# CSE443

## Object Oriented Analysis and Design

# Final Project

**Prof. Dr. Erchan APTOULA**

**Melih Yabaş**

**161044072**

JavaFX software platform was used to implement this project.
This project is actually is a “Model View Controller” project. We need to design a GUI and generate a simulation. The simulation must be created step by step.

First we need to create our individuals. These are characters of the simulation. The user should be able to add them in bulk as well as one by one. So we have to choose a design pattern in appropriate for this situation.
I found appropriate to use Composite Pattern to create them. Because The Composite Pattern describes a group of objects that are treated the same way as a single instance of the same type of object.

```
public interface Element {

    public abstract void move();

    public abstract void draw();
}
```

Element interface is a base for the people. All people in the population have to have a move and a draw method.

```
public class Individual implements Element{

    public  Situation situation;
    private Location location;
    private Director director;
    private Rectangle r;
    private Pane world;
    private int sickTime = 0;
    private int collisionTime = 0;
    private int deadCount = 0;
    private int timeInHosp = 0;
    private int deadTime = (int) (5*100*(1-Mediator.getZ()));
    public static int length = 5;
    public static int healtime = 5 * 50;

    private float M; // wear mask, M=0.2; not wear mask M=1.0
    private int S; // speed [1,5] pixel/second
    private int D; // social distamce [0,9] in pixels
    private int C = 5; // sociality [1,5] seconds
    private int tempC; // Keeps max C of the pair
    private boolean goon = true;


    public Individual(){ }
```

Individual implements Element. This class can be used for single inserting into the simulation.

```java
public class People implements Element{

    ArrayList<Element> people = new ArrayList<>();

    public void move()
    {
        for(Element e:people)
            e.move();
    }

    public void draw()
    {
        for(Element e:people)
            e.draw();

    }

    public void addIndividual(Element p) { people.add(p); }
}
```

People class also implements Element. Both classes implements Element interface. So their types are same! User can use people class to add individuals in bulk.

Now, we should implement interaction between individuals.

```java
public Mediator(){
    R = 1.0f;
    Z = 0.4f;

}
public boolean collision(Individual i1, Individual i2) {

    double distance = Math.sqrt(Math.pow(i1.getLocation().getX() - i2.getLocation().getX(), 2)
            + Math.pow(i1.getLocation().getY() - i2.getLocation().getY(), 2));

    if(distance < 2*i1.getLength() + Math.min(i1.getD(), i2.getD())){
        i1.setTempC(((int)(Math.max(i1.getC(),i2.getC()))));
        i2.setTempC(((int)(Math.max(i1.getC(),i2.getC()))));
        return true;
    }
    else
        return false;

}
```

Mediator pattern is really efficient to do this.
Mediator class have some methods. They gets 2 individuals and calculates all collision possibilities.

```java
public void collisionControl(Individual i1, Individual i2)
{
    if(collision(i1,i2) && i1.getGoon() && i2.getGoon())
    {

        i1.setGoon(false);
        i2.setGoon(false);

        if((i2.getSituation() == Situation.PATIENT && i1.getSituation() == Situation.SUSCEPTIBLE)
        || (i1.getSituation() == Situation.PATIENT && i2.getSituation() == Situation.SUSCEPTIBLE))
        {
            float C = Math.max(i1.getC(),i2.getC());
            float D = Math.min(i1.getD(), i2.getD());
            float P = Math.min(R*(1+C/10)*i1.getM()*i2.getM()*(1-D/10),1);

            if(getRandBool(P)){
                i1.setSituation(Situation.PATIENT);
                i2.setSituation(Situation.PATIENT);
                i1.getR().setStroke(null);
                Controller.incInfect();
                Controller.decHealthy();
            }
```

When a collision occur, this method checks their situations. If one of them is patient, it calculates probability of infection and the other individual is affected from that.

When an individual get the virus, its situation turns into patient.

```java
public enum Situation {
    SUSCEPTIBLE{
        public Color getColor() { return Color.BLUE; }
    }, PATIENT{
        public Color getColor() { return Color.RED; }
    }, HOSPITALED{
        public Color getColor() { return null; }
    }, DEAD{
        public Color getColor() {return null;}
    };

    public abstract Color getColor();
}
```

There are 4 types of situations for an individual. These are; susceptible, patient, hospitaled and dead. Hospitaled and dead situation does not have a color because they do not seem on the canvas.

```java
public class Location {

    private double x;
    private double y;
    private boolean col = true;
    private int S;

    public Location(double x, double y, int S){
        this.x = x;
        this.y = y;
        this.S = S;
    }

    public Location(Pane world, int S){
        this( x: Individual.length + Math.random() * (world.getWidth() - 2*Individual.length)
                , y: Individual.length + Math.random() * (world.getHeight() -2*Individual.length), S);
    }
```

Location class keeps the position of an individual. It keeps x and y of an object but the direction is determined by another class named “Director”.

```
public void RandomDir()
{
    double dir = Math.random() * 2 * Math.PI;
    x = Math.sin(dir);
    y = Math.cos(dir);
}
```

It gives a random direction using sin and cos functions whenever it needed.

```
public class View {

    private ArrayList<Individual> people;
    private Mediator mediator;
    private Random random;

    public View(Pane world, int popSize)
    {
        int i, S, D, C;
        float M;
        random = new Random();
        mediator = new Mediator();
        people = new ArrayList<Individual>();
        for(i=0 ; i< popSize ; i++)
        {
            if(getRandBool( p: 0.5f))
                M = 0.2f;
            else
                M = 1.0f;
            S = getRand( low: 1, high: 500);
            D = getRand( low: 0, high: 9);
            C = getRand( low: 1, high: 5);
                people.add(new Individual(M,S,D,C, Situation.SUSCEPTIBLE, world));
        }
        if(getRandBool( p: 0.5f))
            M = 0.2f;
        else
```

View class is a kind of starter for the program. I keeps an arraylist of individuals and starts its random variables.

Controller uses this class to start simulation.

```
private class Movement extends AnimationTimer{

    private long Time = 100000000l;

    private long last = 0;

    @Override
    public void handle(long now)
    {
        if(now - last > Time){
            step();
            last = now;

            infected.setText(infSize+"");
            healthy.setText(healthySize+"");
            hospitalized.setText(hospitSize+"");
            dead.setText(deadSize+"");
        }
    }
}
```

Controller class uses JavaFX's AnimationTimer class. It creates multiple threads to move different individual objects simultaneously.

```
@FXML
TextField infected;
private static int infSize = 1;
@FXML
TextField healthy;
private static int healthySize = 0;
@FXML
TextField hospitalized;
private static int hospitSize = 0;
@FXML
TextField dead;
private static int deadSize = 0;
```

Controller keeps some variables to show total count of infected, healthy and hospitalized and dead. The canvas updated for every 0.1 seconds.

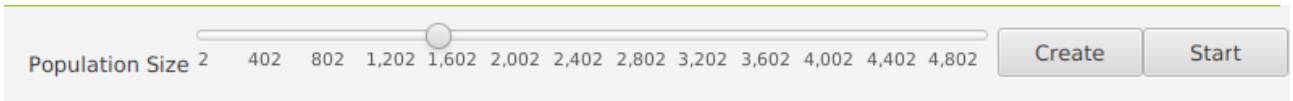
```
public static void incInfect() { infSize++; }

public static void decInfect() { infSize--; }
public static void incHealthy() { healthySize++; }

public static void decHealthy() { healthySize--; }

public static void incHospital() { hospitSize++; }

public static void decHospital() { hospitSize--; }

public static void incDead() { deadSize++; }

public static boolean isEmptyVen() { return (hospitSize < hospitalCapacity); }
```

Some increase/decrease methods to arrange variables to show on the screen. For example when an individual dies, incDead() method will be called.
IsEmptyVen() method checks whether is there an empty place for a sick individual.

# Goals

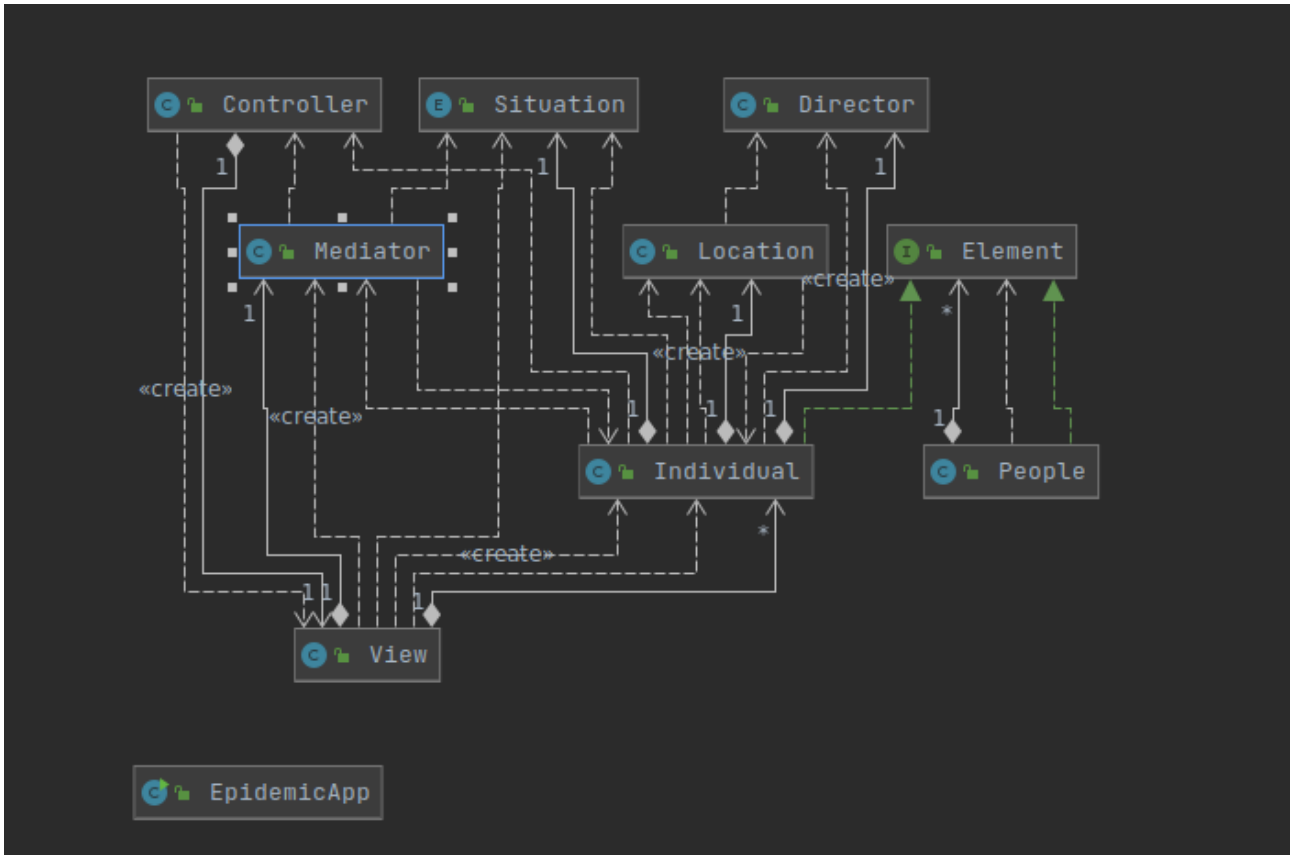
1) Composite pattern is used to create individuals
2) Mediator design pattern is used for collisions..
3) Animation Timer class of JavaFx is used to create a multi-threaded simulation.
4) I implement hospital but I could not implement producer-consumer paradigm for hospital
5) I could not have time for graphs.
6) Project is described in the report.

# Usage

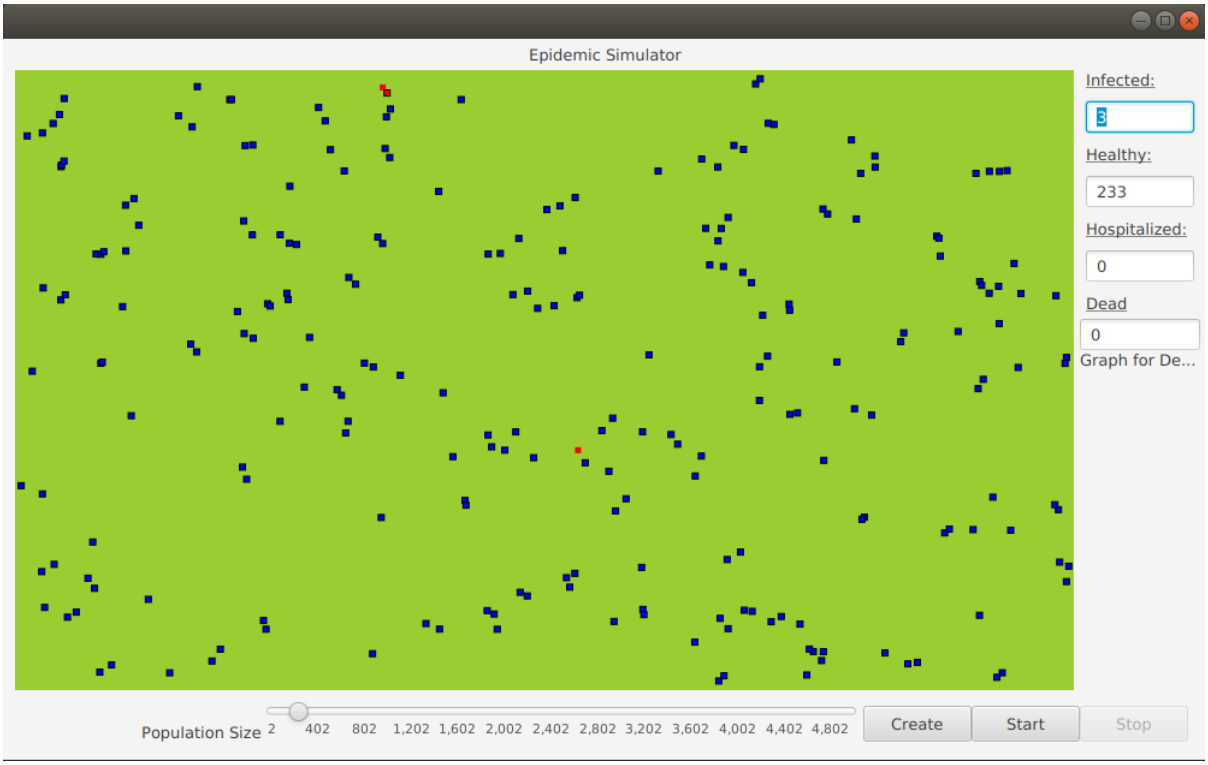


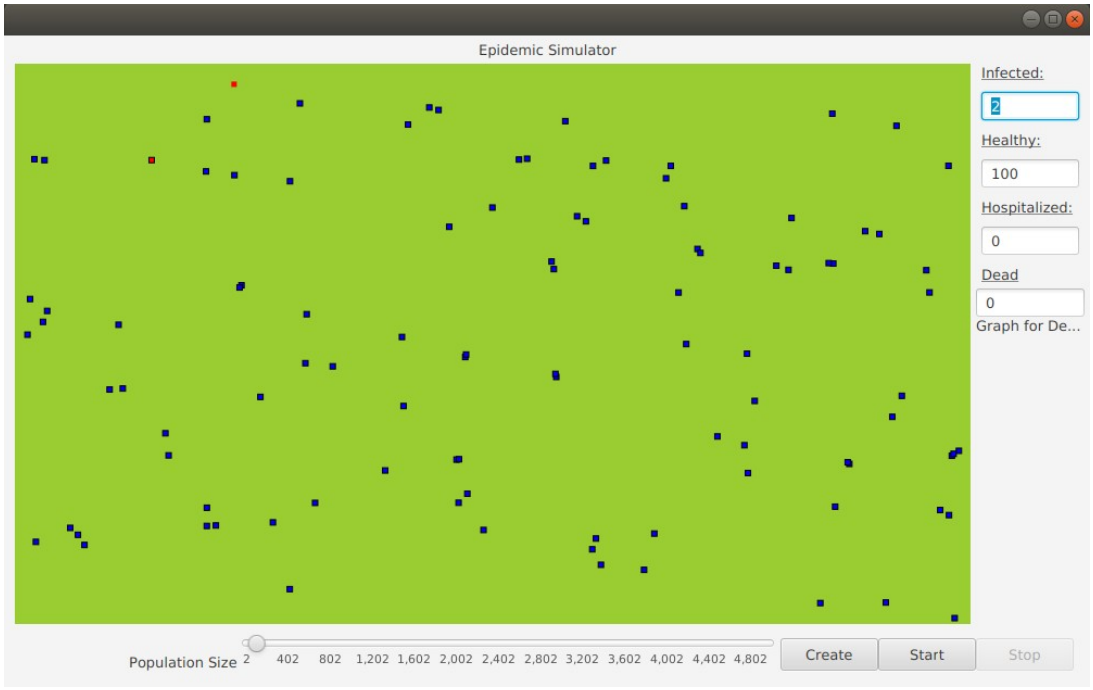

Select the population size using slider. Then click “Create” button. It will create a population. After that, click start button to move people

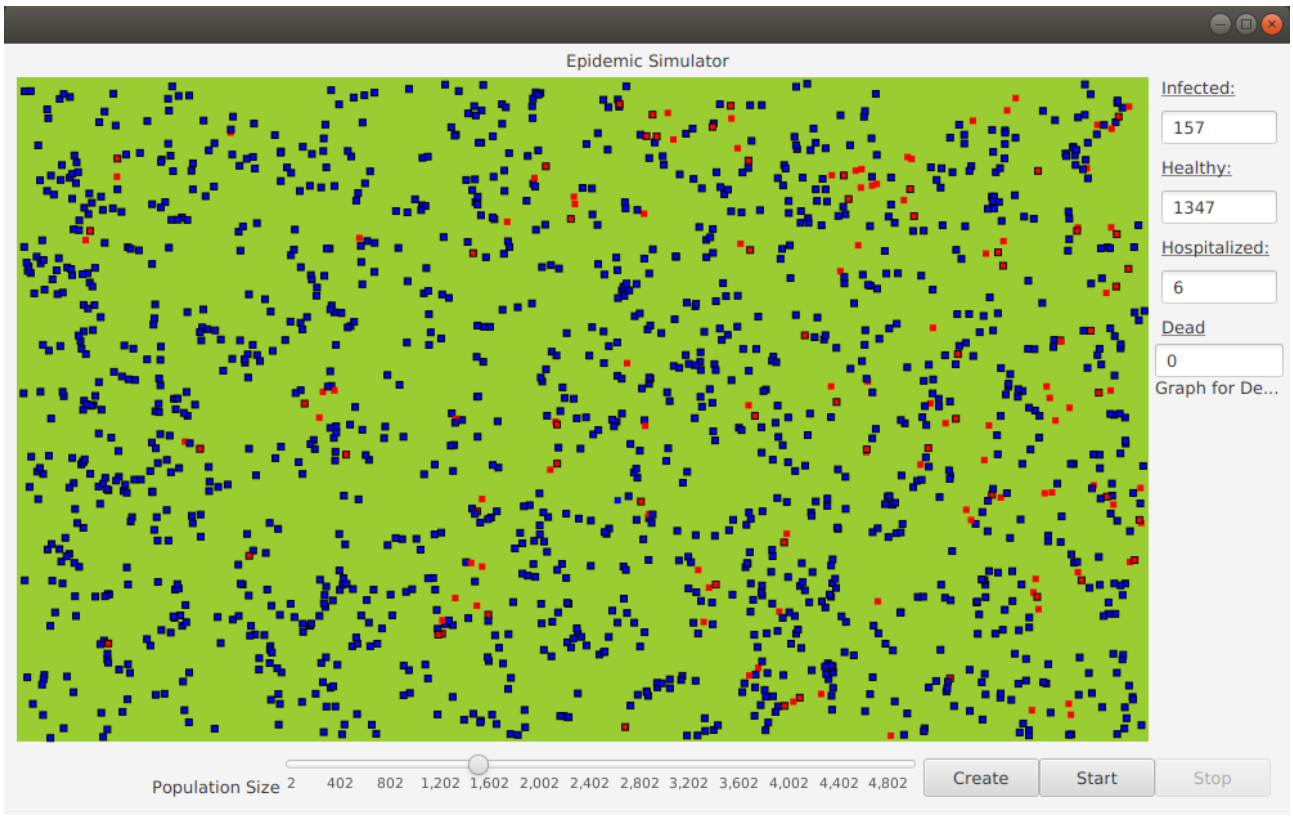
# UML Diagram


Controller
Situation
Director
Mediator
Location
Element
Individual
People
View
EpidemicApp
«create»
«create»
«create»
«create»
«create»
1
*

# Sample Outputs




Epidemic Simulator
Infected:
157
Healthy:
1347
Hospitalized:
6
Dead
0
Graph for De...
Population Size
2
402
802
1,202
1,602
2,002
2,402
2,802
3,202
3,602
4,002
4,402
4,802
Create
Start
Stop