

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

19/15

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:

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

```
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 $x_2 = 2.0$:

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

Use the backside, if needed

Page 1 of 4

OOP.FT.180515.H093

... a new Valuable, double newDX;

```

{
    f = new Valuable();
    dx = new DX();
}
public double integral (double x1, double x2)
{
    return (x2 - x1) / 6 * (f.Value(x1) + 4 * f.Value((x1 + x2) / 2) +
    + f.Value(x2));
}

```

1.2 public class Roots implements Valuable {
 private double a, b;
 public Roots (double newA, double newB) {
 a = newA;
 b = newB;
 }
 public double Value (double x) {
 return (Math.sqrt(x * x + a) + Math.sqrt(x * x + b));
 }
}

5

1.3 Valuable R = new Roots (a, b);
 Function F = new Function (R, 0);
 System.out.println (F.integral (1.0, 2.0));

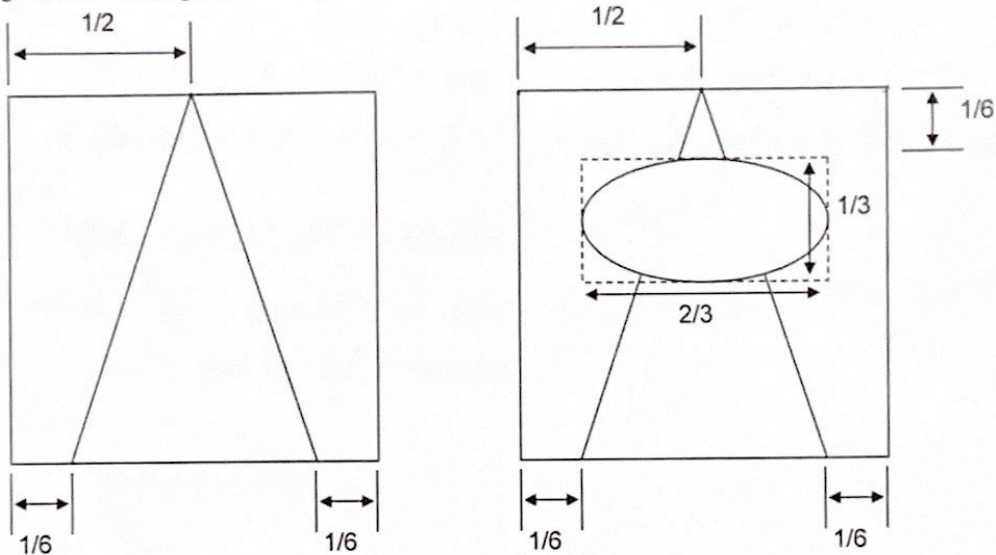
Name and, if possible, ID#:

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 shown:

```
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:



piece {

```
private Rectangle cap;  
public Bishop() { super(size);  
public Bishop(int x, int y, int width, int height)  
    {  
        int x = field.x + (int)(1.16 * field.width);  
        int y = field.y + (int)(1.16 * field.height);  
        public Bishop  
            int width = (int)(2.13 * field.width);  
            int height = (int)(1.13 * field.height);  
        public Bishop(x, y, width, height) {  
            cap = new Rectangle(x, y, width, height);  
        }  
    }  
}
```

```
public void drawCap(Graphics g) {  
    g.drawOval(cap.x, cap.y, cap.width, cap.height);  
}
```

4

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() {  
        for (int row = 0; row < grid.length; row++)  
            for (int col = 0; col < grid[0].length; col++) {  
                if (sum9(row, col) < 2 || sum9(row, col) > 3 && grid[row][col]  
                    == true)  
                    grid[row][col] = false;  
                else if (sum9(row, col) == 3 && grid[row][col] == false)  
                    grid[row][col] = true;  
            }  
        return true;  
    }
```

```
    public void snapshot(Graphics g) { int x = 0; int y = 0;  
        for (int row = 0; row < grid.length; row++) {  
            for (int col = 0; col < grid[0].length; col++) {  
                if (grid[row][col] == true)  
                    g.fillRect(x, y, 4, 4);  
                else  
                    g.drawRect(x, y, 4, 4);  
            }  
            x += 4;  
            y += 4;  
        }  
    }
```

Use the backside, if needed

5

ODP.FT.180515.H093