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 max in the specified range from x_1 to x_2 by looking at:
$f(x_1), f(x_1+dx), f(x_1+2dx), \ldots, f(x_1+k*dx)$, where $k = 1, 2, \ldots$ and $x_1+k*dx < x_2$

```
public class Function {

      private Valuable f;
      private double dx;

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

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

1.2 Implement an expression

$$a * sin(x) + b * cos(x)$$

as a ***public class Harmonic*** that implements the interface ***Valuable*** and encapsulates double parameters ***a*** and ***b***. The parameters are initialized by the two-argument constructor ***public Harmonic(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 ***Harmonic*** and, using the class ***Function***, prints its maximal value in the range from $x_1 = -1.5$ to $x_1 = 1.5$:

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

      //TO BE COMPLETED

}
```

4

OOP.FT.180515.M116

Use the backside, if needed

```
public double value(double x)
};
class Function
{
  private Valuable f;
  private double dx;
  public Function (Valuable newValuable; double newDX){
      dx = dx;
      f = newValuable;
  }
  public double max(double x1, double x2)
  {
      double max = f(dx);
          int k = 1;
      while (x1 + k*dx < x2){
      double temp = f(x1 + k * dx);
      ;f (temp > max ){
       max = temp;
      }
        k++;
    }  return max;
  }
}
```

~~Public cal~~).

```
public class Harmonic {
    implements Valuable;
    private double a;
    private double b;
    public Harmonic (double newA; double new B) {
        a = newA;
        b = newB;
    }
    public double value (double x)
    {
        return a * sin(x) + b * cos(x) ~~sin(x)~~;
    }
}
```

13)

```
public static void main (String args[]) {
    Scanner input = new Scanner (System.in);
    double a = input.nextDouble(), b = input.nextDouble();
    Harmonic h = new Harmonic (~~double~~ a, b);
    Function f = new Function (h, 0,5);
    double max = f.max (-1,5 , 1,5);
    System.out.print ("Max =");
    System.out.print (max);
}
```

OOP.PT.180515-111116

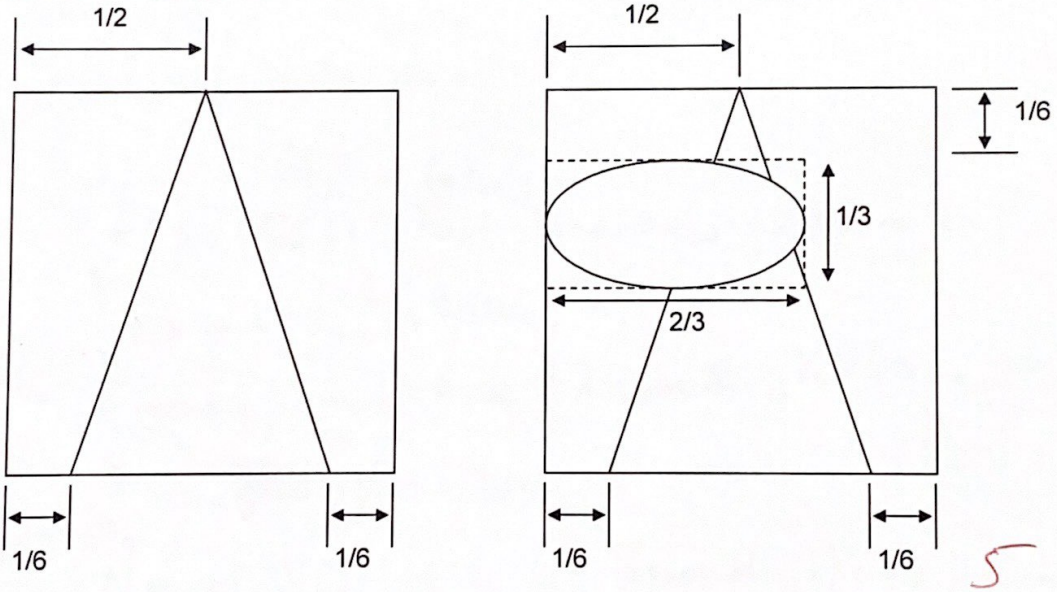
Name and, if possible, ID#: _____

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 Knight 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 knight are shown below:



5

Use the backside, if needed

OOP. FT. 180515. M116

```
public class Knight extends ChessPiece {
    private Rectangle cap;
    public Knight (int size) : ChessPiece (size) {
        cap = new Rectangle (size/2, size);
    }
    public void drawCap(Graphics g)
    {
    ChessPiece.draw(g);
    g.drawOval.x(cap.x, cap.y, 2*size/3; size/3);
    }
}
```

super(size)

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
      }
}
```

0

see SM, KG, GJ, HC

```
int i, j;
for(i=0; i<100, i++)
 for(j=0; j<100; j++)
 {
  newgrid[i][j] = false;
  if(grid[i][j]==true && sum9(i,j)<2 || sum 9(i,j)>3)
  { newgrid[i][j]= false;
  }
  if(grid[i][j]==true && sum 9(i,j)<2 || sum 9(i,j)==3)
  { newgrid[i][j] = true;
  }
  if(grid[i][j]==true && sum 9(i,j)==2 || sum9(i,j)==3)
  { newgrid[i][j] = true;
  }
  if(grid[i][j]==false && sum 9(i,j)==3)
  { newgrid[i][j]= true;
  }
}
grid = newgrid;
```

OOP. FT. 170515. M116

Use the backside, if needed