CS101-004– Practice Problems
Intro to Computer Science

November 5, 2024

---

1. You are given a $n \times n$ matrix of integers. As an example, we will use

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}.$$

Your methods have to work for any $n \times n$ matrix where $n \geq 1$. Write a method that

- Iterates through the matrix in a snake-like motion and fills an array: The first row is traversed from left to right, the second row is traversed from right to left, the third row is traversed from left to right, and so on. Example output:

$$[1, 2, 3, 4, 8, 7, 6, 5, 9, 10, 11, 12, 16, 15, 14, 13].$$

- Iterates through the matrix in a spiral motion: Go left to right along the first row, then down the last column, then right to left along the last row, and so on. No entry should be printed more than once. Again, return an array with the entries. Example output:

$$[1, 2, 3, 4, 8, 12, 16, 15, 14, 13, 9, 5, 6, 7, 11, 10].$$

- Stacks the columns of the matrix and returns the resulting array. Example output:

$$[1, 5, 9, 13, 2, 6, 10, 14, 3, 7, 11, 15, 4, 8, 12, 16].$$

- Rotates the matrix by 90 degrees clockwise. Do this in-place; do not allocate another matrix. Example output:

$$\begin{bmatrix} 13 & 9 & 5 & 1 \\ 14 & 10 & 6 & 2 \\ 15 & 11 & 7 & 3 \\ 16 & 12 & 8 & 4 \end{bmatrix}.$$

2. The following is an exam question from a previous semester:

   Simulating interacting particles is a common problem in the sciences. The particles can be elementary particles (atoms, electrons, ...), planets, or even people.

   In this problem, we will construct a class that represents particles in one dimension.

   By default, all fields should be private.

```
public class Particle {
```

   (a) (2 points) A particle has a position in space and a velocity. Both are `doubles`. Add fields `x` and `v` to the class to store them.

   (b) (2 points) Add a field `numberOfParticles` to the class that stores the number of *all* particles that have been constructed and initialize it to zero.

   (c) (2 points) Implement a public constructor that takes the position and velocity as arguments. The constructor also has to make sure that `numberOfParticles` stays up to date.

   (d) (2 points) Implement a public getter for `numberOfParticles`. To get full points, you have to access `numberOfParticles` in the a way that does not cause a warning.

(e) (1 point) Implement a public getter for `x`.

(f) (2 points) Implement a public method `double getDistance(Particle other)` that calculates the distance between two particles. The distance between two particles is given by $r = \sqrt{(x_1 - x_2)^2}$. You can use the `Math.sqrt()` method to calculate the square root. Example usage: `double sqrtX = Math.sqrt(x);`

```
} // end of class Particle
```

(g) (1 point) Implement a (public) class `ChargedParticle` which is a subclass of `Particle`. Keep the body of the class open (i.e. do not close the braces) so that you can add things to it like you did for the `Particle` class.

(h) (2 points) A charged particle has an additional field `charge` which is an `int`. Add it and name it `q`. Write a public getter for it called `getCharge()`.

(i) (1 point) Add a field `numberOfChargedParticles` to the class that stores the number of *all* charged particles that have been constructed and initialize it to zero.

(j) (3 points) Implement a constructor for a charged particle (arguments: position `x`, velocity `v`, charge `q`). Make sure that `numberOfParticles` and `numberOfChargedParticles` stay up to date. You get full points here only if you do not write duplicate code.

```
} // end of class ChargedParticle
```

(k) (1 point) If a method in `ChargedParticle` requires access to `numberOfParticles` (a private field of `Particle`), what do you need to change? You do not want to make `numberOfParticles` public.

Assume now you are given an array of `Particle`s called `particleArray`. Some particles are charged, some are not.

(l) (5 points) Loop through the array and compute the total charge. The total charge is the sum of the charges of all charged particles. Remember that the array contains also `Particle`s that have no charge. For example, if the array contains three particles, two of which are charged with charges 2 and -1, the total charge is $2 + (-1) = 1$. Complete the method:

```
static int totalCharge(Particle[] particleArray) {
```

```
} // end of method totalCharge
```

(m) (2 points) Someone complains that the `totalCharge` method is way too complicated. They would like to simply loop through all particles, call `getCharge()` on each particle, and add the results together. What do you need to change in the code you wrote so far to make this possible? Again, choose the least disruptive way. If you need to add code, write it (in the correct form) and say in which class it has to be.

(n) (2 points) An electron is a charged particle with mass 1 and charge -1. What is the most natural way to add a class `Electron` to our package? (So far, `Particle` and `ChargedParticle` are in the package.) Implement the class including a constructor that takes no arguments and puts the electron at position 0 with velocity 0. This constructor has to lead to the correct values of `numberOfParticles` and `numberOfChargedParticles`.

(o) (5 points) Given the array of particles `particleArray`, write code that builds an array with only the charged ones. You may assume that you know the number of charged particles in advance.
Complete the following method:

```java
static ChargedParticle[] getChargedParticles(Particle[] particleArray,
    int numChargedParticles) {
```

```java
} // end of method getChargedParticles
```