גיליון תשובות

מספר נבחן:_____

שאלה 1 (30 נקודות)

סעיף א (5 נקודות)

@INV: getRadius() >= 0 && getX() >= getRadius() && getY() >= getRadius()

סעיף ב (3 נקודות)

בשאלה זו נדרש לפתור בעיית בטיחות בהרצה סדרתית. כלומר, למנוע מצב בו לא מתקיימת התכונה הנשמרת גם אם התהליך מורכב מת'רד אחד בלבד.
במימוש הנוכחי, אין בדיקת תקינות הפרמטרים בבנאי וב setters
[שדות המחלקה, אגב, פרטיים (ברירת המחדל ב Java), כך שלא ניתן לשנותם שלא דרך המתודות]

```
class CircleImpl implements Circle {
  int _x;
  int _y;
  int _radius;

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

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

  public void setCenter(int x, int y)    throws Exception {
    if (!checkInv(x, y, _radius))
      throw new Exception();
```

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

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

סעיף ג (4 נקודות)

המחלקה אינה בטוחה תחת חישוב מקבילי שכן היא אינה שומרת על האינווריאנטה.
לדוגמא:

```
Circle c = new CircleImpl(2,2,1);
...

T1: c.setCenter(1,1);
T2: c.setRadius(2);
```

עבור תזמון ת'רדים בו הן T1 והן T2 בודקים בהצלחה את תנאי ההתחלה של המתודות עבור מצב מעגל (2,2,1), יתכנו לאחר מכן השמת המרכז והרדיוס, כך שיתקבל מעגל (1,1,2) שאינו שומר על התכונה הנשמרת..

השאלה באה לבחון את הבנת ההגדרה הפשוטה של בטיחות. תשובות שהצביעו על 'בעיה'/'בלאגן'/'תוצאה לא רצויה', לא קבלו את כל הנקודות.

הפתרון הפשוט, הטריוויאלי עד כדי בנאליות, הינו סנכרון מלא של המחלקה.
סטודנטים רבים בחרו, משום מה, לסנכרן רק את ה setters ולא את ה getters. באופן כללי, זוהי תבנית עיצוב בעייתית (קריאת מידע שכרגע מתעדכן על ידי ת'רד אחר, אי ודאות שהמידע נקרא מה RAM ולא מה cache), הדורשת הסבר. בהסתכלות הצרה של קיום הבטיחות בלבד, אי סנכרון מתודות ה get שומר על הבטיחות רק כאשר קריאת השדה היא אטומית. תשובות ללא סנכרון ה get, אשר לא התייחסו בהסבר כלשהו לנקודה זו, לא קבלו את מלוא הנקודות. לפנים משורת הדין, נלקחה טעות שכיחה זו בחשבון בחישוב הפקטור.

```
class CircleImpl implements Circle {
  int _x;
  int _y;
  int _radius;

  CircleImpl(int x, int y, int radius) throws Exception {
    if (!checkInv(x,y,radius))
```

```
      throw new Exception();
    _x = x;
    _y = y;
    _radius = radius;
  }

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

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

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

סעיף ד

i. (2 נקודות)

```
@INV: getRadius() >= 0
```

ii. (3 נקודות)

המחלקה בטוחה כי הרדיוס תמיד חיובי: נבדק בבנאי ולאחר מכן בתחילת מתודת ההשמה setRadius
מאחר וקריאה וכתיבה של int הינה אטומית, לא יתכן כי שני ת'רדים יכתבו חלקים שונים של הפרמטר החיובי ויצרו מספר שלילי.
בעיית visibility אינה רלבנטית לבטיחות, התלויה רק בערכו החיובי של הרדיוס.

iii. (5 נקודות)

המימוש ללא הסנכרון (של סעיף ב) אינו נכון, גם עבור התכונה הנשמרת השנייה.

תרחיש בו הרצה מקבילית אינה תואמת הרצה סדרתי בסדר כלשהו של פעולות:

```
c.setCenter(1,1);
c.setCenter(3,3);
```

התוצאות האפשריות עבור מיקום מרכז המעגל, בהרצה סדרתית בסדר כל שהוא:
(1,1) עבור הרצת הפעולה השניה ולאחריה הראשונה
(3,3) עבור הרצת הפעולה הראשונה ולאחריה השניה

לעומת זאת, בהרצה מקבילית של שתי הפעולות ע"י שני ת'רדים ניתן להגיע למצבים
(1,3) – השמת 3 ל x, השמת 1 ל y, השמת 3 ל y
(3,1) – השמת 1 ל x, השמת 2 ל x, השמת 3 ל y, השמת 1 ל y

כלומר, קיימים פלטים מקביליים שאינם אפשריים בהרצה סדרתית, בכל סדר של פעולות ← ההרצה אינה נכונה

השאלה באה לבחון את הבנת ההגדרה של נכונות. תשובות שהצביעו על 'בעיה'/'בלאגן'/'מידע לא מעודכן', מבלי לעמוד על הגדרת הנכונות כהתאמה בין מצב מקבילי למצב סדרתי בסר כל שהוא, לא קבלו נקודות.

iv (8 נקודות)

- יש להשתמש בסנכרון בכל גישה למצב הפנימי כדי לפתור בעיות visibility, וכדי לשמור על נכונות ההרצה.
- יש להימנע מסנכרון מלא, כדי לאפשר הרצה במקביל של פעולות על קורדינטת המרכז והרדיוס
← נשתמש בתבנית של סנכרון חלקי בעזרת שני מוניטורים.

גם כאן בחרו סטודנטים רבים שלא לסנכרן את מתודות ה get. תבנית בעייתית זו אינה פותרת את בעיית הנכונות.

```
class CircleImpl implements Circle {
  int _x;
  int _y;
  int _radius;
  Object _lockRadius, _lockCenter;

  CircleImpl(int x, int y, int radius) throws Exception {
    if (getRadius() < 0)
      throw new Exception();
    _x = x;
    _y = y;
    _radius = radius;
  }

  public int getX() {
    synchronized (_lockCenter) {
      return _x;
```

```
    }
  }
  public int getY() {
    synchronized (_lockCenter) {
      return _y;
    }
  }
  public int getRadius() {
    synchronized (_lockRadius) {
      return _radius;
    }
  }

  public void setCenter(int x, int y)    throws Exception {
    synchronized (_lockCenter) {
      _x = x;
      _y = y;
    }
  }
  public void setRadius(int radius)  throws Exception  {
    synchronized (_lockRadius) {
     if (getRadius() < 0)
       throw new Exception();
     _radius = radius;
    }
  }
}
```

שאלה 2 (30 נקודות)

סעיף א,ב,ג (17 נקודות)

```
include <vector>
using namespace std;

class Circle
{
public:
        Circle(){;}
```

```
        virtual ~Circle(){;}
        virtual int getX() = 0;     // returns the x coordinate of the circle center point
        virtual int getY() = 0;     // returns the y coordinate of the circle center point
        virtual int getRadius() = 0;  // returns the length of the circle radius
        virtual void setCenter(int x, int y) = 0; // set the coordinates for the circle center point
        virtual void setRadius(int radius) = 0;// set the length of the circle radius

        virtual vector<Circle*>& getNeighbors()=0;
        virtual void AddNeighbor(Circle*)=0;
        virtual void RemoveNeighbor(Circle*)=0;

};
```

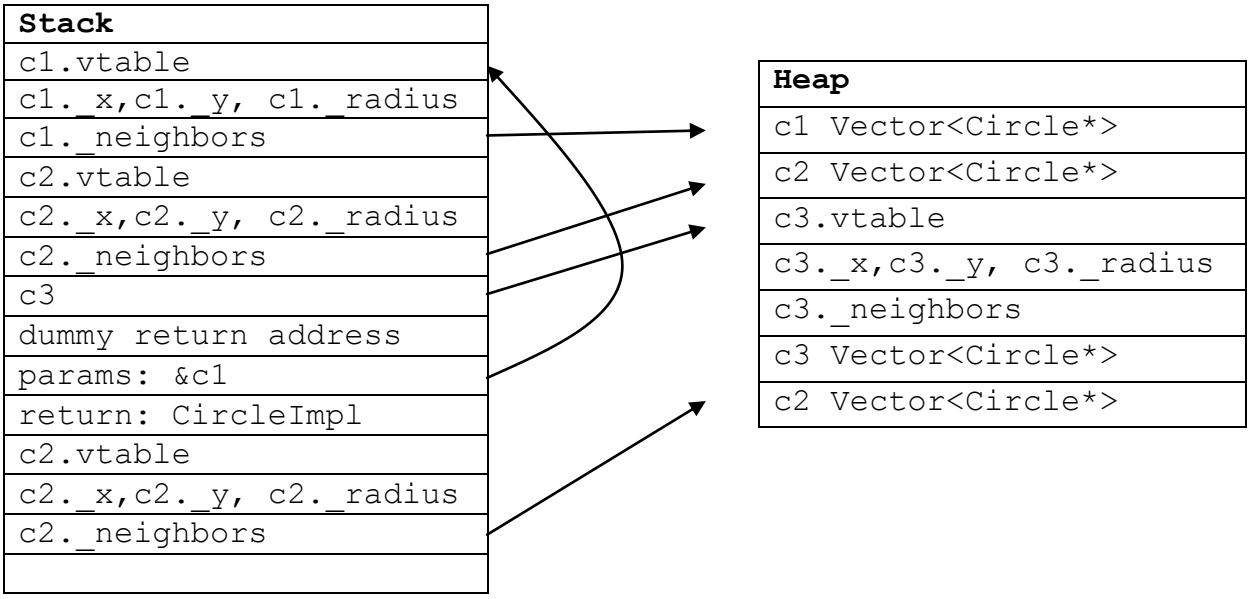
סעיף ג (12 נקודות)

```
class CircleImpl: public Circle
{
public:

        CircleImpl(){;}

        ~CircleImpl();

        CircleImpl(int x, int y, int radius):_x(x),_y(y),_radius(radius){;}

        CircleImpl(const CircleImpl&);

        CircleImpl& operator=(const CircleImpl& rhs);

        int getX(){return _x;}
        int getY(){return _y;}
        int getRadius(){return _radius;}
        void setCenter(int x, int y){_x=x; _y=y;}
        void setRadius(int radius){_radius=radius;}

        vector<Circle*>& getNeighbors();
        void AddNeighbor(Circle*);
        void RemoveNeighbor(Circle*);

private:

        int _x;
        int _y;
        int _radius;
        vector<Circle*> _neighbors;
```

```
};

CircleImpl::~CircleImpl()
{
	_neighbors.clear();
}

CircleImpl::CircleImpl(const CircleImpl& rhs)
{
	_x = rhs._x;
	_y = rhs._y;
	_radius = rhs._radius;
	_neighbors = rhs._neighbors;


}

CircleImpl& CircleImpl::operator=(const CircleImpl& rhs)
{
	_x = rhs._x;
	_y = rhs._y;
	_radius = rhs._radius;
	_neighbors = rhs._neighbors;

	return *this;
}



vector<Circle*>& CircleImpl::getNeighbors()
{
	return _neighbors;
}
void CircleImpl::AddNeighbor(Circle* n)
{
	_neighbors.push_back(n);

}

void CircleImpl::RemoveNeighbor(Circle *n)
{
	for(vector<Circle*>::iterator it = _neighbors.begin(); it != _neighbors.end(); it++){
		if (*it == n){
```

```
                    _neighbors.erase(it);
                    break;
            }
        }
}
```

סעיף ד (13 נקודות)




```
Grading Criteria:

2a+b:     missing virtual, =0 (-2)
2c:  no _neighbors (-6)
          no ctor, copy ctor, operator=, dtor(-2 each). discussion with
no code is ok.
          memory issues - no allocation, deletion, leak (-2)
          incorrect function operation (-4)
2d:  missing entries in table (-1 each)
          wrong values, no dummy (-2)
```

שאלה 3 (30 נקודות)

סעיף א (10 נקודות)

א. עדכנו את הקוד, כך שהמחלקה Reactor תממש את הממשק Counting [10 נקודות]

```
interface Counting {
  int getNumBytesRead();  // returns the total  number of bytes that were read from all clients
  int getNumBytesWrite();  // returns the total  number of bytes that were written to all clients
  int getFailedAccepts ();  // returns the total  number client accepts that were failed
}
```

המתודה_getBytesRead() מחזירה את מספר הבתים אשר נקראו עד כה מכל הלקוחות
המתודה_getBytesWrite() מחזירה את מספר הבתים אשר נכתבו עד כה לכל הלקוחות
המתודה getFailedAccepts() מחזירה את מספר הניסיונות שנכשלו בקבלת לקוח חדש

[Reasoning behind the solution: I do not expect this to be written in your answer]

We must identify:
- Where the requested information is updated (number bytes read, written, accepts fail)
- Where the requested information should be stored and which object should take responsibility to update it
- Who will use the information

The information is updated in:
- ConnectionHandler / read(): `numBytesRead = _sChannel.read(buf);`
- ConnectionHandler / write(): `_sChannel.write(buf);`
- Reactor / run():. `logger.info("problem accepting a new connection: "…)`

The relation Reactor → ConnectionHandler is 1-n (1 connection handler per connection). So we cannot and should not store information in it – the requested information is for the reactor. We must find a clean way to pass information from ConnectionHandler.read and write to the reactor. One way is to go through ReactorData which is already used to pass reference from ReactorData to the ConnectionHandler. Another way is to change the methods read and write to return the requested information. The first method is more general if we want to introduce later additional metrics to the Counting interface.
We will, therefore, store the data in ReactorData, and make this object responsible to update it as well.
The information published by the Counting interface is updated by the Reactor thread (you must convince yourself that this is the case for all 3 cases), but it will be accessed by another thread (some object that monitors the Reactor, gets metric data out of it and stores it in logs or in a monitoring database). So access to the new counting data must be safe.

Solution

1. public class Reactor implements Runnable, Counting {

2. public class CountData {

```
    private int bytesRead;
    private int bytesWritten;
    private int acceptsFailed;
    public CountData() {
        bytesRead = 0; bytesWritten = 0; acceptsFailed = 0;
    }
    public int synchronized getBytesRead() { return bytesRead; }
    public int synchronized getBytesWritten() { return bytesWritten; }
    public int synchronized getAcceptsFailed() { return acceptsFailed; }
    public void synchronized addBytesRead(int more) { bytesRead += more; }
    public int synchronized addBytesWritten(int more) { bytesWritten += more; }
    public int synchronized incAcceptsFailed() { acceptsFailed++; }
}
```

3. public class ReactorData {

```
    private CountData counts;
    public ReactorData(… executor, … selector, … protocolFactory, … tokenizerFactory) {
        counts = new CountData();
        …
    }
    public CountData getCounts() { return counts; }
}
```

4. In class Reactor:

```
        try {
            acceptor.accept();
        } catch (IOException e) {
            _data.getCounts().incAcceptsFailed();
            logger.info("problem accepting a new connection: " + e.getMessage());
        }
```

5. In ConnectionHandler.read():

```
        try {
    numBytesRead = _sChannel.read(buf);
    _data.getCounts().addBytesRead(numBytesRead);
} catch (IOException e) {
    numBytesRead = -1;
}
```

6. In ConnectionHandler.write():

```
        if (buf.remaining() != 0) {
    int numBytesWritten = _sChannel.write(buf);
```

```
    _data.getCounts().addBytesRead(numBytesWritten);
    if (buf.remaining() != 0)
      _outData.add(0, buf);
  }
```

7. In Reactor:

```
int getNumBytesRead()  { return _data.getBytesRead(); }
int getNumBytesWrite() { return _data.getBytesWritten(); }
int getFailedAccepts() { return _data.getFailedAccepts(); }
```

סעיף ב (6 נקודות)

ב. נתונה המחלקה SearchService המכילה את המתודה search(word). מתודה זו מקבלת מילה לחיפוש ומחזירה רשימה של שמות קבצים בהם המילה מופיעה.
המחלקה נעזרת באוסף של קבצי אינדקס המכילים עבור כל מילה את רשימת הקבצים הנדרשת. כל קובץ אינדקס מכיל את רשימות הקבצים עבור מילים המתחילות באות מסוימת. כך שהמתודה search צריכה לבחור את קובץ האינדקס המתאים, על פי האות הראשונה במילה המבוקשת, ואחר כך לסרוק את הקובץ עד המילה המבוקשת, כדי לקבל את רשימת הקבצים בהם היא מופיעה.
להלן הקוד. אופן המימוש של המתודות search1 ו search2 אינו רלבנטי לשאלה זו.

```
class SearchService {
  private Map<Character,File> _indexFiles;
  …
  public List<String> search(String word) {
    try {
      List<Byte> bytes = new LinkedList<Byte>();
      File index = _indexFiles.get(word.charAt(0));
      FileInputStream in = new FileInputStream(index);
     int b = -1;
      while ((b = in.read()) != -1)
        bytes.add((byte)b);
      return search1(word, bytes);
    } catch (Exception e) {
      return new LinkedList<String>();
    }
  }

  protected List<String> search1(String word, List<Byte> bytes) {
    // finds the given word in the given list of bytes, and returns the data (=a list of file names which contains
    // this  word) attached to this word
    …
  }

  protected List<String> search2(String word, List<ByteBuffer> bytes) {
   // finds the given word in the given list of ByteBuffers, and returns the data (=a list of file names which
contains
   //  this  word) attached to this word
    …
  }
}
```

ב. השתמשו במחלקה SearchService ועדכנו את השרת הנוכחי (הניתן בקוד ה Reactor בנספח), כך שיקבל מהלקוח הודעה המכילה מילה לחיפוש ויחזיר ללקוח את רשימת הקבצים בהם היא מופיעה [6 נקודות]

Solution:
In main() change:

```
ServerProtocolFactory protocolMaker = new ServerProtocolFactory() {
   public AsyncServerProtocol create() {
      return new SearchProtocol();
   }
};


public class SearchProtocol implements AsyncServerProtocol {

    SearchService _s = new SearchService();

    public boolean isSearch(String msg) {
      return msg.equals("search");
    }

    // Assume the command comes as "search <queryWord>"
    public String getQuery(String searchCmd) {
       String[] ws = searchCmd.split(" ");
       Return ws[1];
    }

    // ('a', 'b') → 'a#b#'
    public String listToString(List<String> ls) {
       StringBuilder ret = new StringBuilder();
       for (String s : ls) {
          ret.append(s);
          ret.append('#');
       }
       return ret.toString();
    }

    public String processMessage(String msg) {
      if (this._connectionTerminated) {
        return null;
      }
      if (this.isEnd(msg)) {
        this._shouldClose = true;
        return "Ok, bye bye";
      } else if (this.isSearch(msg)) {
          String query = this.getQuery(msg);
          List<String> ls = _s.search(query);
                return this.listToString(ls);
            } else {
                return "Unknown command";
            }
    }
}
```

סעיף ג (4 נקודות)

ג. תארו תרחיש שבו הת'רדים ב Executor נכנסים למצב blocked במהלך ביצוע המתודה processMessage

The code of processMessage eventually calls the method of SearchService.search().
This function includes the call:

```
b = in.read()
```

This is an IO system call which is blocking.
The call is performed by one of the threads of the executor of the reactor.
When this thread performs a blocking system call, the thread is blocked.

Note: Any explanation that involved "locking" or exclusion (synchronize) was wrong. The key point is that a thread can be blocked in different manners – because of locking, or because it waits, or because it performs a system call.

סעיף ד (10 נקודות)

General idea:
The reason for the blocking is a call to blocking IO. When threads in the executor are blocked by blocking IO calls, we waste the limited resources of the reactor in an unbearable manner.

To fix this situation, we must move to non-blocking IO. But reading the index files in non-blocking manner is surely not sufficient. If we read in non-blocking manner, we will not get the full content of the file in one call. And if we call the method search1() or search2() when we haven't read the full content of the file, the call will be an "early call" which is wrong.
We cannot either do a form of "busy wait" where we loop around the non-blocking call on NIO – because this is translating a blocking system call with a blocking busy wait loop.

The solution is to use the reactor to split the work in several stages:
- Start the reading process – read as much as possible.
- Register the file channel to the selector
- React to new events indicating there is new data to read
- When all data is read, continue the search process in a non-blocking manner.

There is no need to create a new reactor, we can use the one that already runs in the server to perform these additional tasks. The flow of events when we get an OP_READ event is this:
- Selector signals OP_READ
- Reactor looks up the selector attachment and invokes handler.read()
- The handler (connection handler) reads from the channel and moves bytes to the Task
- The task is executed in the executor

Look at the code in ProtocolTask.run():

```
public synchronized void run() {
    // Move bytes from socket to tokenizer
    synchronized (_buffers) {
        while(_buffers.size() > 0) {
            ByteBuffer buf = _buffers.remove(0);
            this._tokenizer.addBytes(buf);
        }
    }
    // Move messages from tokenizer to protocol
    while (_tokenizer.hasMessage()) {
        String msg = _tokenizer.nextMessage();
        String response = this._protocol.processMessage(msg);
        if (response != null) {
            try {
                // Move messages from protocol to output socket
                ByteBuffer bytes = _tokenizer.getBytesForMessage(response);
                this._handler.addOutData(bytes);
            } catch (CharacterCodingException e) { e.printStackTrace(); }
        }
    }
}
```

We will add a new way for the Reactor to react to OP_READ events on a file index channel. For this we must get a way to distinguish the attachment as either one of a connected socket or one of a file channel. In parallel to the way we are organized for reading from the socket and performing the work, we will have:
- One object to read from the index file (IndexHandler)
- One Task object to parse the content of the file (tokenizer) and invoke the search continuation and finally pass the result to the client (IndexTask)
- The indexHandler is created and attached to the selector when the "search" command is executed by the Protocol (in the processMessage() method).

These objects must have the following connections:
- Protocol creates IndexHandler and attaches it to the selector.
- IndexHandler fills the IndexTask byte buffers and when all the file is read, passes this task to the executor.
- IndexTask collects all bytes read chunk by chunk from the index file, and when complete, invokes search on the searchService and passes the results to the ConnectionHandler

Solution

In SearchProtocol (the differences are marked in bold):

Add a member to store the ReactorData to access selector and a way to get it in ctor

```
private ReactorData _data;

    // An asynchronous call to the protocol: start working, and when done in the future
```

```
  // pass result back to connectionHandler h.
  // We must change ProtocolTask to invoke this with _handler instead of processMessage
  // and add this as a method in the AsyncProtocol interface.
  public String asyncProcessMessage(String msg, ConnectionHandler h) {
     if (this._connectionTerminated) {
        return null;
     }
     if (this.isEnd(msg)) {
        this._shouldClose = true;
        return "Ok, bye bye";
     } else if (this.isSearch(msg)) {
         String query = this.getQuery(msg);
         File index = _s.getIndex(query); // gets the right file (to add to searchServ.)
         FileChannel in = new FileInputStream(index).getChannel();
         in.configureBlocking(false);
         // Attach the new file channel to the selector
         // React to READ events. Attach a new IndexHandler to this key.
         IndexHandler ih = new IndexHandler(in, _data, query, h);
         SelectionKey key = in.register(_data.getSelector(), OP_READ, ih);

         // Return null to ProtocolTask.run() to indicate computation is not complete
         return null;
          } else {
             return "Unknown command";
          }
  }

public class IndexHandler {
  private final FileChannel _channel;
  private final ReactorData _data;
  private final String _query;
  private IndexTask _task;
  private ConnectionHandler _handler;
  protected final StringMessageTokenizer _tokenizer; // To encode messages

  public IndexHandler(FileChannel in, ReactorData data, String query, ConnectionHandler h) {
     _channel = in;
     _data = data;
     _query = query;
     _handler = h;
     _tokenizer = _data.getTokenizerMaker().create();
     _task = new IndexTask(_query, _tokenizer, _handler);
  }
```

```
    // Invoked by reactor when a read event is triggered by selector
    public void read() {
      ByteBuffer buf = ByteBuffer.allocate(BUFFER_SIZE);
      int numBytesRead = 0;
      try {
        numBytesRead = _channel.read(buf);
      } catch (IOException e) {
        numBytesRead = -1;
      }
      // Did we reach the end of file?
      if (numBytesRead == -1) {
        _data.getExecutor().execute(_task);
      } else {
        buf.flip();
        _task.addIndexBuffer(buf);
      }
    }
}

public class IndexTask implements Runnable {
    // Store the content of the index file to read as a list of byte buffers
    private final List<ByteBuffer> _indexBuffers = new LinkedList<ByteBuffer>();


    public IndexTask(final String query,
                     final StringMessageTokenizer tokenizer,
                     final ConnectionHandler h) {
      _query = query;
      _tokenizer = tokenizer;
      _handler = h;
    }

    public void addIndexBuffer(ByteBuffer b) {
      _indexBuffers.add(b);
    }

    public void run() {
      SearchService s = new SearchService();
      List<String> ls = s.search2(_query, _indexBuffers);
      String response = listToString(ls); // Same as in (λ)
      ByteBuffer bytes = _tokenizer.getBytesForMessage(response);
      _handler.addOutData(bytes);
    }
}
```

Finally, in the reactor, we must distinguish when we get an OP_READ event if it comes from a client socket, to be passed to a ConnectionHandler attachment, or to an IndexHandler. But in fact, the reactor does not really care about this – if instead we just define a common interface ReadHandler, we can just cast the attachment to this common interface and avoid doing any further improper casting:

```
public Interface ReadHandler {
  public void read();
}
```

And mark both ConnectionHandler and IndexHandler as ReadHandlers:

```
public class ConnectionHandler implements ReadHandler { ...
}
public class IndexHandler implements ReadHandler { …
}
```

And accordingly, in Reactor:

```
        if (selKey.isValid() && selKey.isReadable()) {
            ReadHandler handler = (ReadHandler)selKey.attachment();
            logger.info("Channel is ready for reading");
            handler.read();
        }
```

שאלה 4 (10 נקודות)

סעיף א (5 נקודות)

בעקבות התרחבות קטלוג הסרטים של מועדון וְשֶׁרֶט לָנֶפֶש, הוחלט לשמור את המידע עבורו בבסיס נתונים רלציוני.

א. הגדירו מודל נתונים עבור קטלוג הסרטים של המועדון המכיל את המידע הבא:
<u>סרטים</u>: שם הסרט, ארץ ושנת הוצאה, שם הבמאי
<u>במאים</u>: שם הבמאי, רשימת סרטים
<u>שחקנים</u>: שם השחקן, רשימת תפקידים בסרטים

Key point:
The information listed in the question is NOT a relational data model. It is just a verbal description of the required information. It is your job to translate it into a proper relational model, normalized (no repetition of information across tables), and with proper types and keys. In particular, it is not possible to store arrays (or lists) in the relational data model.

To do this, identify the expected cardinality of the relations between the entities:
Movies ←n – 1 → Director (In general a movie has only one director)
Movies ←n – n → Actors (In general, actors play in many movies, and movies have many actors)

1-n relations are encoded by a single foreign key in the table that has the n end.

n-n relations are encoded by a cross-table (2 foreign keys to the related tables) with additional information that characterizes the relation if needed (in our case, the role played by the actor in the movie is data that belongs to the cross table).

Solution:

```
Create table movies(
   Int id primary key;
   Varchar(200) name;
   Varchar(80) country;
   Date publicationDate;
   Int directorId foreign key references directors(id);
 )

Create table directors (
   Int id primary key;
   Varchar(80) name;
)
```

```
Create table actors (
   Int id primary key;
   Varchar(80) name;
)

Create table actors_movies (   -- cross table
   Int actorId foreign key references actors(id);
   Int movieId foreign key references movies(id);
   Varchar(80) role;
   Primary key (actorId, movieId);
)
```

Notes:
- If you assume that a movie can have several directors, then you need a cross table movies_directors as well.
- If you assume that additional information on directors may not be added, or that names of directors are not modified – then you may use the field directorName directly in the movies table.

סעיף ב (5 נקודות)

ב. כתבו שאילתת SQL המחזירה רשימה של: שם-סרט שם-במאי שחקן תפקיד

Note that this is just about joining information from multiple tables without additional criteria, for selection or sort.

```
Select movies.name, directors.name, actors.name, actors_movies.role
From (((movies join directors on movies.directorId = directors.id)
            Join actors_movies on movies.id = actors_movies.movieId)
            Join actors on actors_movies.actorId = actors.id)
```