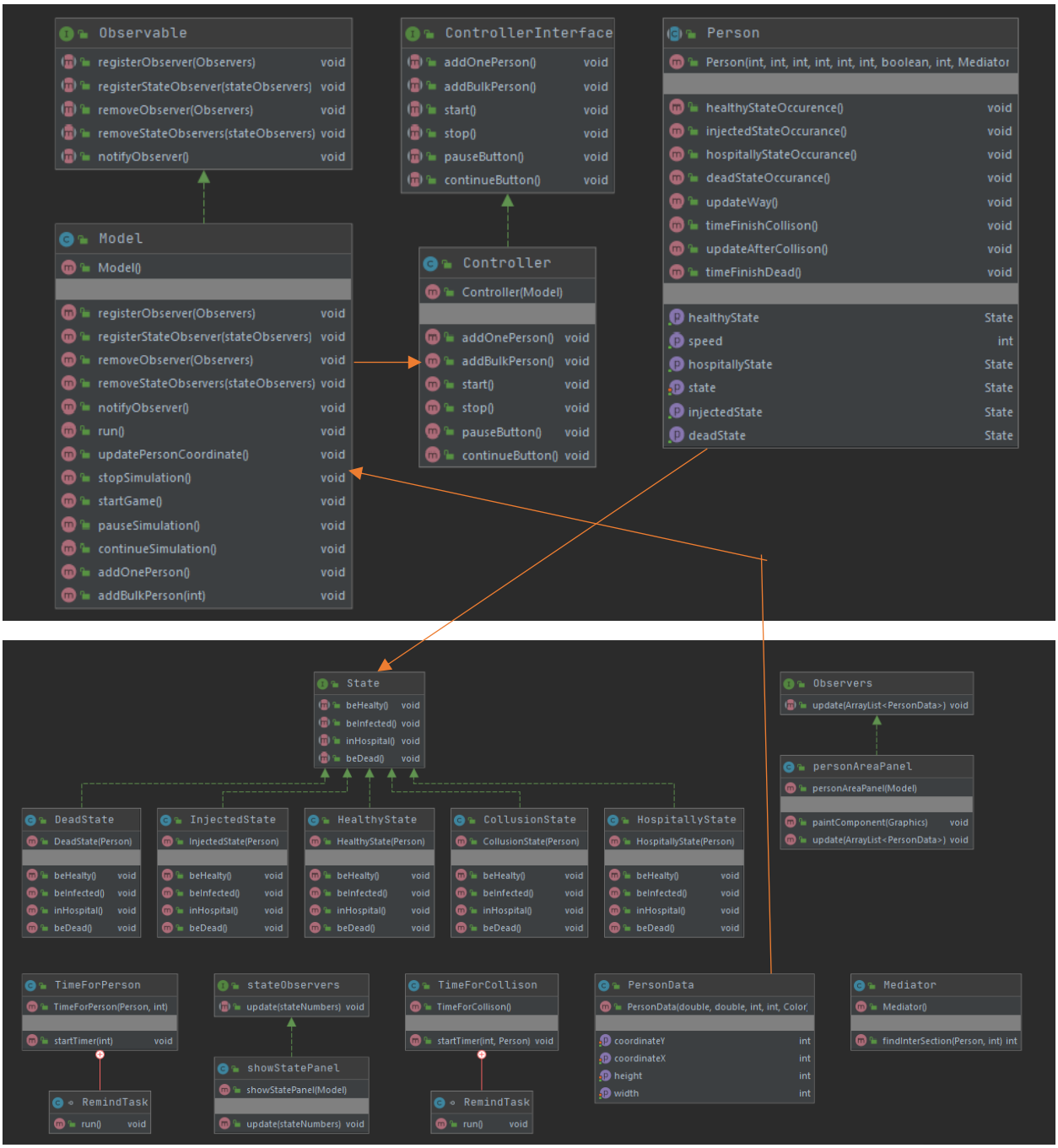
T.C.
GEBZE TEKNİK ÜNİVERSİTESİ
BİLGİSAYAR MÜHENDİSLİĞİ

CSE-443
OBJECT ORIENTED ANALYSIS AND DESİGN

FINAL PROJECT

Öğrencinin Adı ve Soyadı: Samet GÜLMEZ
Öğrencinin Numarası : 161044110


Observable
registerObserver(Observers) void
registerStateObserver(stateObservers) void
removeObserver(Observers) void
removeStateObservers(stateObservers) void
notifyObserver() void
ControllerInterface
addOnePerson() void
addBulkPerson() void
start() void
stop() void
pauseButton() void
continueButton() void
Person
Person(int, int, int, int, int, int, boolean, int, Mediator
healthyStateOccurence() void
injectedStateOccurance() void
hospitallyStateOccurance() void
deadStateOccurance() void
updateWay() void
timeFinishCollison() void
updateAfterCollison() void
timeFinishDead() void
healthyState State
speed int
hospitallyState State
state State
injectedState State
deadState State
Model
Model()
registerObserver(Observers) void
registerStateObserver(stateObservers) void
removeObserver(Observers) void
removeStateObservers(stateObservers) void
notifyObserver() void
run() void
updatePersonCoordinate() void
stopSimulation() void
startGame() void
pauseSimulation() void
continueSimulation() void
addOnePerson() void
addBulkPerson(int) void
Controller
Controller(Model)
addOnePerson() void
addBulkPerson() void
start() void
stop() void
pauseButton() void
continueButton() void
State
beHealty() void
beInfected() void
inHospital() void
beDead() void
Observers
update(ArrayList<PersonData>) void
personAreaPanel
personAreaPanel(Model)
paintComponent(Graphics) void
update(ArrayList<PersonData>) void
DeadState
DeadState(Person)
InjectedState
InjectedState(Person)
HealthyState
HealthyState(Person)
CollusionState
CollusionState(Person)
HospitallyState
HospitallyState(Person)
TimeForPerson
TimeForPerson(Person, int)
startTimer(int) void
RemindTask
run() void
stateObservers
update(stateNumbers) void
showStatePanel
showStatePanel(Model)
update(stateNumbers) void
TimeForCollison
TimeForCollison()
startTimer(int, Person) void
RemindTask
run() void
PersonData
PersonData(double, double, int, int, Color
coordinateY int
coordinateX int
height int
width int
Mediator
Mediator()
findInterSection(Person, int) int
stateNumbers
stateNumbers(int, int, int, int)
restart() void
View
View(personAreaPanel, showStatePanel, ControllerInterface
Main
main(String[]) void

I used the MVC(Model View Controller) approach in my final project. In my project, View represents the interface where the simulation runs. My controller enables me to establish a connection between the view and the model. The model allows me to control the backend of the project.

View

I used the Swing library in my interface. I designed the view section using composition pattern. I created my window using JFrame. I created 3 separate Jpanels inside it. There are buttons on the top panel to control my interface. These buttons are located inside the JPannel. In the second J panel, my simulation is realized. In the third Jpanel, the simulation information is shown as updated.

Controller

Here, on the View side, orders from the button are executed. If a change will be made on the Model side, the Controller informs the model side that this change should be made. I implemented the controller using Strategy pattern. The buttons on the view side can be given more than one function and each button can do the same action.

Model

On the model side, every invisible process of the simulation is done. Users are created here. Virus spread of users to each other is controlled. Every user actions are updated here. I used an observers pattern in my model. Every time people's coordinate information is updated, it is transmitted to observers, or Views. In this way, the flow of people has been provided in an up-to-date manner.

```
class View
```

It represents my interface. Swing library used. It consists of 3 layouts. Each layout has Jpanale. I filled my first layout with buttons and I did this in this class. I created my other panels by creating classes derived from JPanel class. Then I added my JPanel objects to the JFrame layouts. Composition pattern is used.

```
public class personAreaPanel extends JPanel implements
Observers
```

It represents the JPanel inside the JFrame object created in my interface. Simulation takes place through this panel. Draws people in the form of squares on the Panel, using up-to-date information provided by the model. Observer pattern is used to keep this information up to date. With the Observable on the model side, observers are notified by calling the notify () method that you have any changes.

```
public class showStatePanel extends JPanel implements
stateObservers
```

It represents the JPanel inside the JFrame object created in my interface. During the simulation, I show the situation change in this panel. Observer pattern is used to get the information by the model.

```
public class Controller implements ControllerInterface
```

Implemented the Controller Interface. Strategy pattern is used. The buttons are not designed to be given many tasks. It contains Model and VIew objects. Thus, it will enable communication between View and Model. For example, when a button is pressed on the View side, a method is called on the controller and the job on the Model side is performed with that method.

```
public class Model implements Observable,Runnable
```

It uses observer pattern. With this pattern, it transmits the changes to be made on the view. On the model side, my personal information is kept. This information is constantly changing with the thread I created. UpdatePersonCoordinate () so that people can move each time the thread returns; Coordinates are updated with the method. Then, this information is transferred to the View with the notifyObservers () method. Then the Coordinates of the Humans are compared among themselves. If the personal space is violated, an individual who is sick can infect another person. After the formula is calculated, if the disease is contagious, the color of the sick person is changed to red. The methods to be run of the buttons used on the view side are in this class. With the updatePersonCoordinate () method, people's situations are checked and action is taken according to these situations.

```
public class Mediator
```

Mediator pattern is used here. In this class, it is checked whether people are violating each other's social spaces. First of all, I check whether there is an intersection by comparing the coordinates of both people. If there is an incision, I check if there is any patient from both individuals. I do these illnesses with the STATE pattern I used in the Person class. If one of the two individuals is sick, the formula is applied and depending on the result, it will be sick or not. When two individuals cut, they wait where they are, taking the largest of the waiting time between them. In this case I keep a timer using another class (TimeForPerson).

```
public class Person
```

This class represents my people in the simulation. Every person is a Person object. All information about the individual is kept in this class. State pattern is used here. Individuals have health conditions.

Each health status represents a State. The movement paths of the personnel are determined in this class. Every transaction is calculated with the updateWay ()

method. There are methods that every person will call when the Status timer ends. (TimeFinishCollison, timeFinishedDead)

```
public class TimeForPerson
```

This class is created for the timer for each situation. If typeOfTimerFlag value is 1, timer is started for Collisun state. If typeOfTimerFlag value is 2, timer is started for Dead state. In addition, LeftTime is kept for Pause of the game. If the game is stopped, the remaining seconds of each timer are kept and saved. The timer is then started again on the seconds remaining.

```
public class HealthyState implements State
```

It implements the state interface. It allows the transition from its current state to other states. It does not grant passage to states that do not have a pass.

There is no hospital situation in my project. Users are successfully created. A user or collective user can be added through the interface. The game is able to start, end, pause and continue successfully. Users move smoothly. Multi-thread works correctly and always responsive. After the user becomes ill with the given time, it is made dead and removed from the simulation. People's presence information is accurately displayed at the bottom of the window. When the program is started, a sick person is put on the panel. When this disease touches a person, the second of death begins. I did it that way in case the Collison possibility would be longer.