Name and, if possible, ID#:

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

FINAL EXAM

5/15

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
      }
```

f = newValuable;
dx = newdx;
(on the next page)

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

}
```

1 ✓
see GH

Use the backside, if needed

OOP. FT. 180515. M115

```
double max = x1;
f.value(x1 + k*dx);
if (max < f.value(x1 + k*dx))
{
    max = f.value(x1 + k*dx);
}
return max;
```

1.3

```
public class Harmonic
{
    double a, b;
    public Harmonic(double newA, double newB)
    {
        a = newA;
        b = newB;
    }
    public Compute(double x)
    {
        return((a * Math.sin(double x)) + b * (Math.sin(double x)));
    }
}
```

~~public static void main(String[] args)~~

~~{~~

Name and, if possible, ID#:_____

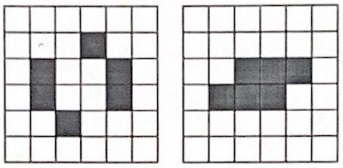
Problem 3

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.



4

```
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.mdl5

```
for (int row=0; row<100; row++)
  for (int col; col<100; col++)
  { if (grid[row][col] == true)
      {if (sum(row,col)<2 || sum(row,col)>3)
          grid[row][col] == false )
          }
      { if (grid[row][col] == true)
          {; if (sum(row,col)=2 || sum(row,col)=3){
              grid[row,col] == true
          }
      {if ( grid[row][col]== false )
          if (sum(row,col)=3){
              grid[row][col] == false )

      }

public void snapshot()
{  if (grid[row][col] == true)
   {g.fillRect(int row, int col, 4, 4);
    }
    else
    { [illegible]
          g.drawRect(int row, int col, 4, 4)
      }

   }
```