Name and, if possible, ID#:

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

FINAL EXAM

14/15

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

}
```

Use the backside, if needed

OOP.FT. 180515 · 4090

```
        1.);
    }
public class Function {
    private Valuable f;
    private double dx;
    public Function(Valuable newValuable, double newDx){
        f = newValuable;
        dx = newDx;
    }
public double integral(double x1, double x2) {
        double integ;
        integ = ((x2-x1)/6)*(f.value(x1) + 4*
        * f.value((x1+x2)/2) + f.value(x2));
        return integ;
    }
}
public class Roots implements Valuable {
    private double a, double b;
    public Roots(double newA, double newB) {
        a = newA;
        b = newB;
    }
    public double value(double x) {
        double value = 0;
        value = Math.sqrt(x*x + get(A)) + Math.sqrt(x*x+get(B));
        return value;
    }
public double getA() {
    return a;
}
public double getB() {
    return b;
}
```

3) public sta

Scanner input = ...

double a = input.nextDouble();
double b = input.nextDouble();
double x1 = 1.0;
double x2 = 2.0;
public Valuable v = new Roots(a, b);
public Function f = new Function(v, 0);
double a = f.integral(x1, x2);
System.out.println(a);
}

5

OOP. HT. 1805.15. H090

Mariam Hakobyan

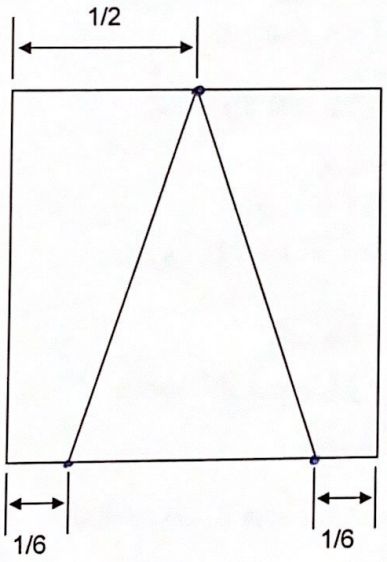
Name and, if possible, ID#:

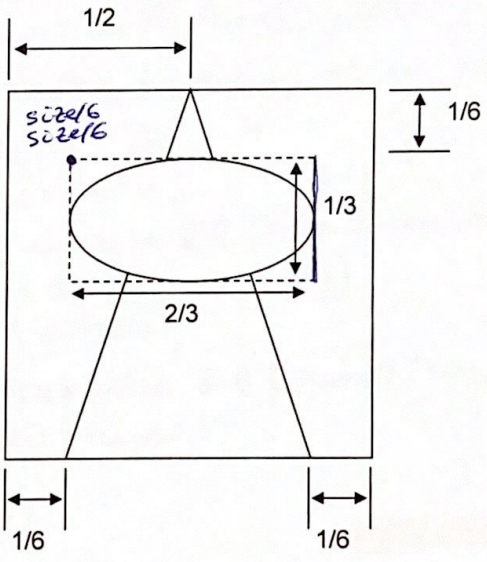
Problem 2

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.180515.1090

```
Chesspiece {
private Reetangle cap;
  public Bishop( size) {
    super(size)
    cap = new Reetangle((2 * size)/3 , size/3)
cap = new Reetangle( size/6, size/6, (2*size)/3,
                                    size/3)
  }
public void drawCap( Graphics g) {
    g.drawOval (size/6, size/6, (2 * size)/3,
                                    size/3)
  g. drawOval( cap.x, cap.y, cap.width, cap.height)
}.
```

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(grid[row][col]==true && (sum9(row,col)<2 ||
                sum9(row,col)>3))
            grid[row][col]=false;
        else if(grid[row][col] == true && (sum9(row,col)==2 ||
                sum9(row,col)==3))
            grid[row][col] = true;
        else if (grid[row][col]==false && (sum9(row,col)==3))
            grid[row][col]=true;

    return grid[row][col];
}
public void snapshot(Graphics g){
    for(int row=0; row<grid.length; row++)
        for(int col=0; col<grid[0].length; col++)
        if(grid[row][col] == false)
            g.drawRect(col*cellSize, row*cellSize, cellSize, cellSize)
        else
            g.fillRect(col*cellSize, row*cellsize, cellsize, cellsize)
    }
}
```

5

OOP.FT.180515.HW10

Use the backside, if needed