שאלה 1.1:

a) The JUnit framework is using a set of tags, in particular, the "@Test" tag.
b) The system is running each testing method (marked by @Test) and logs the result of each testing method.
c) Each testing method is run (as a thread) and may result in one of the following three states: (i) pass, (ii) "fail" or (iii) Error (exception).
d) Once a method is done running, its status is logged: pass (green), fail (blue), or error (red), the runtime is logged as well.
e) After running all the testing methods, the log is presented.
f) The timeout mechanism (e.g., @Timeout(value = 1000, unit = TimeUnit.MILLISECONDS)) is performed using a timer-like thread which can cause an exception if the time period of the timer is done.

שאלה 1.2:

```
public class GraphTest {
    @Timeout(value = 1000, unit = TimeUnit.MILLISECONDS)
    @Test
    public void graphTest_Q1_2() {
        int v_size = 1000;
        int e_size = 10000;
        Graph g = new Graph();
        for(int i=0;i<v_size;i=i+1) {g.addNode(i);}
        while(g.edgeSize()<e_size) {
            int a = (int)(Math.random()*v_size);
            int b = (int)(Math.random()*v_size);
            if(a!=b && g.getEdge(a,b) == null) {
                g.connect(a,b,1);
            }
        }
    }
```

שאלה 2.1:

This class represents a Memory - Graph Algorithm, it extends the GraphAlgo, and stores the algorithm results in a hashMap<Method_Input_MC, Output>:
In case a method (algorithm) is recalled (the same method, input, graph) instead of computing the algorithm - the stored results is returned.
Else (first time - Method_Input_MC), the method (algorithm) is being called anf the result is being stored in the hash map (aka hash-table).
Note: A String is used for concatenating the "Method_Input_MC" – note: no parsing is needed.

שאלה 2.2.

```
import java.util.HashMap;
/**
 * This class represents a Memory - Graph Algorithm, it extends the GraphAlgo,
 * and stores the algorithm results in a hashMap<Methos_Input_MC, Output> so in
case the method (shortestPath) is
 * recalled (the same method, input, graph) instead of computing the algorithm -
the stored results is returned.
 * The implementation below oncludes a single method implementation (shortestPath),
yet it can be extended to other methods.
 */
public class GraphAlgoMemory extends GraphAlgo {
    private HashMap<String, String> _memory;
    public GraphAlgoMemory() {super();
        _memory = new HashMap<String, String>();
    }
    @Override
    public double shortestPath(int src, int dest) {
        String key = "shortestPath_"+src+","+dest+",MC_"+ super.getGraph().getMC();
        String v = this._memory.get(key);
        double ans = -1;
        if (v!=null) {ans = Double.parseDouble(v);}
        else {
            ans = super.shortestPath(src, dest);
            _memory.put(key, ""+ans);
        }
        return ans;
    }
}
```

שאלה 3.1:

בשאלה זו לא נדרשתם לכתוב קוד, הפתרון מובא ב"פסאודו קוד" כדי לאפשר קריאות טובה יותר.

```
In general, the ServerMultiCleints will be implementing GraphInterface, and will
contain (be composed from) a Server object.
Each public method of GraphInterface will be synchronize using the _server object -
see below)
Once a ServerMultiCleints object is being constructed, several threads can safely
call it - as it is now thread-safe.

class Server implements GraphInterface
public class ServerMultiCleints implements GraphInterface{
       private GraphInterface _server;
       ServerMultiCleints(GraphInterface g) {_server = new Server(g);}

       for each method in the interface:
       @Override
       public boolean hasNode(int key) {
          synchronized(_server) {return _server.addNode(key);}}
       // …
       @Override
       public <return value> method_n(<parameters_n>) {
          synchronized(_server) {return _server.method_n(parameters_n);}}
```

שאלה 3.2:

```
* A simple solution is to perform a "sleep(10)", with in each method call:
* public class ServerMultiCleints_100 implements GraphInterface{
*      private GraphInterface _server;
*      private Integer _count;
*      public ServerMultiCleints_100(GraphInterface g) {
*         _count = 0; _server = new Server(g);}
*      @Override
*      public boolean hasNode(int key) {
*              inc();
*              synchronized(_server) {
*                  "sleep(10)"; // mili seconds.
*                  dec();
*                  return _server.addNode(key);
*              }
*      }
*      private void inc() {
*             synchronized(_count) {
*                   _count++;
*                   if(_count>100) {throw new RuntimeException("ERROR: …");}
*                }
*         }
*      private void dec() {synchronized(_count) {_count--;}};
*         // ........
* }
```

שאלה 4.1:

```
# *************** Q4.1 ******************
    def __gt__(self, other):  # class (Medic)
        return self. get_salary() > other.get_salary()

class Nurse(Medic):
    def __init__(self, name: str, family_name: str, id: str, experience: int):
       super().__init__(name,  family_name, id, experience)
    def get_salary(self):
        return 8000 + self.experience*800

class Doctor(Medic):
    def __init__(self, name: str, family_name: str, id: str, experience: int):
       super().__init__(name,  family_name, id, experience)
    def get_salary(self):
        return 12000 + self.experience*1000
```

שאלה 4.2.

```
# *************** Q4.2 ******************
class TestMedic(unittest.TestCase):
    def test_sun(self):
        nurse = Nurse("Noa", "Levi", "1223", experience=2)
        doc = Doctor("Michael", "Wag", "1111", experience=2)
        m = MedicPayroll()
        m.add_medic(nurse)
        m.add_medic(doc)
        sum = m.get_all_month_salary()
        req = 12000+2*1000 + 8000+2*800
        self.assertEqual(sum, req," ERR wrong sum of all salaries")

    def test_max(self):
        nurse = Nurse("Noa", "Levi", "1223", experience=20)
        doc = Doctor("Michael", "Wag", "1111", experience=2)
        m = MedicPayroll()
        m.add_medic(nurse)
        m.add_medic(doc)
        self.assertEqual(nurse, m.get_most_expensive_medic(), "ERR: wrong max...")
```