OBJECT-ORIENTED PROGRAMMING

JAVA PROJECT

In our commitment to keep you safe, you will be creating a shopping simulator instead of shopping for real.

PROJECT DESCRIPTION:

The goal of the project is to create a simulator of the retail world, with a pandemic twist. The main entities in this world are clients, suppliers, shops, and products.

FOR MORE QUESTIONS OR CONCERNS, KEEP READING THE FOLLOWING PAGES.

A product should have a name, brand, best before date, and a unique identifier.

Products are sold in shops. Each shop is characterized by its name, address, max client capacity, storage capacity, current supply (available products). Shops can be divided into wholesale stores and retail shops. Wholesale stores have the option of creating produce (filling up the warehouse), whereas retail stores do not have this capability. Periodically, shops check the best before dates of their products and offer sales for almost expired products and remove expired products.

Products are delivered from wholesale stores to retail shops by suppliers. A supplier is described by a unique identifier, name of the company he/she works for, car brand, trunk capacity (measured in products). Suppliers travel on roads by following predefined delivery routes (lists of stops). While traveling, suppliers burn gas proportionally to the road traveled. At each shop (route stop) suppliers fill up the tanks of their cars and exchange products – they leave products at retail shops and get new ones from warehouses.

Clients are characterized by their first name, last name, identifier (e.g., Driver's License or SSN), cart capacity (max number of products she/he can carry), current position on the map, and next stop (shop). Clients travel on sidewalks paved between retail shops, and randomly choose their next stop after visiting each shop. In each shop, a client buys a random number of products, but no more than her/his cart's max capacity. If a client bought more products than can currently fit into his cart, she/he consumes (removes from the cart) as many products in the cart as is required to fit his new shopping load.

All this is happening during a pandemic. A person (client or supplier) can be sick and potentially transmit the disease. Depending on whether a person wears a mask or not, there is a different chance of transmitting the disease to people whom he/she meets. When a user-defined percentage of people are sick, the number of people that can be inside a shop is limited to 25% of their max capacity (lockdown). A person stops being sick after visiting a user-defined number of shops. A user-defined percentage of the population can be vaccinated. A vaccinated person has a lower chance of getting the disease, but when sick is equally infectious.

Functional requirements

- The user can create clients and suppliers through a separate **control panel**
- The control panel can be used to set world parameters, such as the disease transmission rates for people with and without masks, lockdown threshold, number of shop visits before recovering from the disease, number of vaccinated people, chances of getting the disease when vaccinated
- Moving clients and suppliers are presented in the **map panel**
- All clients and suppliers are **drawn on the map** according to their current coordinates
- Every client/supplier is a **separate thread**
- The user can read about the basic properties of an object (client, supplier, shop) in a separate **information window**, which appears after clicking on an object
- Using buttons in the information window, the user can **remove a supplier/client** from the map, **change the supplier's route**, **create a new product** in the wholesale store
- The map should include a minimum of **10 retail shops** and **3 wholesale stores**
- Retail shops have limited product storage; when a shop is fully stocked up, the supplier has to wait until there is enough free storage to unload his delivery; while the supplier is waiting, he/she is not burning any gas
- Retail shops have limited client capacity; when a shop is full, the client has to wait until somebody gets out of the shop for him/her to come in
- Suppliers should have tanks in their cars big enough to travel between any pair of shops on the map (no out-of-gas problems)
- **Roads** on which suppliers travel should cross; there can be only one car on an intersection at a time; to make intersections safe, use *semaphores* or *monitors*
- **Sidewalks** on which clients travel should cross; there can be only one client on an intersection at a time; to make intersections safe, use *semaphores* or *monitors*
- Since stores have **limited capacity**, they should also be equipped with some sort of thread synchronization mechanism to controlled who is inside the shop

- The application should solve any deadlock problems; if there is a deadlock it will be resolved by removing a client/supplier
- Waiting at an intersection should cause traffic jams
- The simulation should be safe – **no collisions**
- The project will be graded based on code quality and functionality; eye-pleasing graphics will be treated as an asset worth improving the grade, but bare-bones visualizations will in no way negatively affect the project's grade
- Similarly to visual aspects, although not required, pop-cultural references will be treated as an asset (e.g., HMM, Transport Tycoon, GTA, Lego City, Outbreak)

Technical requirements

- By **December 1, 2020**, the student is required to upload a **UML class diagram** of the project
- By **December 8, 2020**, the student should have prepared a project template with **Java classes** matching (or improving upon) the UML class diagram
- All non-static fields should be private and, if needed, accessed through **getters and setters**
- All public elements should be documented with **Javadoc** and the project should be accompanied by a Javadoc-generated html documentation
- The project should be uploaded to the e-learning site by **January 29, 2021**, in the form of zip file containing:
  - The **source code** of the project
  - A **runnable *.jar** file
  - A simple **readme** containing the following information: first name, last name of the author, student ID number, short program instructions
- Delayed projects will have a -1-grade modifier for each week of delay
- Certain aspects of the project can be negotiated, a requirement can be traded for an additional feature; any changes to the project requirements must be agreed upon with the lecturer

Any plagiarism will result in a 2.0 grade given to any person copying the code of others as well as any person sharing their code; detecting plagiarism after grading the project results in changing the grade to 2.0

IN CASE OF ANY QUESTIONS PLEASE WRITE ON THE E-LEARNING WEB FORUM OR WRITE TO DB