Name and, if possible, ID#:_

AMERICAN UNIVERSITY OF ARMENIA

College of Science and Engineering

COMP120 Introduction to Object-Oriented Programming

FINAL EXAM

8/15

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

}
```

3

Use the backside, if needed

OOP.FT.180515.114

```
public class function{
  private valuable F;
  private double dx;
public function (valuable newValuable, double newDX){
  this.F= newValuable
  this.dx= newDX;
  }
public double Integral (double X1, double X2){
 return((X2-X1)/6)*(F(X1)+4*F((X1+X2)/2)+F(X2));
 }
1,2 Public class Roots {
 Private double a;
 Private double b;
 Public Roots (double newA, double new B){
 this.a = newA; this.b= new B; }
                    value
 Public double expression(){
 return = Math.sqrt(x*x+a) + Math.sqrt(x*x+b); }
1,3 public static void main(String args[]) {
 ~~double x1= 1.0;~~
 ~~double x2= 2.0;~~
 scanner input = new Scanner (System.in);
 double a = input.nextDouble(),b=input.nxtDouble();
  Objs = new Roots(a,b);
  double x1= 1.0; double x2=2.0;
  Function z = new Funtion();
  z.Function();
 }
```

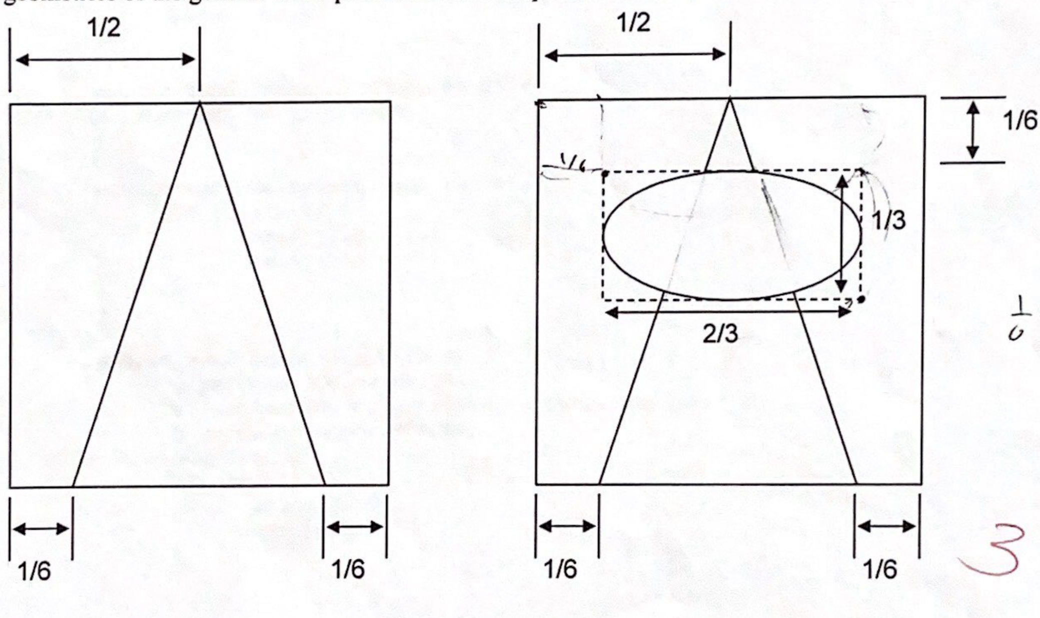
Name and, if possible, ID#:__

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:



$\frac{1}{6} + 3$

3

Use the backside, if needed

OOP.FT. 180515.114

```
e {

public void drawCap( Graphics g){
cap = new Rectangle( size * 2/3, size * 1/3));
g.drawOval( size* 1/6 , size * 1/6, size * 5/6, size * 1/2);
g. setColor( Color.White);
g. FillOval( size* 1/6, size* 1/6, size* 5/6, size* 1/2))
}
```

public Bishop()-?

Name and, if possible, ID#:__

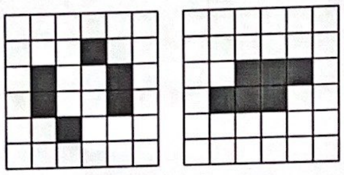
Problem 3

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-FT, 180515.114

```
[illegible]
private int sumg(int row, int col);
if(((sumg< 2) & (sumg > 3)) || ((sum S==2) || (sum g== 3))) [illegible]

    return true;
} else
}   return false)

  }
  }


public void snapshot(Graphic g){
g. setColor(Color.black);   (i)         (g)                (4)             (4)
g. drawRect(int topLeftx, int topLeftY, int width, int height);
                           (i)             (g)       (4)          (4)
g.fillRect(int topLeftx, int topLeftY, int width, int height);

}

}



for ( int i=0; i<100; i++)
    for int g=0; g<100; g++) {
      ;
      ;
```

(2 Second 52 [illegible] 5
[illegible] for [illegible])

2