New York University
Department of Computer Science

CS101-004– Practice Problems
Intro to Computer Science

Instructor: Tobias Blickhan

-

Name: _____
Net ID: _____

---

1. Recall the `Particles` class from Midterm 2:

```
1  class Particle {
2      private double x;
3      private double v;
4      public Particle(double x, double v) {
5          this.x = x;
6          this.v = v;
7      }
8      public double getPosition() {
9          return x;
10     }
11     public double getVelocity() {
12         return v;
13     }
14     public void move(double dt) {
15         x += v * dt;
16     }
17     public void setVelocity(double v) {
18         this.v = v;
19     }
20     public double getDistance(Particle other) {
21         return Math.abs(x - other.getPosition());
22     }
23 }
```

   (a) Write a class `Simulation` that holds an array of particles and a double `dt`.

   (b) Add a constructor that takes an array of particles and a double `dt` and initializes the fields.

   (c) Add a method `void updatePositions()` that moves all particles in the array by `dt`. If a particle ends up at a position < 0 or > 1, reverse its direction ($v \leftarrow -v$).

   (d) Add a method `void updateVelocities()` that updates the velocities of all particles in the array. The only thing that changes the particles' velocities is a collision with another particle. If two particles are closer than 0.01, they collide. In this case, the velocity of the first particle is set to the velocity of the second particle and vice versa.
   Hint: Make sure you do not double-count particles. If particle i collides with particle j, do not have particle j collide with i in the same iteration.

   (e) Add a method `sortParticles()` that sorts the particles by their position (ascending). You do not have to use a fancy sorting algorithm, just use one you know. (If you are curious on how you could use `Arrays.sort()` to do this, look it up! It is good to know but not part of the exam.)

   (f) (IDE) Write a main method that creates an array of 10 particles with random positions and velocities (use `Math.random()`), using `dt` $= 0.05$. Run a simulation for 100 iterations. In each iteration, update the positions, update the velocities, sort the particles, and print the positions of all particles. There is a "pretty" printing function in the solution code you can use.

(g) (Bonus, this might take some time) Instead of the collisions, we can add charges -1 to all particles and have them repell each other. The force between two particles is given by Coulomb's law: $F = \frac{\pm 0.01}{r^2}$. The velocity of the particle is then updated by $v \leftarrow v + \frac{1}{2} F \cdot dt$ (the factor $\frac{1}{2}$ is again to not double-count).

2. You are given the following abstract class:

```
24 public abstract class RandomNumberGenerator {
25     protected int state;
26     public RandomNumberGenerator(int seed) {
27         state = seed;
28     }
29     public abstract int nextInt();
30     public abstract double nextDouble(); // returns a number in [0, 1)
31     public abstract boolean nextBoolean();
32 }
```

(a) One way to generate pseudo-random numbers is to perform multiplication and modulo operations. Implement a subclass `MLCG` of `RandomNumberGenerator` that uses the following formula to generate the next random number integer:

$$x_{n+1} = (a \cdot x_n) \mod m, \tag{1}$$

where $a = 2^{31} - 1$ and $m = 16807$. x_0 is the seed value passed to the constructor. Use `Math.pow(2, 31)` to calculate a but remember this returns a `double`.

- All `MLCG` objects should share the same a and m values and their value should not change.
- The state of the random number generator should change such that you can repeatedly call `nextInt()` to get a new random number. The state at iteration n is x_n.
- You need to write three methods and a constructor, the latter takes a seed (int) and relies on the constructor of the superclass.

Hint: What is the largest value that x_{n+1} can take?

(b) The seed passed to the constructor should be positive and strictly smaller than m. Re-write the constructor so that it throws an `IllegalArgumentException` if it is not.

(c) Your random number generator works fine, but after a while, it starts returning negative numbers, even though $(a \cdot x_n) \mod m$ should always be positive. What is going on?

(d) Assume now that this issue is fixed. Next, write a method that takes any (!) RandomNumberGenerator and an integer n. (This method would be in a separate test class.) The method then generates n random numbers between 0 and 1. While doing so, it counts the number of values between 0 and 0.1, 0.1 and 0.2, and so on. There are ten such "bins". Return an array of length 10 with the counts.

(e) (Recommended to do in your IDE) Implement another RandomNumberGenerator subclass called `VonNeumannRNG` that uses the middle-square method: Given a seed (a 4-digit integer), the next random number is generated by squaring the input and writing the

result as a 8-digit integer. If the squared input does not have 8 digits, we pad it with zeros from the start. The next random number is then given by the middle 4 integers in the resulting number. This number is printed and then used used as input for the next iteration. For illustration, here is the whole workflow on an example:

1. We start with a seed, say 1234
2. Square it, giving 1522756
3. Pad with zeros, giving 01522756
4. Take the middle four digits, giving 5227
5. Print 5227 and use it as new input
6. Square it, giving 27321529
7. Padding not needed
8. Take the middle four digits, giving 3215
9. Print 3215 and use it as new input
10. Square it, giving 10336225
11. Padding not needed
12. Take the middle four digits, giving 3362
13. Print 3362 and use it as new input
14. ...

The `VonNeumannRNG` should extend `RandomNumberGenerator`. You can and should make a bunch of (private) helper methods here to handle the padding, middle digit extraction, etc. If the seed is too small, pad it with zeros. If the seed is too large, you can take the last four digits. Hint: $1234/1 \mod 10 = 4$, $1234/10 \mod 10 = 3$, $1234/100 \mod 10 = 2$, ..., $1234/10000 \mod 10 = 0$, where "/" is integer division. Note: This RNG has a tendency to get stuck at outputting zeros, you can add a check for this if you want.

(f) (Needs to be done in your IDE) Write a test class that creates an array of length two holding `RandomNumberGenerator` objects, one for each subclass. Then, for each of the objects, generate 10000 random numbers between 0 and 1 and compare the bin counts (defined in part (d)) for the two generators.

(g) (Bonus) We can compare our generators to Java's `Random`, since we can also pass a seed to it. However, `Random` is not an instance of `RandomNumberGenerator`. Write a Wrapper class that extends `RandomNumberGenerator` and holds a `Random` object. You do not have to bother updating the `state` yourself now, since the wrapped `Random` object has its own (an interface would be better suited here).

(h) (Bonus) Take a look at the `next` method of `Random`:
https://docs.oracle.com/javase/8/docs/api/java/util/Random.html.
Write a subclass of `Random` and override the `next` method to return a fixed value, say 42. Then, create an instance of your subclass and call `nextInt()`, `nextDouble()`, `nextBoolean()` on it a few times.

3. Assume we are given an array of data points called `dataPoints`. Each data point itself is a `double[d]` of length d and there are n data points in the array. In other words, `dataPoints`

is of type `double[n][d]` and can be thought of as a data matrix with n rows and d columns. Making a sketch might help.

(a) How do you access the i-th data point from the array? Write the code for it. You do not have to assign it to anything.

(b) What is the fastest way to loop through all elements of the array? Why?
You do not have to write code. Making a sketch might help to explain.

(c) In machine learning, *batching* is a common procedure. Batching is the process of splitting a dataset into smaller parts, called *batches*, and processing these batches one by one.
Complete the method given below that takes the array of data points and an integer `batchSize` as arguments and returns a random subset of the data (i.e. elements of `dataPoints`, or rows of the data matrix). The subset should have size `batchSize`.
You can add the same data point multiple times to the batch.
Hint: You might want to read the next question before answering this one.

```
1  // assume there this method is in the main method of a class and
       import java.util.Random has been done
2
3  static Random r = new Random();
4  // reminder: get random integer between 0 and n-1 with r.nextInt(n)
5
6  static double[][] getBatch(double[][] dataPoints, int batchSize) {
7
8      // your code here
9
10 } // end of method getBatch
```

(d) (2 points) There are actually two ways to solve the preceding question:
(1): copy the data point values themselves.
(2): copy references.
Which one did you implement? How does one implement the other one? You do not have to write pure code, but be precise.

(e) What would be a reason to prefer method (2) over (1)? You do not have to write a lot.

(f) Implement a method that takes the array of data points and two integers `numBatches` and `batchSize` and returns an array of batches. Each batch should have size `batchSize` and there should be `numBatches` of them.
Make sure that you do not unnecessarily copy data points. Use the method `getBatch` you implemented already.

4. (a) Write an abstract class `Shape` with abstract methods `double getArea()` and `double getPerimeter()`.
For all of the following subclasses, add a constructor and the methods `getArea()` and `getPerimeter()` if necessary.

(b) Write a subclass `Circle` that extends `Shape` and has a field `double radius`. You can use `Math.PI` for π.

(c) Write a subclass `Rectangle` that extends `Shape` and has fields `double width` and `double height`.

(d) Write a subclass `Square` that extends `Rectangle`. Add a constructor that only takes one argument, the side length.

(e) Write a subclass `Triangle` that extends `Shape` and has fields `double side1`, `double side2`, and `double side3`. You can use Heron's formula for the area: For sides a, b, c, the area is $\sqrt{s(s-a)(s-b)(s-c)}$ where s is half the perimeter. Square roots can be computed with `Math.sqrt()`.

(f) Add a subclass `RightTriangle` that extends `Triangle` and has a constructor that takes two arguments, the two shorter sides. The length of the third side can be calculated with the Pythagorean theorem $a^2 + b^2 = c^2$. Override `getArea()` using the simpler formula $A = \frac{1}{2}ab$. Is this possible when `side1, side2, side3` are private?

(g) Add a subclass `EquilateralTriangle` that extends `Triangle` and has a constructor that only takes one argument, the side length. Again, use the fields of the superclass.

5. Add an additional character to the videogame from the example code at (`src/abstractions/videogame`).

6. Implement an in-place version of the `transpose` method in `src/abstractions/abstract_matrices/Matrix.java`, Line 28, such that it operates on the matrix itself. This method should throw an Exception when called for a non-square Matrix.

7. Check the code in `src/inheritance/numbers` from the example code. How would you re-write this in a way that utilizes abstract classes? You can compare your idea to the way Java implements the Numbers classes:
https://docs.oracle.com/javase/tutorial/java/data/numberclasses.html