NAZARBAYEV UNIVERSITY
SCHOOL OF ENGINEERING AND DIGITAL SCIENCES

CSCI 235 – Programming Languages

Fall 2019 – Project

NU Graduation

Asst. Prof. Selim Temizer

Instructions

1. This homework project consists of ***1 common programming task for all CSCI 235 students***.
2. You will have ***about 3 weeks*** to ***work*** on the task and to ***submit*** your solution through ***Moodle***.
3. This is an ***individual exercise***, so please do ***not*** share any code with other students. We will check your solutions using ***software comparison systems***, and we will not tolerate any cases of cheating.
4. Your solution will consist of ***multiple Java files***. ***Don't*** use ***packages*** in your solution. Make sure that you have a ***public runnable*** class called ***NURunner***, and your solution should run without any errors when we type the following commands:

   Prompt> **javac NURunner.java**

   Prompt> **java NURunner**
5. When you are finished with writing and testing your code, submit ***only*** your ***Java*** files (no bytecode, no binaries, no IDE-generated files, no other files; just Java files) through ***Moodle***. You may ***zip*** all your Java files first (with the name **Solution.zip**), and then submit this single file.
6. Your solution should ***strictly obey all specifications*** below to get ***full grade*** (30 points).
7. ***Good luck***!

Notes

1. Please ***read*** all instructions, notes, specifications, etc. ***very carefully***.
2. We ***suggest*** all students to use ***lab computers*** (in ***Linux*** mode). Lab computers have nice standard configurations. Your own laptops have all very different configurations, and we will ***not*** be able to help you with ***Java***/***IDE***/***Other*** problems on your laptops.
3. You are supposed to know all the basic stuff like what a ***file***, ***editor***, ***terminal***, ***etc.*** is. We ***cannot*** help you with topics covered in your 1st year curriculum.
4. We will ***not*** answer questions like "***How can I create an object?***" or "***How can I call a method?***", or questions on any other topic covered in the class.
5. Make sure that your solutions ***can*** run on ***lab computers*** in ***Linux*** mode without any problems. When grading your solutions, we will be using the same Java distribution as on the lab computers.
6. ***Don't forget*** to submit your solution ***on time***. Moodle system will have a strict deadline time for submissions, and you should ***not*** miss it. We will grade ***only*** the solutions successfully submitted through ***Moodle***.

NU Graduation

In this project, we will build an animation about Nazarbayev University (NU) students collecting grades to finish the school, and having a graduation ceremony in the end.

We will use object-oriented design principles when building the animation. Basically, we will need the following classes to implement our application:

- ***Animation Entities***: These entities consist of a map of NU, academicians, assessment (grading) items, students and important people invited to deliver exciting speeches at the graduation ceremony. Some entities are stationary and some are mobile. Most animation entities will have various strings (like names, and other related data) displayed aside.
    - ***University Map***: A stationary map of NU that is displayed as the background image inside the window.
    - ***Academicians***: Academicians have 3 different motion types: an academician can **rest** for a while, or can move in a **zigzag** fashion (moving in a random direction, and bouncing off the boundaries of the window), or can pick a random target location within the university boundaries and directly **goto** that target. If an academician is not resting, s/he prepares random assessment items (labs, homeworks and quizzes) and scatters them around for the students.
    - ***Assessment Items***: Assessment items are created and scattered around by academicians. Once they are created, they stay where they are placed (stationary). There are 3 types of assessment items: **labs** have random grades between 2-4 and are denoted with red circles, **homeworks** have random grades between 1-3 and are denoted with green rectangles, and **quizzes** have random grades between 3-5 and are denoted with blue triangles. If a student comes close enough to an assessment item, the student picks it up, and earns its grade.
    - ***Students***: Students have 4 different motion types: **rest**, **zigzag** and **goto** behaviors are similar to the academicians. The 4th motion type is called **closest**, and when the student is in the closest state, s/he moves directly to the closest assessment item, picks it up, and keeps moving to the next closest assessment item. If a student collects 100 or more points worth of grades, s/he goes to the summer amphitheater (Stella) and waits for the graduation ceremony.
    - ***Speakers***: Speakers become visible around the summer amphitheater when all students graduate (collect 100+ points). Speakers are very important people invited to the graduation ceremony to deliver exciting speeches. Speakers are stationary (they don't move).
- ***Common***: Brings together animation parameters, instances of entities, and other utility fields / methods that are required for running the application (you may think of this class as a storage for all your global data items for the animation).
- ***Display***: Extends *JPanel* and enables us to draw images and 2-dimensional graphics on it.
- ***Vector2D***: A utility class for representing entity positions and containing various utility methods that manipulate position data.
- ***NURunner***: A class that contains the main method.

However, we will also use the following ***software design patterns*** that will require us to extend our basic design:

- ***Abstract Factory***: Specific factories will create different types of assessment items.
- ***State***: As a minimum, 5 types of states should be prepared: *Stationary*, *Rest*, *ZigZag*, *GotoXY*, and *Closest*. At any given time during the animation, each animation entity will be in a state picked from this list, and act accordingly. You may also prepare an *Invisible* state to hide the

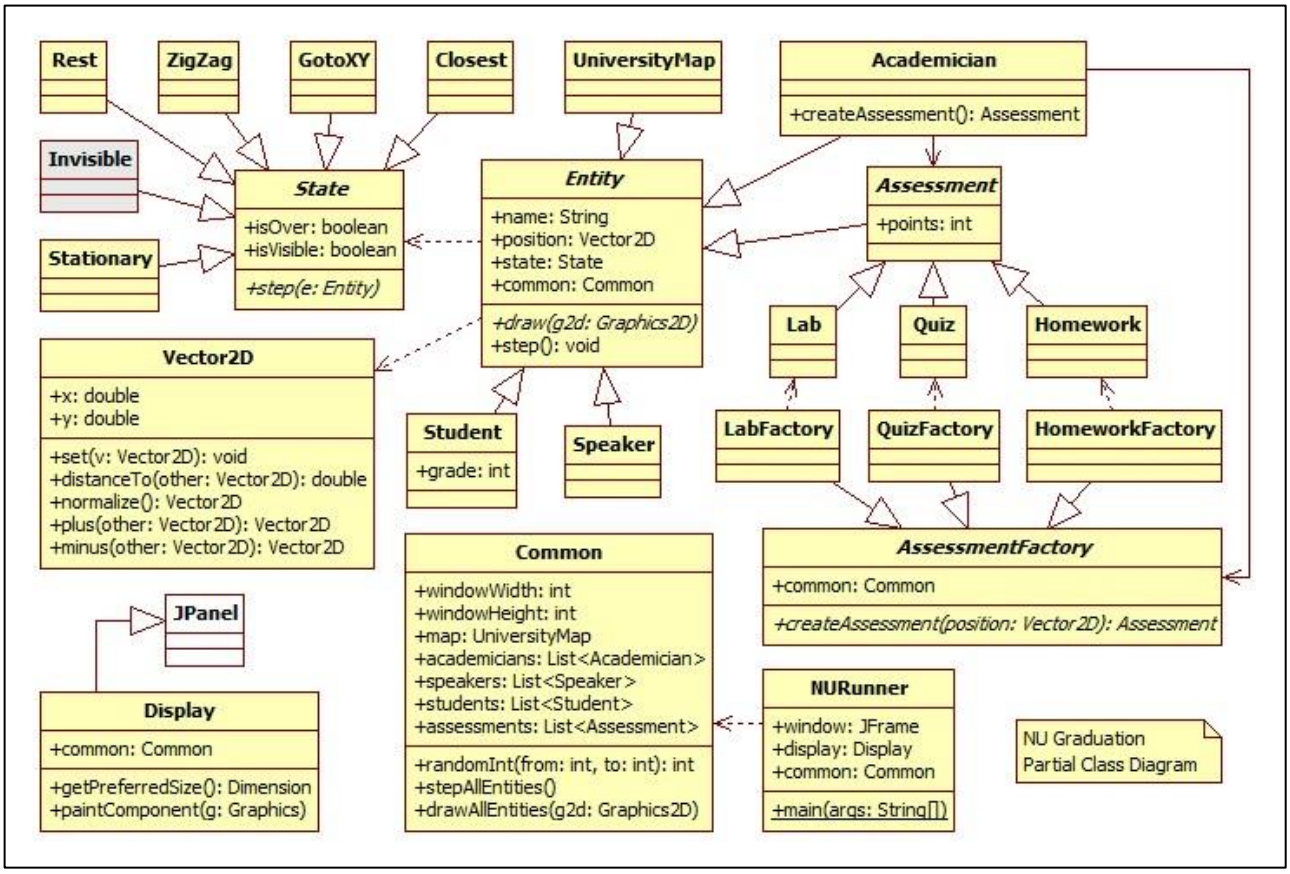
speakers until the graduation ceremony (I did not need to use an *Invisible* state in my implementation).

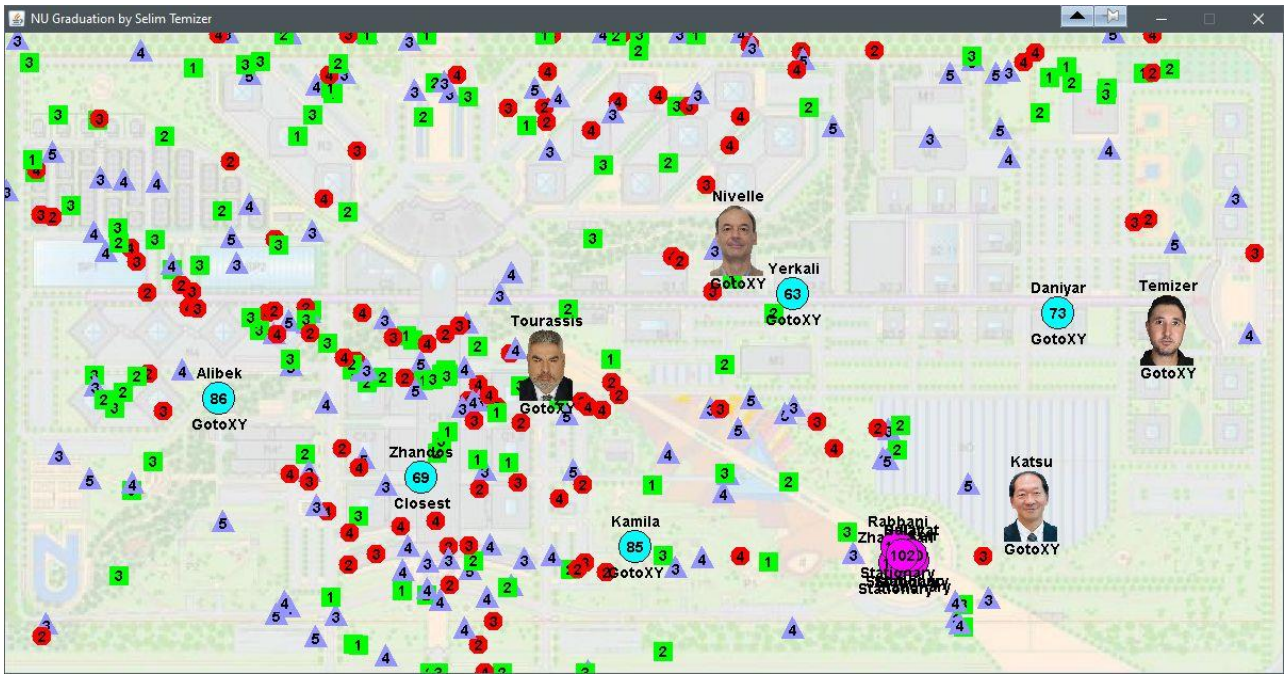
Using the above design patterns, we should be able to:

- Prepare an environment with some predefined number of various properly initialized entities. As the animation proceeds, some animation entities will change their states as described above. When grading, we need to clearly see the changing states during the flow of the application.
- As a suggestion, ideally a *stepAllEntities* method in the *Common* class might ask each entity to act for one step, then might check which entities interact with each other. This *stepAllEntities* method might be called over and over (with a screen refresh and sleeping for a few milliseconds in between) to create the animation.

The following artifacts are provided in this homework bundle:

1. A detailed UML class diagram describing important aspects of my sample implementation. You may use this diagram as a guideline while working on your solutions.
2. A screenshot of my sample implementation which shows students collecting grades to graduate.
3. Another screenshot that shows the graduation ceremony at the end of the animation.
4. My sample implementation in the form of a compiled Java archive (a runnable file with the extension "jar"). You may run this sample implementation to observe how the animation should work in general. Note that I used Java Development Kit version 8 when compiling my solution (so, your Java Runtime Environment should be at least version 8 to run my implementation).





Don't forget to submit your solution ***on time***. Moodle system will have a strict deadline time for submissions, and you should ***not*** miss it. We will grade ***only*** the solutions successfully submitted through ***Moodle***.
