גיליון תשובות

מספר נבחן:_____

שאלה 1 (30 נקודות)

סעיף א (20 נקודות)

(i)

```
//@PRE: checkInv(other.getX(),other.getY(),other.getRadius())
//@POST: getRadius() – other.getRadius() >=
//          Math.max(Math.abs(getY()-other.getY()),Math.abs(getX()-   other.getX()))
void union(Circle other) throws Exception;
```

(ii)

```
public synchronized void union(Circle other) throws Exception {
    if (!checkInv(other.getX(),other.getY(),other.getRadius()))
      throw new Exception("Illegal circle: " + other);
    synchronized (other) {
      setRadius(calculateUnionRadius(_x,_y,_radius,other.getX(),other.getY(),other.getRadius()));
    }
}
```

(iii)

```
Circle c1,c2;
…
T1: c1.union(c2);
T2: c2.union(c1);
```

T1 נועל את this (c1) ולפני שהספיק לנעול את other (c2), עובר זמן ה cpu ל T2.
T2 נועל את this (c2) ונכנס למצב blocked בעקבות ניסיון נעילת c1, התפוס ע"י T1.
T1 נכנס למצב blocked בעקבות ניסיון נעילת c2, התפוס ע"י T2.

(iv)

We simply implement the resource ordering technique, exactly as taught in class:

```
public void union(Circle other) throws Exception {
    if (!checkInv(other.getX(),other.getY(),other.getRadius()))
      throw new Exception("Illegal circle: " + other);
    if (System.identityHashCode(this) > System.identityHashCode(other))
      CircleImpl.union(this,other,this,other);
    else
      CircleImpl.union(other,this,this,other);
}

protected static void union(Object synch1, Object synch2, CircleImpl c1, Circle c2) throws Exception {
    // synch1 and synch2 determine the order of the locking
    // c1 and c2 determine the order, in terms of the union logic (c1 should cover c2)
    synchronized (synch1) {
      synchronized (synch2) {
        c1.setRadius(
          c1.calculateUnionRadius(c1.getX(),c1.getY(),c1.getRadius(),c2.getX(),c2.getY(),c2.getRadius()));
      }
    }
}
```

סעיף ב (10 נקודות)

ניישם את תבנית reader/writer כפי שנלמדה בהרצאות:

- במחלקה CircleImpl נחליף את הסנכרון ב semaphore
- נעדכן את המחלקה Semaphore כך שתממש את מדיניות הכניסה לאובייקט כתנאי עבור ה wait

```
class CircleImpl implements Circle {
  private int _x;
  private int _y;
  private int _radius;
  private Semaphore _sem;

  CircleImpl(int x, int y, int radius) throws Exception {
    if (!checkInv(x,y,radius))
      throw new Exception();
    _x = x;
    _y = y;
    _radius = radius;
    _sem = new Semaphore();
```

```
}

public synchronized int getX() throws InterruptedException  {
   int ret = 0;
   _sem.acquireRead();
   ret = _x;
   _sem. releaseRead();
   return ret;
}
public synchronized int getY()    throws InterruptedException  {
   int ret = 0;
   _sem.acquireRead();
   ret = _y;
   _sem. releaseRead();
   return ret;
}
public synchronized int getRadius()  throws InterruptedException  {
     int ret = 0;
   _sem.acquireRead();
   ret = _radius;
   _sem. releaseRead();
   return ret;
}

public synchronized void setCenter(int x, int y)    throws Exception {
   _sem.acquireWrite();
  if (!checkInv(x, y, _radius)) {
     _sem. releaseWrite();
     throw new Exception();
   }
   _x = x;
   _y = y;
   _sem. releaseWrite();
}
public synchronized void setRadius(int radius)  throws Exception  {
   _sem.acquireWrite();
  if (!checkInv(_x, _y, radius)) {
     _sem. releaseWrite();
     throw new Exception();
   }
   _radius = radius;
   _sem. releaseWrite();
}
```

```
  protected boolean checkInv(int x, int y, int radius) {
    return getRadius() >= 0 && getX() >= getRadius() && getY() >= getRadius();
  }
}
class  Semaphore {
   protected int activeReaders_;
   protected int activeWriters_;

   public Semaphore() {
      activeReaders_ = 0;
      activeWriters_ = 0;
   }

   public synchronized void acquireRead() throws InterruptedException {
      while (activeWriters_>0)
         this.wait();
      activeReaders_++;
   }

   public synchronized void acquireWrite() throws InterruptedException {
      while (activeReaders_> 0 || activeWriters_>0)
         this.wait();
      activeWriters_++;
   }

   public synchronized void releaseRead() throws InterruptedException {
      if (activeReaders_ > 0) {
         activeReaders_--;
         this.notifyAll();
      }
   }
   public synchronized void releaseWrite() throws InterruptedException {
      if (activeWriters_ > 0) {
         activeWriters_--;
         this.notifyAll();
      }
   }
}
```

שאלה 2 (30 נקודות)

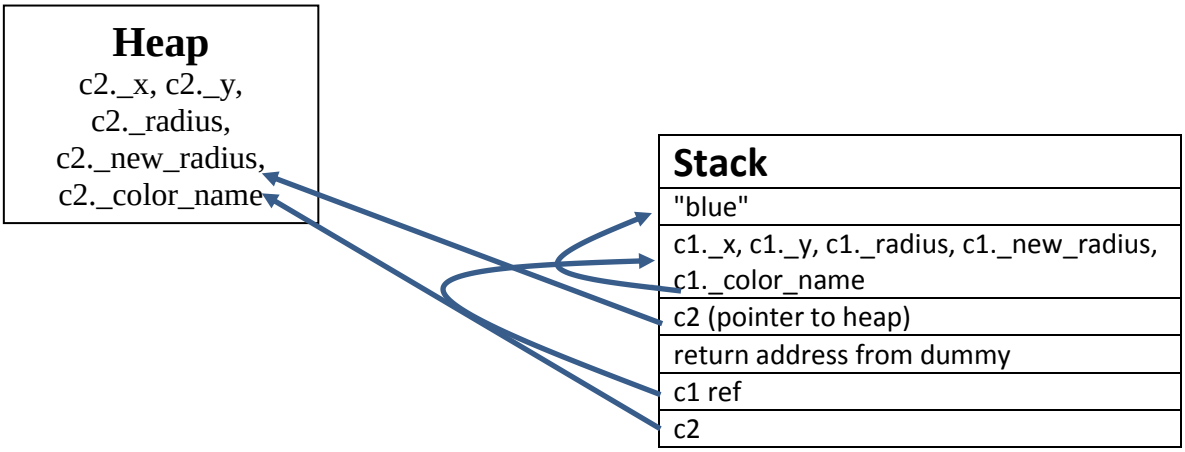
.א

```
class Circle{
public:
```

```
        Circle():_x(0),_y(0),_radius(0){;}
        Circle(int x, int y, int r):_x(x),_y(y),_radius(r){;}
        virtual ~Circle(){;}

        int getX() const {return _x;}
        int getY() const {return _y;}
        int getRadius() const {return _radius;}

        void setX(int x)  {_x=x;}
        void setY(int y) {_y=y;}

        virtual void setCenter(int x, int y){_x=x;_y=y;}
        virtual void setRadius(int radius){_radius=radius;}

private:
        int _x;
        int _y;
        int _radius;
};


class BubblesCircle: public Circle{
public:
        BubblesCircle():_color_name(0){;}
        BubblesCircle(int x, int y, int radius, int newradius, const char* color)
            :Circle(x,y, radius),_new_radius(newradius),_color_name(color){;}
        ~BubblesCircle(){;}
        BubblesCircle(const BubblesCircle& b)
           :Circle(b.getX(), b.getY(), b.getRadius()), _new_radius(b._new_radius),_color_name(b._color_name){;}
        BubblesCircle& operator=(const BubblesCircle& b) {
          if (this != &b) {
            setX(b.getX());
            setY(b.getY());
            setRadius(b.getRadius());
           _new_radius = b._new_radius;
           _color_name = b._color_name;
          }
          return *this;
        }

        virtual void setRadius(int p){_new_radius = p;}
        virtual int  getRadius() const {return _new_radius;}
        const char*  getName() {return _color_name;}
        void setName(const char *name) {_color_name=name;}
```

```
private:
        int     _new_radius;
        const char *_color_name;
};
```

ב.



ג2.

```
void ReplaceColor(vector<BubblesCircle> &set, const char *s, const char *t)
{
        for (vector<BubblesCircle>::iterator& it = set.begin(); it != set.end(); it++)
        {
                if ((*it).getName()== s)
                        (*it).setName(t);
        }
}
```

מפתח ניקוד:
2א. מימוש תקין של כל אחת מהפונקציות שווה 2נק.
2ב. כניסה חסרה בטבלה או חוסר עקביות עם מימוש של סעיף א: -2
2ג. אי מעבר על המערך: -4
2ג. אי בדיקה של צבע מקור/ אי החלפה לצבע יעד -3

שאלה 3 (30 נקודות)

סעיף א (5 נקודות)

Nothing special here – just make sure to follow the regular RMI rules
http://www.cs.bgu.ac.il/~spl121/RMI

```
public interface IndexService
        extends java.rmi.Remote {
        public List<String> search(String query)
           throws java.rmi.RemoteException;
    }
```

And add a note that we hope List<String> is a serializable Object.

סעיף ב (5 נקודות)

We make sure that IndexServiceImpl meets RMI requirements:
- Implements the service interface IndexService
- Extends the RMI UnicastRemoteObject class
- Throws RemoteException in its constructor

And that we meet the new requirements:
- Each IndexServiceImpl is responsible for a single index file (as opposed to one service that accesses all the index files in the code provided in the question).

So, we copy the relevant code from what was provided in the question and do the following adjustments:
- Remove the map _indexFiles and replace it by a single _indexFile.
- Pass the index file as a parameter of the constructor.

A good optimization to do in this case is to read the whole file in RAM only once instead of reading the file each time a query is processed. But this is beyond the scope of what this question checked.

```
public class IndexServiceImpl implements IndexService
   extends java.rmi.UnicastRemoteObject {

  private  File _indexFile;
  ...
   public IndexServiceImpl(String fileName) throws java.rmi.RemoteException {
      _indexFile = new File(filename);
   }

   public List<String> search(String query) throws java.rmi.RemoteException {
     try {
       List<Byte> bytes = new LinkedList<Byte>();
       FileInputStream in = new FileInputStream(_indexFile);
      int b = -1;
       while ((b = in.read()) != -1)
          bytes.add((byte)b);
```

```
        return search1(word, bytes);
      } catch (Exception e) {
        return new LinkedList<String>();
      }
    }
    // rest unchanged…
}
```

סעיף ג (5 נקודות)

We must register each IndexService with a different name – so that the rmi clients can decide to which instance of IndexService they connect. We will adopt the same convention as in the original code: each index file is identified by a single letter. So we use a single letter as part of the RMI name published by each service.

The invocation of this server will be like this:

java IndexServer silver.bgu.ac.il 10212 /usr/index/a.idx a

which will publish in the rmi naming server running on silver:10212 the following RMI name: IndexServer-a

```
class IndexServer {
   public static void main(String[] args) {
     // rmiHost and rmiPort give us the address of the rmiRegistry name server.
     String rmiHost = args[0];
     int rmiPort = Integer.parseInt(args[1]);

     // The letter that identifies the index file and the path of the index file are also provided as args
    String indexFileName = args[2];
    String indexName = args[3];

     IndexService service = new IndexServiceImpl(indexFileName);
     String rmiName = "IndexServer-" + indexName;
      Naming.rebind("rmi://" + rmiHost + ":" + rmiPort + "/" + rmiName);
      System.out.println("Index server " + rmiName + " for file " + indexFileName + " is ready…");
   }
}
```

סעיף ד (10 נקודות)

We only need to change the implementation of SearchService. The Reactor code can remain the same generic code we use as usual and the protocol is also unchanged – except that we replace SearchService with SearchServiceWithRMI which has the same interface – search(word).

```
class SearchServiceWithRMI {
   // Change from direct access to index files to remote objects that encapsulate searching over the files
   private Map<Character, IndexService> _indexServices;
  …
  public List<String> search(String word) {
    try {
      IndexService indexService = _indexServices.get(word.charAt(0));
      return indexService.search(word);
    } catch (Exception e) {
      return new LinkedList<String>();
    }
  }
  // Constructor connects to all required IndexService implementations through RMI
  public SearchServiceWithRMI(String rmiHost, int rmiPort)
  {
     for (Character c = 'a';  c <= 'z'; c++) {
         IndexService isc = (IndexService)Naming.lookup("rmi://" + rmiHost + ":" + rmiPort + "//IndexServer-" + c);
        _indexServices.set(c, isc);
     }
   }
}
```

To adjust this version, we must update the calls of the SearchService constructor to obtain the required parameters (rmiHost and rmiPort). The way to do this is to add the 2 params to the main of the reactor and pass these parameters through the protocol factory:

```
public static void main(String args[]) {
      if (args.length != 4) {
         System.err.println("Usage: java Reactor <port> <pool_size> <rmiHost> <rmiPort>");
         System.exit(1);
      }
      try {
         ServerProtocolFactory protocolMaker = new ServerProtocolFactory() {
            public AsyncServerProtocol create() {
              return new SearchProtocolWithRMI(args[2], args[3])
            }
         };
```

```
public class SearchProtocolWithRMI implements AsyncServerProtocol {
    private SearchServiceWithRMI _s;
```

```
    public SearchProtocolWithRMI(String rmiPort, String rmiHost) {
        _s = new SearchServiceWithRMI(rmiHost, rmiPort);
    }
    // rest unchanged
}
```

Note: to avoid the cost of constructing the SearchServiceWithRMI object for each protocol instance, it would be in practice beneficial to make this class a singleton (with a static create() method that returns always the same instance).

סעיף ה (5 נקודות)

The following processes must be run to deploy the system:

1. rmiRegistry on a certain host with a certain port:
   [On silver.bgu.ac.il] rmiRegistry 2012
2. One instance of IndexServiceImpl for each letter:
   java IndexServer silver.bgu.ac.il 2012 /usr/index/a.idx a
   java IndexServer silver.bgu.ac.il 2012 /usr/index/b.idx b
   …
   java IndexServer silver.bgu.ac.il 2012 /usr/index/z.idx z
3. One instance of the reactor:
   The parameters are: port of the reactor, poolsize, rmiHost and rmiPort
   [On black.bgu.ac.il] java Reactor 12012 10 silver.bgu.ac.il 2012
4. Then run the client which needs to know the port and host of the reactor only
   java Client black.bgu.ac.il 12012

שאלה 4 (10 נקודות)

סעיף א (5 נקודות)

In the current solution, the primary key of actors_movies is (actorId, movieId).
This means that we cannot have the same actor in the same movie more than once (because a primary key cannot have repetitions – values must be unique).

If we want to allow the required modification, we must add the field role to the primary key:

Primary key (actorId, movieId, role);

סעיף ב (5 נקודות)

```
Select actors_movies.role
```

```
From ((actors inner join actors_movies on actors.id = actors_movies.actorId)
        Inner join movies on movies.id = actors_movies.movieId)
            Inner join directors on movies.directorId = directors.Id
Where directors.name = 'Efraim Kishon' and
        actors.name = 'Haim Topol'
```