Name and, if possible, ID#:

AMERICAN UNIVERSITY OF ARMENIA
College of Science and Engineering
COMP120 Introduction to Object-Oriented Programming

FINAL EXAM

Date:	Monday, May 18 2015
Starting time:	09:20
Duration:	1 hour 40 minutes
Attention:	**ANY TYPE OF COMMUNICATION IS PROHIBITED**

13/15

Please write down your name at the top of all used pages

Problem 1

Consider below a ***public interface Valuable*** that includes the only method ***public double value(double x)***:

```
public interface Valuable {

      public double value(double x);
}
```

1.1 Implement a ***public class Function*** that encapsulates a member variable of type ***Valuable*** and computes its derivative at the specified point ***x*** using the approximation:

$$f'(x) \approx \frac{f(x+dx) - f(x-dx)}{(2 * dx)}$$

```
public class Function {

      private Valuable f;
      private double dx;

      public Function(Valuable newValuable, double newDX) {
            //TO BE IMPLEMENTED
      }

      public double derivative(double x) {
            //TO BE IMPLEMENTED
      }
```

1.2 Implement an expression

$$exp(-a * (x - c)^2)$$

as a ***public class Gauss*** that implements the interface ***Valuable*** and encapsulates double parameters ***a*** and ***c***. The parameters are initialized by the two-argument constructor ***public Gauss(double newA, double newC)***;

1.3 In a separate ***public static void main(String args[])*** write a code that inputs two double values, creates an object of type ***Gauss*** and, using the class ***Function***, prints the value of its derivative at the ***x = 1.0*** point:

```
public static void main(String args[]) {
      Scanner input = new Scanner(System.in);
      double a = input.nextDouble(), c = input.nextDouble();

      //TO BE COMPLETED

}
```

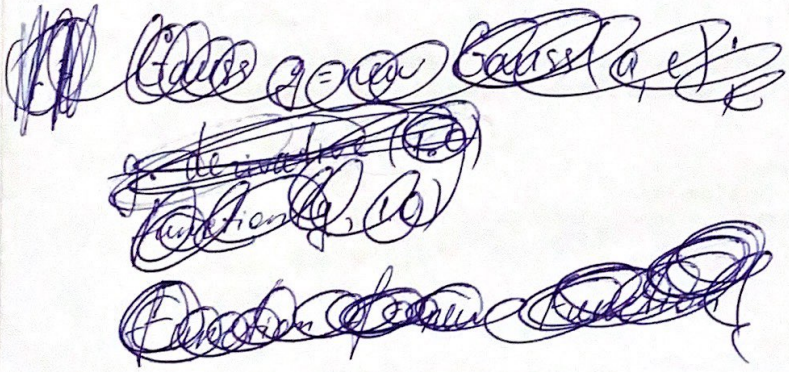
OOP. FT. 180525. H089

Use the backside, if needed

ble, double
newDX){

f=newValuable;
dx=newDX;
}
public double derivative(double x){
return(f.value(x+dx)-f.value(x-dx))/2*dx

(1.2) public class Gauss implements Valuable{
private double a, c;
public Gauss(double newA, double newC){
a=newA;
c=newC;
}
public double value(double x){
return Math.exp(-a*(x-c)*(x-c));
}



4

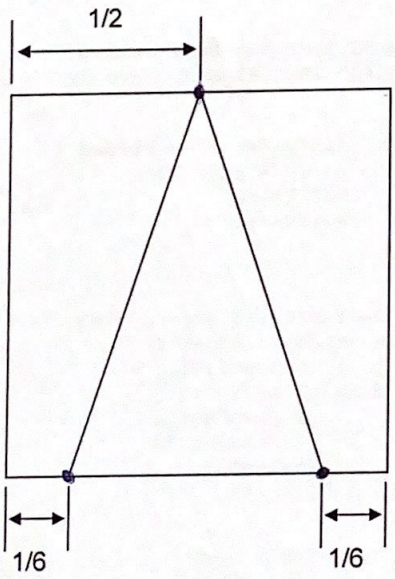
Name and, if possible, ID#: _____

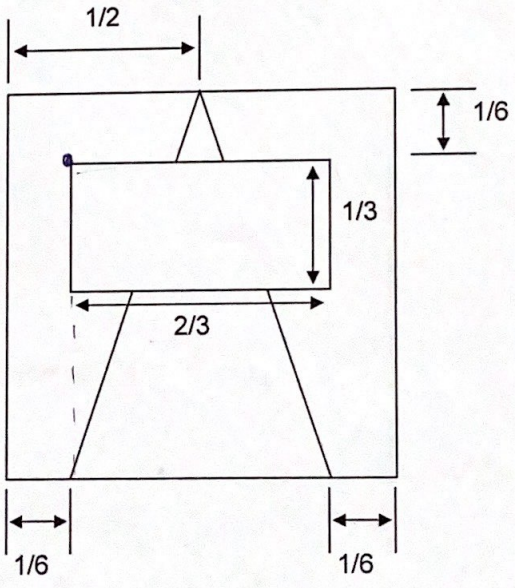
Problem 2

All 6 types of chess pieces can be drawn based on simple sketches consisting of a triangular base and rectangular cap. Consider below a ***public class ChessPiece*** that implements the triangular base only. Its geometry relative to the unit size of the square field is also sown:

```
public class ChessPiece {

      private Rectangle field;
      private Polygon base;

      public ChessPiece(int size) {
           field = new Rectangle(size, size);
           base = new Polygon(); //initially empty polygon
            base.addPoint(size / 6, size); //left vertex of the base
            base.addPoint(5 * size / 6, size); //right vertex of the base
            base.addPoint(size / 2, 0); //top vertex of the base
      }

      public void drawBase(Graphics g) {
            g.drawRect(field.x, field.y, field.width, field.height);
            g.drawPolygon(base);
      }

      public void drawCap(Graphics g) {
      }

      public void draw(Graphics g) {
            g.drawBase(g);
            g.drawCap(g);
      }
}
```

Extend a ***public class Rook extends ChessPiece*** that encapsulates ***Rectangle cap*** member variable. Implement the constructor and override ***public void drawCap(Graphics g)***. The geometries of the general chess piece and the rook are shown below:





Use the backside, if needed

OOP. FT. 180515.H089

```
public Rook (int size) { super(size);
  cap = new Rectangle (2*size/3, size/3);
}
public void drawCap(Graphics g) {
    g.drawRect(size/6, size/6, 2*size/3, size/3);
}
                                                  //
```

Name and, if possible, ID#:

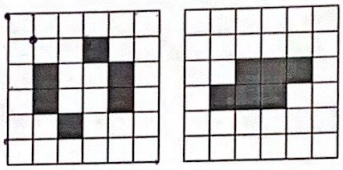
Problem 3

Consider the famous **Game of Life** cellular automaton – a two-dimensional square grid of cells, each of which can appear in one of two possible states: **alive** – ***true***, or **dead** – ***false***. At each time step called **tick** all cells are updated depending on ***8*** neighbors adjacent horizontally, vertically or diagonally, as follows:

- An alive (***true***) cell dies (becomes ***false***), if it has less than ***2*** or more than ***3*** live neighbors;
- An alive (***true***) cell remains alive, if it has ***2*** or ***3*** alive neighbors;
- A dead (***false***) cell becomes alive (***true***), if it has exactly ***3*** alive neighbors.

Complete a Java ***public class Life*** that extends ***public class Animator*** and animates the **Game of Life**. It encapsulates a ***100-by-100 private boolean grid[][]*** and initializes it randomly. Your task is to implement the methods ***public boolean tick()*** and ***public void snapshot(Graphics g)***. Draw squares for dead cells and fill squares – for alive ones. Use the methods ***g.drawRect(int topLeftX, int topLeftY, int width, int height)*** and ***g.fillRect(int topLeftX, int topLeftY, int width, int height)***. Use the default cell size = ***4***. You may also use a method ***private int sum9(int row, int col)*** that returns the number of alive neighbors of a cell at the specified ***int row*** and ***int col***.

An example of an initial state is shown in the left figure. The right figure depicts the state after one tick.



```
public class Animator extends JApplet {

      public boolean tick() {
//TO BE OVERRIDEN IN LIFE CLASS
            return true;
      }

      public void snapshot(Graphics g) {
//TO BE OVERRIDEN IN LIFE CLASS
      }

      public void delay(int lag) {
            if (lag > 0) {
                  delay(lag - 1);
                  delay(lag - 1);
            }
      }

      public void paint(Graphics g) {
            g.setColor(Color.WHITE);
            g.fillRect(0, 0, getWidth(), getHeight());
            g.setColor(Color.BLACK);
            snapshot(g);
            if (tick()) {
                  delay(25);
                  repaint();
            }
      }
}
```

(public class Life is shown on the next page)

OOP. FI. 180515. 14089

```
for(int row=0; row<grid.length; row++) {
    for(int col=0; col<grid[0].length; col++) {
if((grid[row][col]==1 && sum9(row,col)==2) ||
   grid[row][col]==1 && sum9(row, col==3)) {
return grid[row][col]=1;
}
else if (grid[row][col]==0 && sum9(row,col)==3){
return   grid[row][col]=7;
}
else ((grid[row][col]==1 && sum9(row,col)<2) ||
return grid[row,col]==1&& sum9(row,col)>3)) {
        grid[row][col]=0;
}
return true;
}
public void snapshot(Graphics g) {
for(int row=0; row<grid.length; row++) {
    for(int col=0; col<grid[0].length; col++) {
        g.setColor(Color.BLACK);
        g.drawRect(col, row, 4,4);
if(grid[row][col]==1) {
        g.setColor(Color.BLACK);
    g.fillRect(col,row,4,4);
}
else if(grid[row][col]==0) {
    g.setColor(Color.WHITE);
    g.fillRect(col,row,4,4);
}
}
}
}
```

5