Name and, if possible, ID#.____

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

FINAL EXAM

9/15

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

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 integral in the specified range from x_1 to x_2 using the approximation:

$$\int_{x1}^{x2} f(x)dx \approx \frac{x_2 - x_1}{6}\left(f(x_1) + 4f\left(\frac{x_1 + x_2}{2}\right) + f(x_2)\right)$$

```
public class Function {

      private Valuable f;
      private double dx;
      public Function(Valuable newValuable, double newDX) {
            //TO BE IMPLEMENTED
      }

      public double integral(double x1, double x2) {
            //TO BE IMPLEMENTED
      }
```

1.2 Implement an expression

$$\sqrt{x^2 + a} + \sqrt{x^2 + b}$$

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

1.3 In a separate ***public static void main(String args[])*** write a code that inputs two double values, creates an object of type ***Roots*** and, using the class ***Function***, prints the value of its integral from $x_1 = 1.0$ to $x_1 = 2.0$:

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

      //TO BE COMPLETED

}
```

OOP. FT. 180315. M110

```
private Valuable f;
private a;
private b;
public Roots(double newA, double newB){
        ? { f.value(double x){
                return sqrt(x*x+newA)+sqrt(x*x+newB);
            }
    }
}
```

1.3)
```
public static void main(String args[]){
    Roots obj = new Roots(a, b);
    Function obj1 = new Function();
    obj.integral(1.0, 2.0)
}
```

3

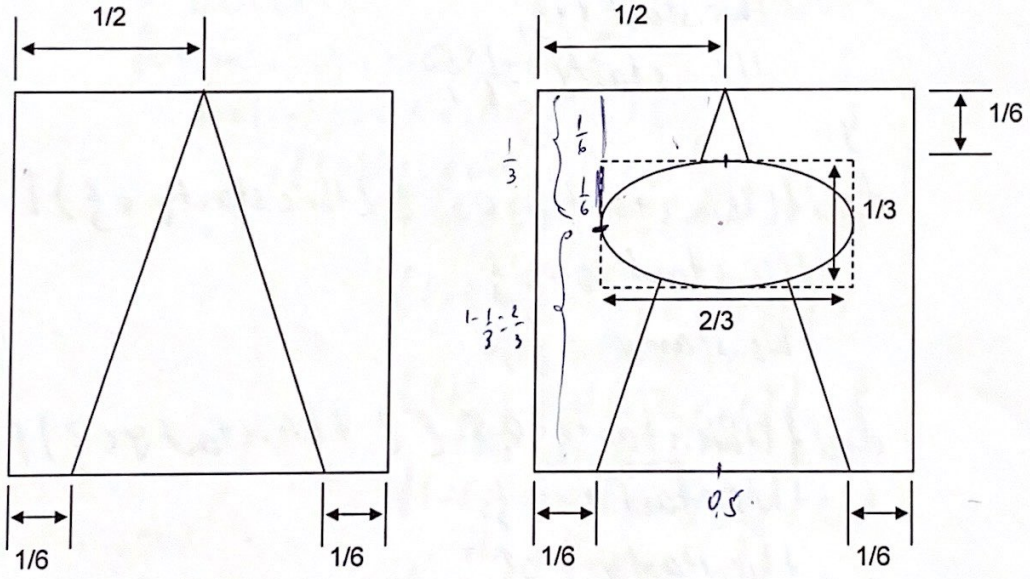
Name and, if possible, ID#:_____

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 Bishop 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 bishop are shown below:



Use the backside, if needed

OOP.FT.180315.M110

```
public class Bishop extends class Piece {
    private Rectangle cap;
    public Bishop(int size){
super parent.drawBase(size);
    parent.draw(Graphics.g);
}
public void drawCap(Graphics g) {
    private startx = 1/6
    private starty = 2/3
? for(int i=0; i <1000, i++){
        g.drawRect(this.startx, this.starty, 0, 0);
        if(this.startx<0,5 && this.starty >2/3){
            this.startx += 1/3;
            this.starty += 1/6;
        }
        else if(this.startx>0,5 && this.starty> 2/3){
            this.startx += 1/3;
            this.starty -= 1/6;
        }
        else if(this.startx>0,5 && this.starty< 2/3){
            this.startx -= 1/3;
            this.starty -= 1/6;
        }
        else if(this.startx<0,5 && this.starty< 2/3){
            this.startx -= 1/3;
            this.starty += 1/6;
        }
        else
            break;
    }
}
```

Name and, if possible, ID#: _____

```
public class Life extends Animator {

      private boolean grid[][] = new boolean[100][100];
      private int cellSize = 4;

      public void init() {
            for (int row = 0; row < grid.length; row++)
                  for (int col = 0; col < grid[0].length; col++)
                        grid[row][col] = Math.random() < 0.5;
      }

      private int sum9(int row, int col) {
            int result = grid[row][col] ? -1 : 0;

            for (int i = Math.max(0, row - 1);
                        i < Math.min(grid.length - 1, row + 1); i++)
                  for (int j = Math.max(0, col - 1);
                              j < Math.min(grid[0].length - 1, col+ 1); j++)
                        result += grid[i][j] ? 1 : 0;

            return result;
      }

      public boolean tick() {
            //TO BE IMPLEMENTED
      }

      public void snapshot(Graphics g) {
            //TO BE IMPLEMENTED
      }
}
```

```
public boolean tick(){
    int alives=0;
    for(int i=0, i<100, i++){
        for(int j=0,j<100,j++){
            if(this.grid[i][j]{
                if(this.grid[i+1][j])
                    this.alives++;
                if(this.grid[i+1][j+1])
                    this.alives++;
                if(this.grid[i][j+1])
                    this.alives++;
                if(this.grid[i-1][j+1])
                    this.alives++;
                if(this.grid[i-1][j])
                    this.alives++;
                if(this.grid[i-1][j-1])
                    this.alives++;
                if(this.grid[i][j-1])
                    this.alives++;
```

is done in sum9()

OOP.FT.180315.mtw

Use the backside, if needed

```
if(this.grid[i+1][s-1]
    this.alives++;
if (this.alives == 2|| this.alives == 3)
    this.grid[i][s] = True;
else{
    this.grid[i][s] = False;
```

Incomplete conditions

```
    }
  }
 }
}
public void snapshot(Graphics g){
    g.setColor(Color.White);
    g.fillRect(0,0, getWidth(),getHeight());
    g.setColor(Color.Black);
    for(int i=0; i<100; i++){
      for(int s=0, s<100, s++){
          if(this.grid[i][s] == False)
              g.fillRect(i, s, 4, 4);
          else{
              g.setColor(Color.White)
              g.fillRect(i, s, 4, 4);
          }
      }
    }
}
```