part I (90 points). Implement the following API:

```
public class Pedometer
  // creates a Pedometer at coordinate (cx, cy) where each unit represents 1 meter
  public Pedometer(int cx, int cy)

  // returns a String representation of this Pedometer that includes the
  // current position and the distance traveled in kilometers and meters
  public String toString()

  // returns the total distance in meters traveled by this Pedometer
  public int distanceTraveled()

  // is the distance traveled by this Pedometer less than
  // the distance traveled by that Pedometer
  public boolean lessThan(Pedometer that)

  // moves the Pedometer to a new position (nx, ny)
  public void moveTo(int nx, int ny)

  // moves the Pedometer: east -'E', west -'W', north -'N', or south -'S' by
  // the given number of meters m (m >= 0) and updates the new position
  public void move(char dir, int m)

  // directly tests all instance methods in this class
  public static void main(String[] args)
```

Here is some more information about the required behavior:

- *Two-argument constructor.* The initial distance traveled is zero (0). Units are in meters.

- *String representation.* The format of a **`Pedometer`** includes the current position and total distance:

  `current:(#,#)/distance:#km,#m`

  where **`#`** is an integer value, **`km`** represents kilometers (1000 meters / kilometer) and **`m`** represents meters. The `String` returned must not contain any spaces. For example, if the current position is (6,7) and total distance traveled is 13 meters, then **`toString()`** returns:

  `current:(6,7)/distance:0km,13m`

  If the current position is (-51,10) and total distance traveled is 2,121 meters, then **`toString()`** returns:

  `current:(-51,10)/distance:2km,121m`

- *moveTo(int nx, int ny).* Updates the current position (cx, cy) and total distance traveled by moving to position (nx, ny). The distance for a move is calculated using the *Manhattan or taxicab distance* metric:

  `|nx - cx| + |ny - cy|` meters (where **`|...|`** means *absolute value*)

  The absolute value is used since coordinates may be positive or negative, but distance is always positive.

For example, if a **Pedometer** object is currently positioned at (1, 2). Then the call to **moveTo(-2, 4)** corresponds to a distance of

|-2 - 1| + |4 - 2| = 5 meters for this move, and added to the total distance traveled.

The updated current position of the **Pedometer** object is now (-2, 4).

- *move(char dir, int m).* If the value for **dir** is anything but the capital character **'E','W','N','S'** then directly raise an **IllegalArgumentException**. If **m < 0**, raise an **IllegalArgumentException**.

  - **'E' -** move east **m** meters: increase the current x coordinate by **m** meters.
  - **'W' -** move west **m** meters: decrease the current x coordinate by **m** meters.
  - **'N' -** move north **m** meters: increase the current y coordinate by **m** meters.
  - **'S' -** move south **m** meters: decrease the current y coordinate by **m** meters.
  - The distance traveled for a move is: **m** meters.
  - The **Pedometer**'s current position and total distance traveled must be updated.

- *Test client.* The **main()** method must call each instance method directly and must also help verify that each method works as prescribed (e.g., by printing results to standard output). Hint: use the `toString()` and `distanceTraveled()` methods to test your `moveTo(...) and move(...)` methods.

- *API*. You may define private helper methods, but you may not modify the existing public API. Ignore the API message if you do not implement the constructor specified in Part II.

- **Make sure your solution compiles!**

Part II (10 points). Do not attempt this part until you have successfully completed Part I. **Again, your entire solution must compile**!

Add the following three-argument constructor to your **Pedometer** class. You should not have to modify the constructor and methods you wrote in Part I to implement the constructor in Part II, except for the **main()** client, which must include test cases for this part.

```
// creates a Pedometer starting at position (cx, cy) and then moves the Pedometer
// using each direction/meters pair in the moves queue; assume moves != null
public Pedometer(int cx, int cy, Queue<String> moves)
```

The argument **moves** is a queue of **String** values, where each **String** contains two (2) characters: a single direction character, either **E, W, N, S** followed by a single digit, either **0, 1, 2, 3, 4, 5, 6, 7, 8, 9**. No spaces are allowed. Examples of valid **String** values:

- **"W1"** - move west one (1) meter
- **"E9"** - move east nine (9) meters

For example, suppose a queue contains the following valid `String` values:

"E1" "S1" "E3" "W2"

This corresponds to the following moves: **move('E',1); move('S',1); move('E',3); move('W',2);**

This constructor must directly throw an **IllegalArgumentException** if any **String** value in the **moves** queue is invalid. Examples of **String** values that must throw an **IllegalArgumentException**:

"W" "W 9" "W9E1" "n4" "S100" "9 W" ""