laura.juan03@estudiant.upf.edu
rocio.gilabert01@estudiant.upf.edu

upf. Universitat Pompeu Fabra Barcelona

LAB 1: IMPLEMENTING A CLASS

Introduction:

This program should print on the screen a window with some agents (represented with red balls) that move through a defined space following a certain direction to get to a target. To do this program we have defined five different classes. The classes TestWorld, WorldGUI and Vec2D are given in the code, and we have implemented them according to what we did in the seminar 1, and we have created the AgentClasss and WorldClass classes to implement the code.

The methods used in the `Vec2D` class are:

1. `Vec2D(double xInit, double yInit)`
2. `Vec2D(Vec2D v)`
3. `add(Vec2D v)`
4. `subtract(Vec2D v)`
5. `length()`
6. `normalize()`

The methods used in the `AgentClass` class are:

1. `Agentclass(Vec2D p, double r)`
2. `SetTarget(Vec2D t)`
3. `SetSpeed( double s)`
4. `UpdatePosition()`
5. `TargetReached()`
6. `isColliding(Agentclass agent)`
7. `paint(Graphics g)`

The methods used in the `WorldClass` class are:

1. `Worldclass(int w, int h, int numA)`
2. `Vec2D randomPos()`
3. `randomRadius()`
4. `simulationStep()`
5. `Collisions()`
6. `paint(Graphics g)`

The methods used in the `WorldGUI` are:

laura.juan03@estudiant.upf.edu
rocio.gilabert01@estudiant.upf.edu

upf. Universitat Pompeu Fabra Barcelona

1. `WorldGUI()`
2. `simulate(int steps)`
3. `paint(Graphics g)`
4. `void manageCollisions()`

The methods used in the TestWorld class are:

1. `main( String[] args )`

Description:

From our point of view the best possible option to develop the program is to structure it in different classes so that it is more visual, ordered and easy to follow. That's why an alternative option could be to write all the code in the same class and moreover the program wouldn't be as ordered and structured as in the other way.
Now we will explain every class with its own methods.

- **Agentclass:** for the agent class we have created a constructor named "`Agentclass`" where we initialize the class assigning to each agent a position and a radius.

```
public Agentclass(Vec2D p, double r){
    position  = p;
    radius = r;
}
```

The next method is called "`SetTarget`" and assigns to the agent a direction to reach the desired target.

```
public void SetTarget(Vec2D t){
    target = t;
    direction = new Vec2D(t) ;
    direction.subtract(position);
    direction.normalize();
}
```

Another one is the "`SetSpeed`" method whose function is to assign a speed to each agent.

```
public void SetSpeed( double s){
    speed = s;

}
```

laura.juan03@estudiant.upf.edu
rocio.gilabert01@estudiant.upf.edu

Another method is the "`UpdatePosition()`" changes the position of the agents by multiplying the speed by the direction and adding it to the initial position.

```
public void  UpdatePosition(){
    double NewPosX = speed* direction.getX();
    double NewPosY = speed * direction.getY();

    Vec2D new_position= new Vec2D(NewPosX, NewPosY);
    position.add(new_position);
}
```

"`TargetReached()`" method is a boolean that checks whether the agent has reached the target or not.

```
public boolean TargetReached(){
    Vec2D x = new Vec2D(target.getX() - position.getX(), target.getY() - position.getY());
    double length =  x.length();
    if (length < radius){
        return true;
    }
    return false;
}
```

Another method is "`isColliding`" which is a boolean too and detects and manages collisions between agents.

```
public boolean isColliding(Agentclass agent){
    Vec2D x = new Vec2D (agent.position);
    double radius1 = agent.radius;
    x.subtract(position);
    double distance1 = x.length();
    double total_radius = radius1 + radius;
    if (distance1 <= total_radius){
        return true;
    }
    return false;
}
```

The last method of the AgentClass is called "`paint`" and it's necessary to visualize the agents.

```
public void paint(Graphics g){
    int x = (int) (position.getX() - radius);
    int y = (int) (position.getY() - radius);
    int d = (int) (2*radius);

    g.setColor(Color.RED);
    g.fillOval(x, y, d, d);
}
```

- **WorldClass:** in order to create all the functions of the world we have created a new class with the following methods. The first is the constructor that is called "`public Worldclass(int w, int h, int numA)`" . In this function we have initialized all the variables of the window (width, height and margin), and also we have initialized agents in the world, setting a position and a radius, a speed and a target.

laura.juan03@estudiant.upf.edu
rocio.gilabert01@estudiant.upf.edu


```
public Worldclass(int w, int h, int numA){
    width = w;
    height = h;
    margin = 30;
    numAgents = numA;
    agents = new Agentclass[numAgents];

    for(int i = 0; i <numAgents; i++){
        agents[i] = new Agentclass(randomPos(), randomRadius());
        agents[i].SetSpeed(s: 1);
        agents[i].SetTarget(randomPos());
    }
}

private Vec2D randomPos(){
    double x = margin + Math.random() * (width - 2*margin);
    double y = margin + Math.random() * (height - 2*margin);
    return new Vec2D(x, y);
}
```

Next we have defined two methods, "`private Vec2D randomPos()`" and "`private double randomRadius()`" that assign random values for the position and radius of the agents.

```
private Vec2D randomPos(){
    double x = margin + Math.random() * (width - 2*margin);
    double y = margin + Math.random() * (height - 2*margin);
    return new Vec2D(x, y);
}
```

```
private double randomRadius(){
    return 5 + Math.random()* ( margin -5);
}
```

The next function is called "`public void simulationStep()`" and it checks if the agent has reached the target and in that case it assigns another target and if not it

```
public void simulationStep(){
    for(int i = 0; i <numAgents; i++){
        if(agents[i].TargetReached() == true){
            agents[i].SetTarget(randomPos());
        }
        else{
            agents[i].UpdatePosition();
        }
    }

}
```

changes its position.

The last function "`public void paint(Graphics g)`" calls the function "`paint`" defined in the Agent class, and assigns it to every node.

- **WorldGUI:** here we have called the Worldclass in order to call the functions that we have created there.

Now we have called all the functions that are created in the Worldclass, such as in the constructor "`public WorldGUI()`" :

```
public WorldGUI() {
    // initialize the world here
    World = new Worldclass(w: 800,h: 600,numA: 10);
```

```
        for ( )
            age
        }
    }
```

```
    // define a world attribute here
    private Worldclass World;
```

laura.juan03@estudiant.upf.edu
rocio.gilabert01@estudiant.upf.edu

or in the `void simulate(int steps)` and `public void paint(Graphics g)` functions:

- **<u>Manage Collisions</u> (extra exercise):**

```
public void simulate(int steps) throws Exception {
    for (int i = 0; i < steps; ++i) {
        // sleep for 10 milliseconds
        Thread.sleep(millis: 10);

        // perform a simulation step of the world
        World.simulationStep();

        repaint();
    }
}
```

```
public void paint(Graphics g) {
    super.paint(g);

    World.paint(g);

}
```

```
public void set_newTarget(Agentclass agent){

    double x =  agent.getPosition().getX() - agent.getTarget().getX();
    double y =  agent.getPosition().getY() - agent.getTarget().getY();

    Vec2D newTarget = new Vec2D(x,y);
    newTarget.add(agent.getPosition());
    agent.SetTarget(newTarget);

}

public void manageCollisions(){
    for(int i=0; i<numAgents; i++){
        for(int j=i+1; j<numAgents; j++){
            if(agents[i].isColliding(agents[j]) == true ){
                set_newTarget(agents[i]);
                set_newTarget(agents[j]);

            }
        }
    }
}
```

For this exercise we have created new different methods in the program. First we have created to getter functions that return the position and the target of the agents so that we can access them. Then we have created a new function "`public void set_newTarget(Agentclass agent)`" that subtracts the position and the target of the agent and creates a new target so that when the agent collides with another agent it will go back to the initial point. Then, thanks to the "`isColliding`" method we used a for loop to check if the function is true or not and it that case we assign a new target.

laura.juan03@estudiant.upf.edu
rocio.gilabert01@estudiant.upf.edu

Universitat Pompeu Fabra Barcelona

Conclusion:

In this laboratory we have learned how to implement a design consisting of the Agent and World classes in Java.
After completing every part, the result is the expected one. However, although at the beginning the optional part didn't work like it should after changing some things, finally it does what it should.
Nevertheless we strongly think that the practice class was correctly done and the document explained enough according to what we had to do.
To sum up, we only have to say that the lab was okay according to what we have done in class and we only have had doubts at doing the optional part.