Java Object Oriented Programming
Implementing classes and interfaces

1 Introduction

The objective of this lab session is to learn to design an object oriented applica-tion applying the concept of inheritance and also using interfaces. The session is mandatory and you have to deliver the source code of the java project and a document describing the implementation.

2 Creating the main GUI Application

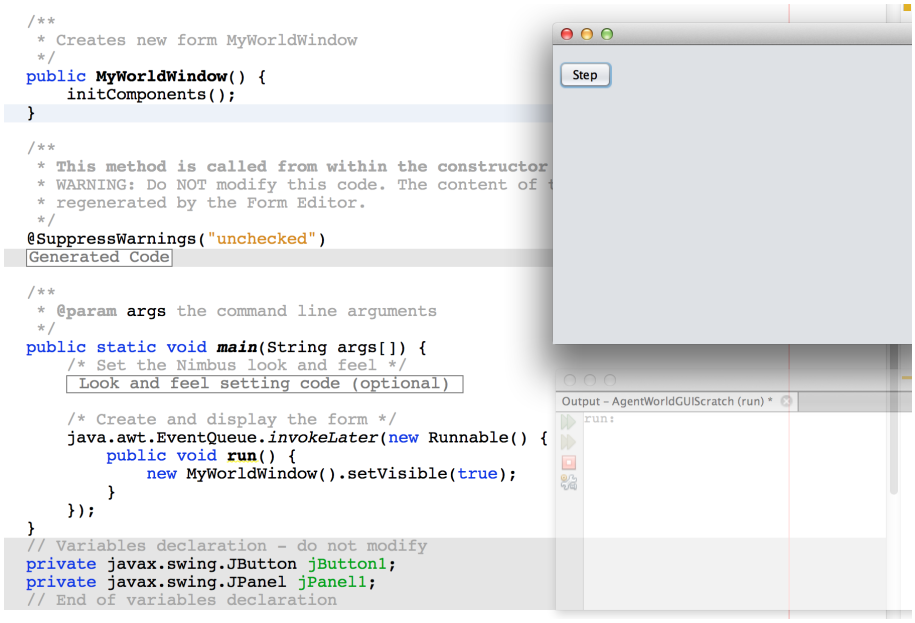
The steps for creating the graphics project are the same than lab session 1 (you should be very fast with this!). We will start creating a Netbeans java project EntityWorldGUI that will be a GUI project.

- Create the project EntityWorldGUI and remember from previous lab session to **UNSELECT** the create main class option.
- We will then add a JFrame container as in last session giving it the name MyWorldWindow. Remember: right-click the EntityWorldGUI node and choose New ; JFrame Form.
- We will add finally a JPanel in JFrame as in last session. Remember click JFrame in the right hand side column. Then put it into the window frame. Extend it to occupy the whole window as in previous session.
- Finally add a JButton inside the panel. Change the name of the button to Step.
- We are also going to add a behavior to the button. Right mouse button click on top of it and go to: Events; Action; Action Performed.
- As we did in the previous session add the following code for the button:

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    repaint();
```

}

The repaint() method is used to redraw the Panel every time the button is pressed. Now switch to the sources view of the file MyWorldWindow. Run the program and test everything works. Remember when you click play you are asked to select the main class just say OK. The result after executing should be similar to the one showed in the figure:



3 Adding the supplied files

In Netbeans there is no explicit menu to add files to an existing project. Simply copy the supplied files : Vec2D.java, World.java, Entity.java, MyPolygon.java in the sources folder where the project is stored.

Important: Run the program and check everything is ok. You should see the window with nothing different.

4 Changes in the main window class

Three new steps to be performed in the main window class MyWorldWindow.

- Add a declaration of a World: w and initialize by creating an instance in the constructor of MyWorldWindow.
- Change the initial setting of the window size (added previously) obtaining the real width and height of the world getters.

- Add the function paint as specified in the code below.

The result of these three steps should be the following code:

```
public class MyWorldWindow extends javax.swing.JFrame {
    World w;

    public MyWorldWindow() {
        initComponents();
        w = new World();
        setSize(w.getW(),w.getH());
    }

    public void paint(Graphics g) {
        super.paint(g);
        w.run(5);
        w.draw();
    }
```

Important: Run the program and check everything is ok. You should see small black dots in the screen which are the positions of our entities.

5 The shape hierarchy

Java already has a shape hierarchy. See it in the figure below. The class hierarchy has methods very similar to what we discussed in class. Method intersects (determines if two shapes intersect by its bounding box) and method contains (determines if the coordinates are inside the shape).

Constructor Summary

Constructors

Constructor and Description
`Polygon()` Creates an empty polygon.
`Polygon(int[] xpoints, int[] ypoints, int npoints)` Constructs and initializes a `Polygon` from the specified parameters.

Methods

Modifier and Type	Method and Description
void	`addPoint(int x, int y)` Appends the specified coordinates to this `Polygon`.
boolean	`contains(double x, double y)` Tests if the specified coordinates are inside the boundary of the `Shape`, as describec
boolean	`contains(double x, double y, double w, double h)` Tests if the interior of the `Shape` entirely contains the specified rectangular area.
boolean	`contains(int x, int y)` Determines whether the specified coordinates are inside this `Polygon`.
boolean	`contains(Point p)` Determines whether the specified `Point` is inside this `Polygon`.
boolean	`contains(Point2D p)` Tests if a specified `Point2D` is inside the boundary of the `Shape`, as described by th
boolean	`intersects(double x, double y, double w, double h)` Tests if the interior of the `Shape` intersects the interior of a specified rectangular area
boolean	`intersects(Rectangle2D r)` Tests if the interior of the `Shape` intersects the interior of a specified `Rectangle2D`.

We have used the Polygon class to implement a new class MyPolygon that inherits from Polygon as we suggested in the seminar (this file has been supplied).

- It contains a method Triangle that creates an isosceles triangle in the direction pointed by the passed vector parameter.
- It contains a function to create a random polygon.

You will add now two functions to the MyPolygon class:

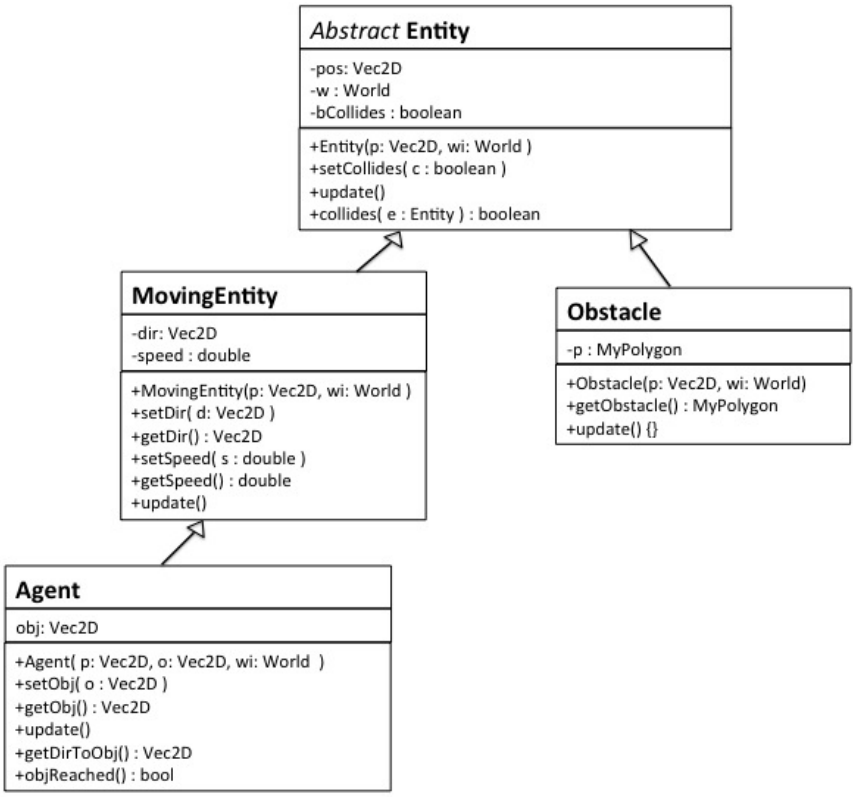
- getCentroid() that returns a Vec2D the center of the polygon. The center of the polygon is computed by summing up all the x and y coordinates and dividing by the number of points. You can access x coordinates directly using the attribute xpoints (an array of int). You can access y coordinates directly using the vector ypoints. The number of points is stored in the attribute npoints. All these attributes are accessible because they are protected in the parent's class Polygon from which MyPolygon inherits.
- center() that will center the polygon at the origin $(0, 0)$. For this you will use the previous function to compute the center and then you will use the method translate of Polygon to center it. The method translate is passed two int's. To place it at $(0, 0)$ you will have to pass the coordinates with a minus sign.

Important: Run the program and check everything is ok. Let's test the class. Go to the constructor of MyWorldWindow and declare and create a MyPolygon. Translate it to the center of the screen so that you can see it using the method translate. Finally draw it in the paint function. Resulting code should look like:

```
public class MyWorldWindow extends javax.swing.JFrame {

    World w;
    MyPolygon p;

    public MyWorldWindow() {
        initComponents();
        w = new World();
        setSize(w.getW(),w.getH());

        p = new MyPolygon();
        p.randomPolygon();
        p.translate(400,300);
    }

    public void paint(Graphics g)
    {
        super.paint(g);
        w.run(15);
        w.draw(g);
        g.drawPolygon(p);
    }
```

6 The world of entities

We will implement in this section the Entity hierarchy. Follow these steps carefully. The attributes and functions that we are going to implement appear in the following design:



- Add a class called MovingEntity that inherits from Entity.
- Run the program and see what happens: the constructors are missing.
- Add the attributes dir, speed to it as in the class diagram design.
- Add the necessary constructor. Notice that entity is created with a position and a reference to the world. Just create an arbitrary direction using the constructor Vec2D(0,1).
- Add a class Agent that inherits from MovingEntity with its appropriate constructor described in the following section 6.2.1.
- Add a class Obstacle that inherits from Entity with its appropriate constructor described in the following section 6.3.1.

- We are now going to make Entity an abstract class. Write the word abstract in front of the class.
- Run the program: what happens? We are going to fix this problem. Change the World constructor so that it creates Agents and not Entities.
- **Important:** Now the program should run fine.

6.1 The MovingEntity methods

We give now some details of the implemented methods.

6.1.1 The MovingEntity constructor

Call the parent constructor with a position and the world reference. Create an arbitrary direction dir and set the speed to 1.

6.1.2 Getters and setters

Implement the MovingEntity getters and setters as in the class diagram: for dir and speed.

6.1.3 The update

Implement the update method as in previous lab session. The position of the Entity is updated according to its speed:

```
public void update() {
    pos.setX( speed * dir.getX() + pos.getX()  );
    pos.setY( speed * dir.getY() + pos.getY()  );
}
```

6.2 The agents methods

We are now going to implement the Agents following the class diagram.

6.2.1 The Agent constructor

The agent constructor has as parameters the position, the objective and the world reference. Call the constructor of the parent class. Create an arbitrary objective obj and an arbitrary direction dir.

6.2.2 Getters and setters

Implement the Obj getters and setters.

6.2.3 Managing objectives

Implement getDirObj and objReached as in previous lab session.

```
public Vec2D getDirObj() {
    Vec2D dobj = new Vec2D( obj.getX(), obj.getY());
    dobj.minus(pos);
    dobj.normalize();
    return dobj;
}


public boolean objReached() {
    boolean result = (pos.dist( obj ) < 10);
    return result;
}
```

6.2.4 The agent update

We are going to override the update method of the agent. In it we are going to call the update of its parent class (remember that this is done by using the special word super.update(). We are going to add a modification for its direction that in previous lab session was in the world update loop:

```
dir.turnTo( getDirObj() );
```

6.2.5 Managing the agent's shape

We are going to implement a method getShape that will return a MyPolygon object with the appropriate shape in it. With the following signature:

```
public MyPolygon getShape() {
}
```

This function creates a triangle polygon in the direction dir and after centering it will translate it into position pos. Write with one instruction per line:

- First create a MyPolygon object and put it in variable t
- call the method Triangle to create a triangle with the direction of the agent
- call the center method
- call the translate method with the pos coordinates
- return the MyPolygon object

6.3 The obstacle methods

6.3.1 Obstacle constructor

We are going to call the constructor of the parent class with the position and the world reference.

- First create a MyPolygon object and put it in variable p
- call the method randomPolygon to create the Obstacle
- call the center method
- call the translate method with the pos coordinates

6.3.2 Obstacle getters

Add a getter to access for accessing the MyPolygon attribute of obstacle:

```
MyPolygon getObstacle() { return p; }
```

7 Adding the Drawable interface

The following imports will be necessary, add them in the beginning of the agent, obstacle and MovingEntity files:

```
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Polygon;
```

We are now going to add the Drawable interface. Go to new file and click the option Java Interface and Next. In the name field write Drawable and click finish. Add the import in the of the file and a method in the interface, the whole file should look like that:

```
import java.awt.Graphics;

public interface Drawable {
    public void draw(Graphics g);
}
```

Make the class World class implement the interface Drawable. What happens if you now run the program? Make the class Entity implement the interface Drawable. Run the program and what happens? We have to implement the draw methods. Add a method draw in the class MovingEntity:

```
public void draw(Graphics g) {
    if(bCollides) g.setColor(Color.RED);
    else          g.setColor(Color.BLACK);
    g.fillOval((int)(pos.getX() - 3), (int)(pos.getY() - 3), 6, 6);
}
```

Add the method draw to the class Agent to draw the agent and its objective:

```
public void draw(Graphics g) {
    if(bCollides) g.setColor(Color.RED);
    else          g.setColor(Color.BLUE);
    g.drawPolygon( getShape() );
    g.setColor(Color.BLUE);
    g.drawOval((int)(obj.getX() - 2), (int)(obj.getY() - 2), 4, 4);
}
```

Add the method draw in the class Obstacle:

```
public void draw(Graphics g) {
    if(bCollides) g.setColor(Color.RED);
    else          g.setColor(Color.ORANGE);
    g.drawPolygon(p);
}
```

8 World

Now create in the main program some Agents, Obstacles and MovingEntity. Run the program to check what happens. Now go to the method draw of World and add a call to the draw method of each entity in the for loop. Run the program and see if it has any effect.

8.1 World update

We are going to add the objective check in the update loop of the world. After the update of the entity we are going to check if the variable stores an instance of an agent (remember that this is done with instanceof). If this is the case we are going to cast the variable into an Agent variable. Then we are going to test if the objReached and if it is the case we are going to set a new objective inside the world with randomPointInsideWorld.

```
void update() {
    removeOutsiders();
    processCollisions();
    for(int i=0; i<N; i++) {
        Entity ei = (Entity) entities.get(i);
        ei.update();
        if(ei instanceof Agent) {
            ...
        }
    }
}
```

9 Improving the collisions

We are now going to override the method collides in the agent to improve it. If the colliding entity is an obstacle we are going to use the interscect function available in shape. Look at its signature it receives a bounding box. You can access its bounding box by calling the function getBounds2D as in the code below. Remember you can use the getter getObstacle to access MyPolygon:

```
public boolean collides(Entity e) {
    if(e instanceof Obstacle) {
        Obstacle o = (Obstacle) e;
        MyPolygon t = getShape();
        ...
        ... t.getBounds2D() ...
      ...
    }
    return super.collides(e);
}
```

10 Adding projectiles

We will consider that Projectiles are of type MovingEntity. We will add a method addProjectile in the world class that has as parameter the agent which fires:

```
void addProjectile(Agent a) {
        ...
    N = entities.size();
}
```

The method will:

- Create a moving entity m
- set its direction with the setDir setter
- set its speed to the double of the agent speed
- add the moving entity to the list of entities
- set N to the new number of entities

We will now add a call to addProjectile in the update method of Agent using the random condition that follows:

```
public void update() {
        ...
        if((int) (Math.random() * 100) == 0) ...
}
```

11 Supplementary Things

If you want to do more things you can add the energy points to the entity and try to subtract them correctly when they collide. You can then use the Comparable interface described in class to sort the agents with higher energy points. Remove entities i f they reach 0 energy.