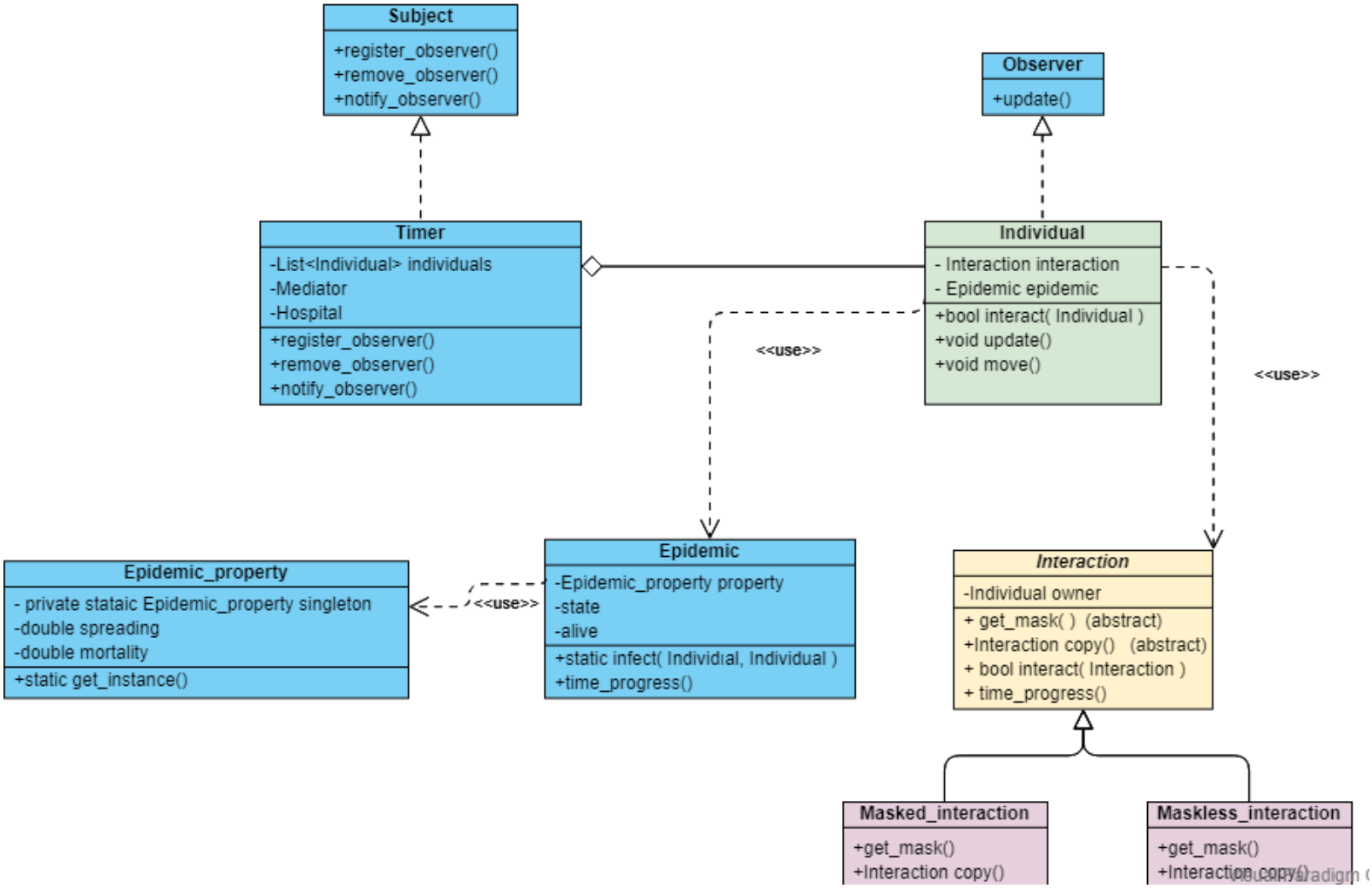
Gebze Technical University
Computer Engineering

Object Oriented Analysis and Design
CSE443 – 2020

FINAL-REPORT

Yusuf Abdullah ARSLANALP
151044046



Creating Individual Object

For individual class I used three design pattern.

1) Strategy: Using strategy individual objects can change their interactions dynamically. For now there are two kinds of interactions: Masked_interaction and Maskless_interactin.

   There can be new requirements in future. Such as more stronger masks(like N95) might be in use. Or health care professionals might have special dresses for avoiding epidemic. For this requirements we all we need to do is extending interaction abstract class.

   In edition in future we might want to test new scenarios using simulation. Such as: If epidemic reaches %2 in society people starts to wear mask. By means of strategy individuals changes their interactions from maskless to masked dynamicly.

2) Observer: Timer class implements Subject interface and Individual class implements observer interface. At every seconds timer objects notifies all registered individual objects. And individual objects moves from one location to another location in canvas.

3) Singleton: All individual object should use same values for spreading factor and mortality. Because using simulation we want to examine the consequences of a particular epidemic with different individual interactions. Best way to making sure all individuals will use same spreading factor and mortality rate is using singleton design pattern.

Singleton class are given in below. We can see that class has a private constructor. So the only way for creating object is using get instance class. If singleton object already created it returns created object.

```
2
3   public class Epidemic_property {
4       private double spreading_factor;
5       private double mortality;
6
7       private static Epidemic_property singleton;
8
9   @   private Epidemic_property( double spreading_factor, double mortality )
10      {
11          this.spreading_factor = spreading_factor;
12          this.mortality = mortality;
13
14      }
15
16      static Epidemic_property
17  @   getInstance(  )
18      {
19          if( singleton != null )
20          {
21              return singleton;
22          }
23          System.out.println( "Singleton Object not Initialized" );
24          return null;
25      }
26
27      static Epidemic_property
28      getInstance( double spreading_factor, double mortality )
29      {
30          singleton = new Epidemic_property( spreading_factor, mortality );
31          return singleton;
32      }
```

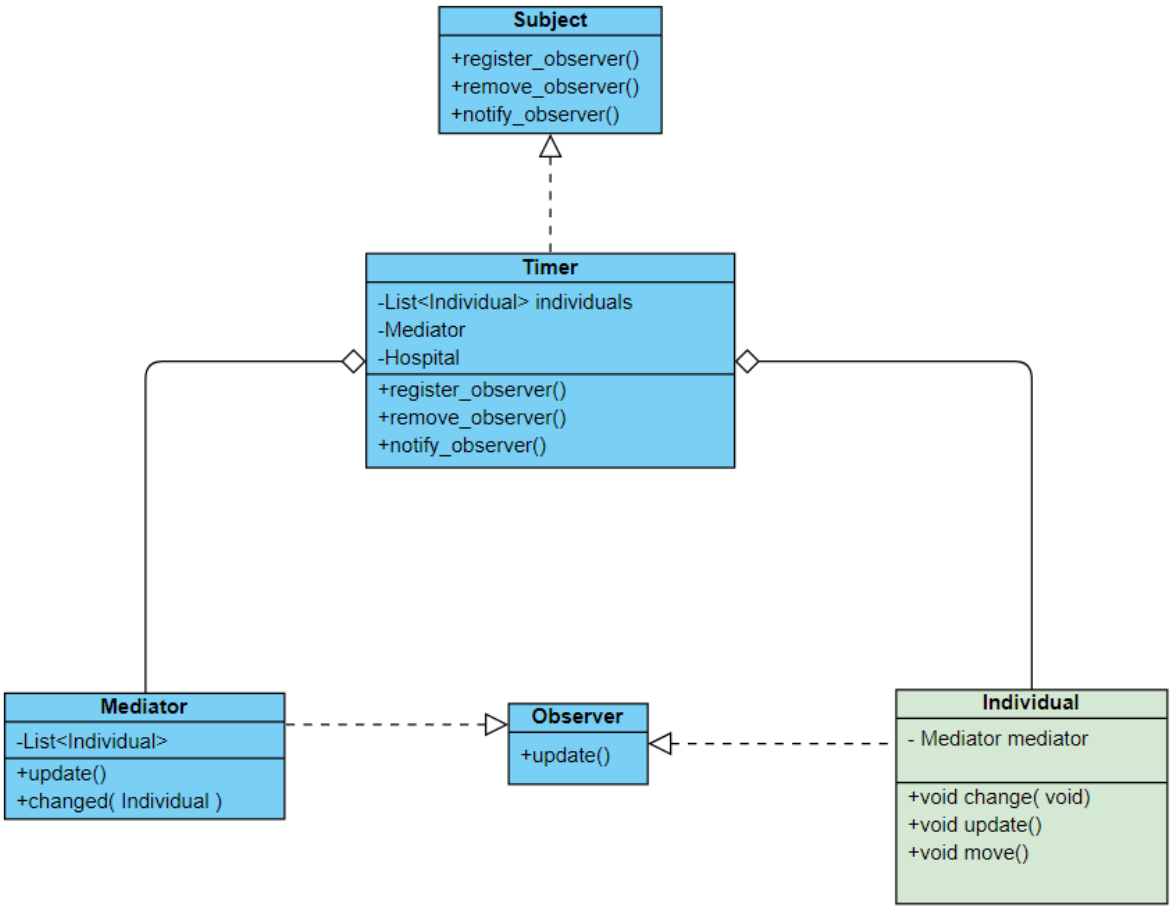
Creation steps for an Individual class are given in below. The code snipped belongs to controller class. For creating individual classes lots of input required. All the necessary inputs are taken from the user. In line 88 spreading_factor and mortality is set. For masked and maskless individual two different interaction class used(102 and 105). Social distance and collision time are set using Interaction constructor(102) The health and sick people are set using constructor of Epidemic class(107).

```
void
add_pressed()
{
    //if a valid value assigned for spreading and mortality
    //set epidemic property using singleton class;
    if( (spreding > 0.1) & (mortality > 0.1) )
    {
        //this is a singleton class.
        Epidemic_property.getInstance( spreding, mortality );
    }
    if( individual_num >0 )
    {
        add_individual( );
    }
}

void add_individual( )
{
    Individual individual = new Individual( mediator, speed, mediator.map, mediator.timer, hospital );
    Interaction interaction;
    if( masked == true )
    {
        interaction = new Masked_Interaction( individual, social_distance, collision );
    }
    else{
        interaction = new Maskless_interaction( individual, social_distance, collision );
    }
    Epidemic epidemic = new Epidemic( individual, !(healthy) );

    individual.set_interaction( interaction );
    individual.set_epidemic( epidemic );
```

Individual and Mediator Interaction



The Timer class is subject class. Mediator class and Individual class is observer class. At every second timer class first notify all registered observers. Then notify mediator class.

The notify function of Timer class given below:

```
    @Override
    public void notify_observer() {

        for( int i = 0; i < individul_obsevers.size(); i++ )
        {
            individul_obsevers.get(i).update();
        }
        hospital.update();
        mediator_observer.update();
    }
```

The notified individuals calls changed() function of mediator. And mediator add individuals to a list. The changed function of ***individual*** class are in below:

```
void changed(  )
{
    if( epidemic.is_in_hospital() == false ) {
        mediator.changed( ind: this );
    }
}
```

The change function of ***mediator*** class are in below:

```
void changed( Individual ind )
{
    individual_list.add( ind );
}
```

Only the individuals which in the canvas call change function of mediator. The dead individuals and hospitalized individuals do not call changed function of mediator.

The timer object notify mediator after individual objects. And mediator object interacts individuals which is in the individual_list.

The update() function of mediator given below:

```
@Override
public void update() {
    view.update();
    make_interaction();
    //reset individual list
    individual_list = new ArrayList<Individual>();
}
```

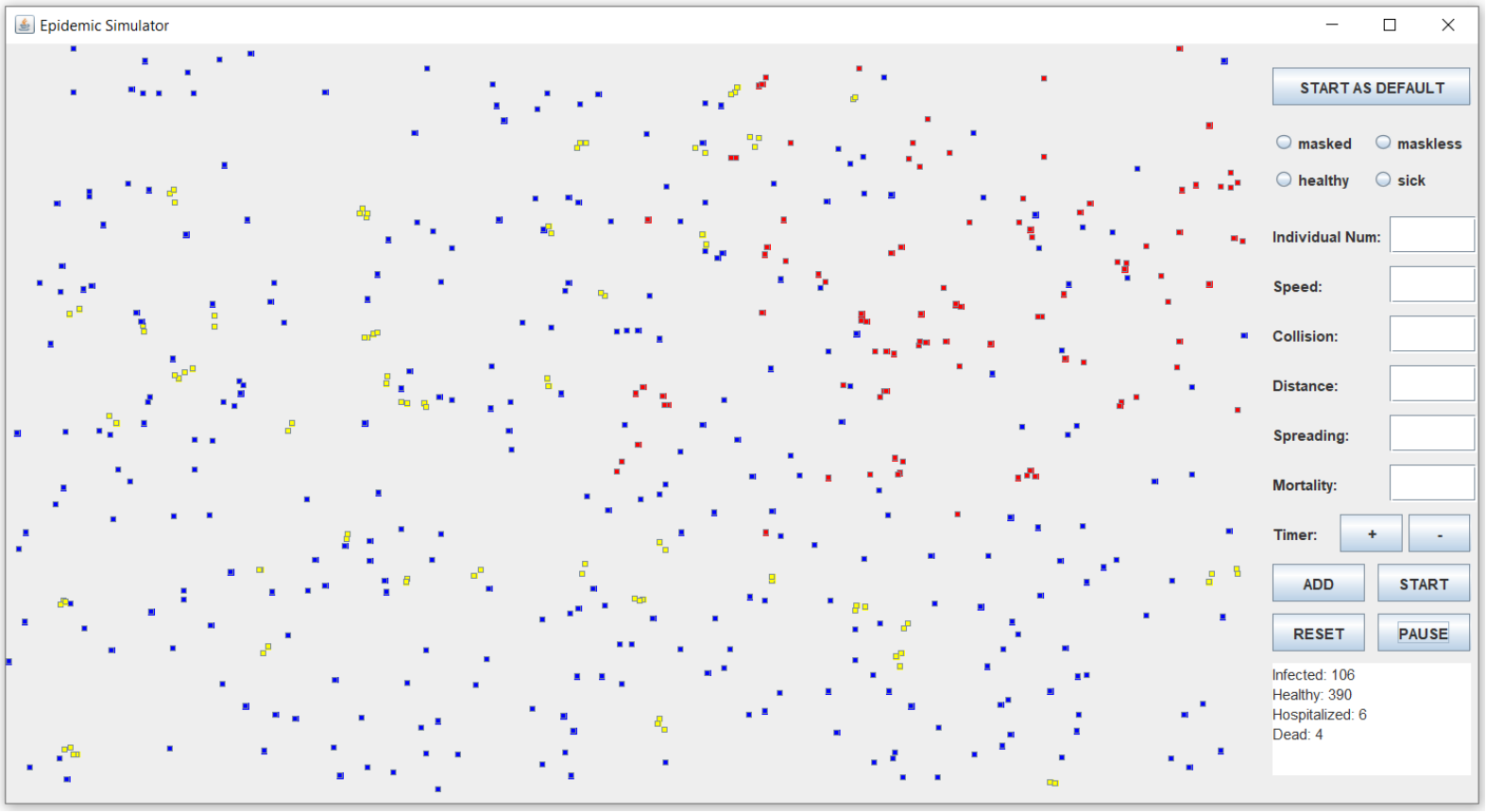
make_interaction() function of mediator given below:

```
void make_interaction()
{
    boolean interacted;
    for( int i = 0; i < individual_list.size()-1; i++ )
    {
        for( int j = i+1; j < individual_list.size(); j++ )
        {
            interacted = individual_list.get(i).interact( individual_list.get(j) );
            if( interacted == true )
            {
                break;
            }
        }
    }
}
```

GUI

The GUI of the program allow user to pause and continue. The program has two thread. Main thread and GUI thread. But for very large inputs like 100 000 individual program freeze. To avoid this situation the GUI thread should be split in to two thread. One thread should listen buttons in GUI. Other thread should perform user requests when buttons pressed.

A screen shoot for GUI are given below:



Description Of GUI Components:

In the GUI: blue squares represent healthy individuals and red squares represent sick individuals. Yellow groups indicates individuals communicating with each other.

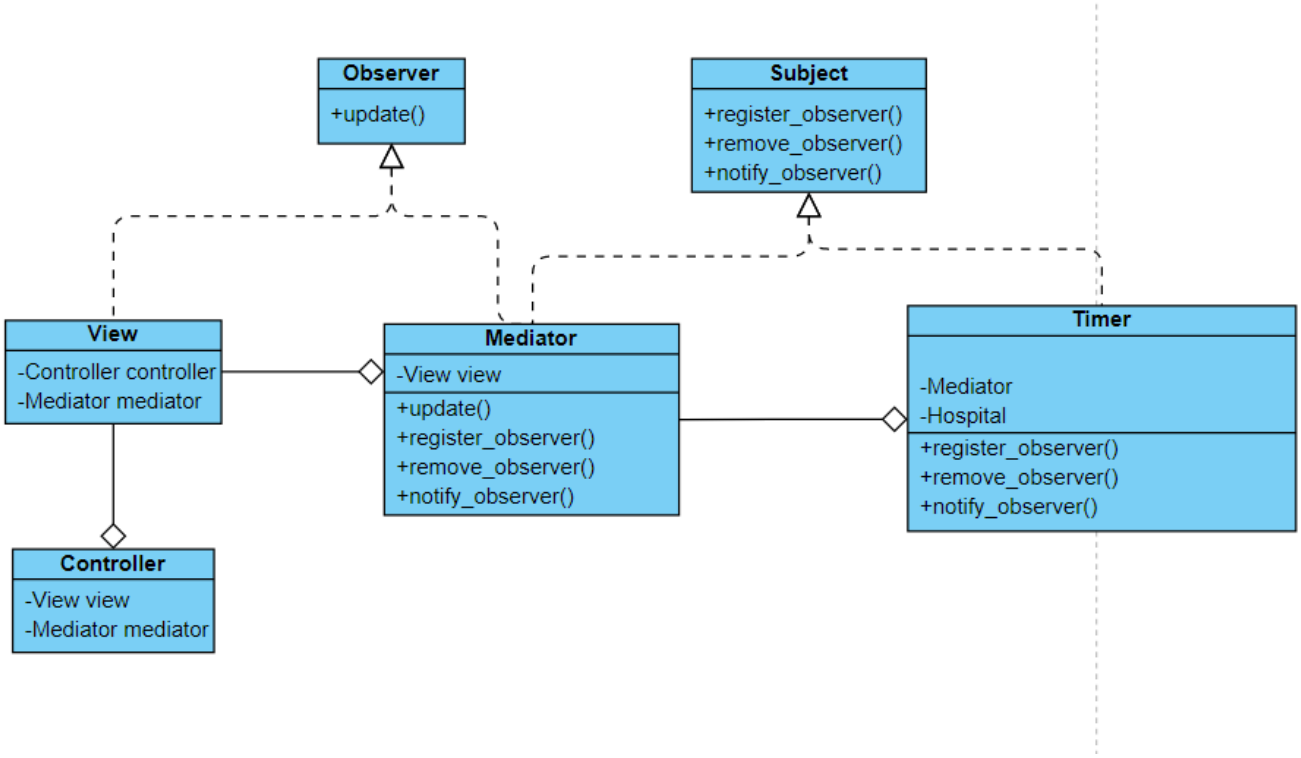
Using **START AS DEFAULT** button user can start sinulation without entering input. With this button 500 Individual created. One of them is sick.

There is a monitor at bottom right corner of GUI. Monitor Infected, healthyi hospitalized and dead peoples.

There are “+” and “-” buttons in gui. That buttons chance speed of smulation. “-” speed up simulation. “+” slow down simulation. Timer represent period of simulation.

MVC

I used MVC *Architectural Pattern* for GUI classes.



The mediator class is the model class. Mediator class implemnts both Subject interface and Observer interface. Because at every secons Timer objects notify Mediator object. Then mediator objects notify view object. Then notified view objects updates its state using functions of mediator.

The notify and update functions of mediator class are in below:

```
118     @Override
119     public void notify_observer() {
120         view.update();
121     }
122
123     @Override
124     public void update() {
125         view.update();
126         make_interaction();
127         //reset individual list
128         individual_list = new ArrayList<Individual>();
129     }
```

The update function of View class are in below:

```
167     @Override
168     public void update() {
169         states = mediator.get_states();
170         String str = controller.get_monitor_string();
171         monitor.setText( str );
172         SwingUtilities.invokeLater(
173                 new Runnable() {
174                     @Override
175                     public void run() {
176                             move_btns(  );
177                     }
178                 }
179
180         );
181     }
```

Producer Consumer:

The below code is the notify function of Timer class. Every second timer object call notify_observer() function. Notify_observer() function updates individuals sequentially.

```
78      @Override
79      public void notify_observer() {
80
81          for( int i = 0; i < individul_obsevers.size(); i++ )
82          {
83              individul_obsevers.get(i).update();
84          }
85          hospital.update();
86          mediator_observer.update();
87      }
```

If the updated individual needs to go hospital, epidemic object of individual apply to hospital. The below code belongs to Epidemic class. In the below owner is the reference of individual object.

```
            }
            if( (lossed_life >= 25) & (waiting_for_hospital == false) )
            {
                Hospital hospital = owner.getHospital();
                hospital.apply_hospital( owner );
                waiting_for_hospital = true;
            }
```

The hospital class add individuals in to a Queue. The below code belongs to Hospital class:

```
    Queue<Individual> waiting_patients = new LinkedList<~>();

    void
    apply_hospital( Individual individual )
    {
        waiting_patients.add( individual );
    }
```

In the above we showed that notify_observer() functuion of Timer class notify hospital after individuals. Notified hospital object accepts individuals to hospital sequentiaally. The accept_patients function of Hospital class given below:

```
    void
    accept_patints()
    {
        Individual individual = waiting_patients.peek();
        if(  individual.is_alive() )
        {
            individual.accepted_hospital();
            num_of_patient++;
        }
        waiting_patients.remove();

    }
```

If the capacity of the hospital is full then the patients in the queue should vait for the next second.

I Have a note in below:

When the user press pause button the GUI thread set value of cont to false. Cont varible is in the start() function of timer class. The start function given in below:

```
void start()
{
    while( true ) {
        if( cont == true )
        {
            current_time++;
            notify_observer();
            try {
                Thread.sleep( period );
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        if( cont == false )
        {
            try {
                Thread.sleep( millis: 200 );
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

There are two sleep function in above code. I added second sleep function for efficiency concern. In pause mode without second sleep function CPU usage of my computer goes to %20. But when I add second sleep function The CPU usage of my computer reduces to %3.