גיליון תשובות

מספר נבחן:_____

שאלה 1 (30 נקודות)

סעיף א (12 נקודות)

המחלקה BankAccount בטוחה תחת הרצת התהליך.
הבעיה ממועד א', של קנית מניה שלא בערך שנבדק בתנאי ההתחלה תוך חריגה מסכום משיכת היתר, אינה קיימת. הסכום שיגרע מהחשבון הוא אותו סכום שנבדק בתנאי ההתחלה. סכום זה אוחסן במשתנה price במחסנית של ה Dealer הנוכחי, כך שאף Dealer אחר אינו מכיר אותו.

מצד שני הרצת התהליך אינה נכונה. בהרצה סידרתית פעולת הקניה תתבצע באופן רציף, כך שאם התבצעו שתי פקודות קניה לאותה מניה, סכום הקניה של הפעולה השניה גבוהה יותר, עקב עידכון מחיר המניה בגמר הקניה הראשונה. בעוד שבחישוב מקבילי יתכן כי שני Dealer יקנו את המניה באותו מחיר, במקרה בו שניהם קראו את מחיר המניה למשתנה price וביצעו את הקניה כל אחד על בסיס מחיר זה.

סעיף ב (6 נקודות)

האינווריאנטה של BankAccount תלויה בערך של המניות שבבעלות בעל החשבון. על אף שהאינווריאנטה עבור חשבון נתון מתקיימת – עקב בדיקת תנאי ההתחלה לפני כל קניה לשם מניעת חריגה ממשיכת היתר המותרת – מכירות עתידיות של המניה מחשבונות אחרים יפרו את קיום האינווריאנטה. לדוגמא, נניח כי בחשבון מסוים:

maxOverDraft_ = 100

savings_ = -200

stocksValues() = 100 (one Checkpoint stock with price 100)

אם ה Dealer מוכר בהמשך מניה אחת של Checkpoint של לקוח אחר, ערכה יקטן מ 100, והחשבון הנ"ל יהיה במשיכת יתר לא חוקית.

סעיף ג (12 נקודות)

אם אין ללקוח מספיק כסף לקנית מניה, עליו להמתין לאחד מהתרחישים הבאים:
- גידול ב _savings שלו עקב הפקדת כספים או כתוצאה מקנית מניות.
- גידול בערך מניותיו עקב עלית הביקוש להן (= קניות של המניה)
- הפחתת ערך המניה אותה הוא רוצה לקנות עקב ירידת הביקוש לה (= מכירות של המניה)

מעיקר הדין יש להמתין הן על BankAccount והן על Stock – על ידי יצירת אוביקט המתנה כך שמבוצע עליו notify בכל פעם שערך מניה משתנה ובכל פעם שסכום הכסף בחשבון עולה.
כתשובה נכונה, קיבלנו שתי אפשרויות:

1. המתנה על BankAccount עם notify ב incSavings()
2. המתנה על Stock עם notify ב buyStock(stock) וב sellStock(stock).

נדרש לפרט את הקוד הרלבנטי (אני לא עושה זאת כאן – אלוף עצלות ואלוף שום)

חבק עלול להתרחש כאשר שני Dealers נכנסים ל wait עקב אי קיום תנאי ההתחלה.
באפשרות השניה יתכן גם חבק עקב אי שחרור הנעילה של BankAccount שכן ה wait שיחרר רק את הנעילה של Stock.

שאלה 2 (10 נקודות)

```
class Polygon {
public:
  Polygon() : area_(-1) {}
  void addPoint(const Point& pt ) {
    invalidateArea();
    points_.push_back(pt);
  }

  const Point& getPoint(eonst int i)  const {  return points_[i];  }
  int getNumPoints()  const  { return points_.size(); }
  double getArea() const {
    if( area_ < 0 ) // if not yet calculated and cached
       calcArea();     // calculate now
      return area_;
  }
private:
  void invalidateArea()  { area_ = -1; }
  void calcArea()  const  {
    area_ = 0;
    vector<Point>::iterator it;
    for( it = points_.begin(); it != points_.end(); ++it )
      area_ += /* some work */;
    }
    vector<Point> points_;
    double   area_;
};
```

הערה: לגבי getArea() ו calcArea() הנקראת על ידה, אם ניתן הסבר מדוע אין להגדירן כ const התשובה נתקבלה.

שאלה 3 (10 נקודות)

```
// x.h: the most efficient version
```

```
//
#include "a.h" // class A (has virtual functions)
class B;
class C;
class E;
class XImpl;
class X : public A
{
public:
   X( const C& );
   B f( int, char* );
   C f( int, C );
   C& g( B );
   E h( E );
private:
   XImpl* pimpl_; // opaque pointer to forward-declared class
};
```

```
//new file XImpl.h
#include <string>
#include <list>
#include "B.h"
#include "C.h"
#include "D.h"

struct XImpl  {
  std::string  name_;
  std::list<C> clist_;
  B  b_;
  D  d_;
};
```

שאלה 4 (10 נקודות)

```
class Point  {
public:
```

```
  Point(int x,int y) { x_ = new int(x));  y_ = new int(y); }
  Point(const Point& other) { x_ = new int(*other.x_); y_ = new int(*other.y_); {}
  ~Point() { delete x_; delete y_; }
  Point& operator=(const Point& other) {
    if (this != &other) {
      delete x_;
      delete y_;
      x_ = new int(*other.x_);
      y_ = new int(*other.y_);
    }
    return *this;
  }
  Point& operator+=(const Point& other) {
    int tmpx = (*other.x_) + (*x_);
    int tmpy = (*other.y_) + (*y_);
    delete x_;
    delete y_;
    x_ = new int(tmpx);
    y_ = new int(tmpy);
    return *this;
  }
  bool operator==(const Point& other) {
    return ((*x_ == *other.x_) && (*y_ == *other.y_));
  }
  int getX() const { return *x_; }
  int getY() const { return *y_; }
protected:
  const int* x_;
  const int* y_;
};
```

<u>הערה</u>:

משמעות ההגדרות:

```
const int* x_;
const int* y_;
```

היא שלא ניתן לבצע השמות לתוכן המוצבע על ידי x_ ו y_, אך ניתן לבצע השמה של כתובת ל x_ ו y_ עצמם, כפי שבא לידי ביטוי בקוד שניתן לכם, בו מבוצע ב constructors השמת ערך new ל x_ ו y_, ומחיקת הזכרון המוצבע על ידי x_ ו y_ ב destructor.

<u>בכל זאת החלטנו לקבל כתשובה נכונה</u> השמות לתוכן של x_ ו y_:

```
  Point& operator=(const Point& other) {
```

```
    if (this != &other) {
      (*x_) = (*other.x_);
      (*y_) = (*other.y_);
      return *this;
    }
  }

  Point& operator+=(const Point& other) {
    (*x_) += (*other.x_);
    (*y_) += (*other.y_);
    return *this;
}
```

על אף שתשובה שכזו מתאימה להגדרות:

```
int* const x_;
int* const y_;
```

שאלה 5 (10 נקודות)

[A] **True -** For example, when one calls a method on a remote object (o2) received as a parameter to a remote object (o1), then the address of o2 is already known without querying rmiregistry.

[B] **False -** RMI communication is implemented in terms of TCP communication. There is no security advantage.

[C] **True** - A process in which a remote object is defined must know all the interfaces of the parameters that are sent to the remote object, otherwise the code could not compile.

[D] **False** - There is no locking operation ever defined in RMI.

[E] **True** - The client must be written in Java.

שאלה 6 (10 נקודות)

סעיף א (10 נקודות)

The observation is that the initial connection by clients is slow.
The reason for this can be at 2 levels:

- The server process is slow - and it takes time before it performs the call to accept on the server socket. Therefore, incoming connections are delayed

because of the delay in the server side.
- The TCP layer of the Operating System is too slow to perform the accept on the incoming port. The three-way handshake process on each incoming connection takes time, and the server port becomes the bottleneck of the TCP accept layer. This is independant of the time it takes on the process time to perform the accept call. As a result, the queue of incoming calls fills up and the incoming connect calls are rejected or time out.

The 2 levels must be addressed to correct the problem.

Introducing 10 server sockets instead of 1 helps correct the OS overloading. We assume here that clients distribute the connections uniformly among the 10 server ports. The best way to achieve this uniform distribution is by making sure the clients randomly pick one of the 10 ports when they connect.

Using 10 ports reduces the limitation of the incoming port queue of the listen/accept server socket (by default, this queue is configured to hold up to 5 incoming connection requests).

This consideration disqualifies "position 1".

The distinction between position 2 and 3 is the following: in position 2, as soon as the accept request is received by the selector, the reactor performs the accept call immediately. In position 3, the call to accept is delayed by queuing it into the thread pool executor. When a worker thread is available, it performs the accept call.

Note that in both cases, the server socket is configured to be non-blocking. This means that the selector will notify that the channel is acceptable only after the TCP layer of the Operating System is ready so that the call to accept() will be non-blocking -- that is, the connection request is already in the queue and the resulting socketChannel can be built with no delay.

In all cases, Position 3 will result in a connection time that is longer than position 2. The reason is that, between the time a connection request is received and the call to accept on the port is performed, in position 3, one must go through the scheduler of the thread pool and compete with other actions the pool is executing (including read and write actions). In position 2, the accept is called immediately when the reactor is notified.

The danger in position 2 is that the execution time of the "accept()" could be long, and end up slowing down the whole reactor thread. But this is highly unlikely, because the execution of accept() is an easy system call to the TCP layer -- which only allocates the resulting socket.

The conclusion is that, if the objective of the modification is to reduce the delay between a client connect request and the first connection, then Position 2 is the preferred method.

סעיף ב (10 נקודות)

השינויים הנדרשים:

במחלקה Reactor
- ייצור של עשרה ServerSocketChannels והשמתם ב Selector עם ConnectionAcceptor עבור ארוע OP_ACCEPT.
- עבור ארוע OP_ACCEPT העברת ה ConnectionAcceptor ל Executor

במחלקה ConnectionAcceptor
- הגדרת המחלקה כמממשת את הממשק Runnable
- הגדרת המתודה run() על ידי קריאה ל accept()

הקוד הנוסף מסומן באדום

```
protected Vector<Integer> _ports;  // vector of 10 port for the 10 ServerSocketsChannels
   protected ExecutorService _pool;
   protected boolean _shouldRun;
   ...
   public void run() {
      try {
         Selector selector = Selector.open();
         for (int i=0; i<10; i++) {
            ServerSocketChannel ssChannel = ServerSocketChannel.open();
            ssChannel.configureBlocking(false);
            ssChannel.socket().bind(new InetSocketAddress(_ports.get(i));
            ssChannel.register(selector, SelectionKey.OP_ACCEPT,
                                          new ConnectionAcceptor(selector, ssChannel, _pool));
         }
         while (_shouldRun) {
            selector.select();
            Iterator it = selector.selectedKeys().iterator();
            while (it.hasNext()) {
               SelectionKey selKey = (SelectionKey)it.next();
               it.remove();
               if (selKey.isValid() && selKey.isAcceptable()) {
                  ConnectionAcceptor connectionAcceptor = (ConnectionAcceptor)selKey.attachment();
                  connectionAcceptor.accept();
                  _pool.execute(connectionAcceptor);
               }
               if (selKey.isValid() && selKey.isReadable()) {
                  ConnectionReader connectionReader = (ConnectionReader)selKey.attachment();
                  connectionReader.read();
```

```
                }
            }
        }
    } catch (IOException e) {
        e.printStackTrace(System.err);
        stopReactor();
    }
    stopReactor();
  }
  ...
}
```

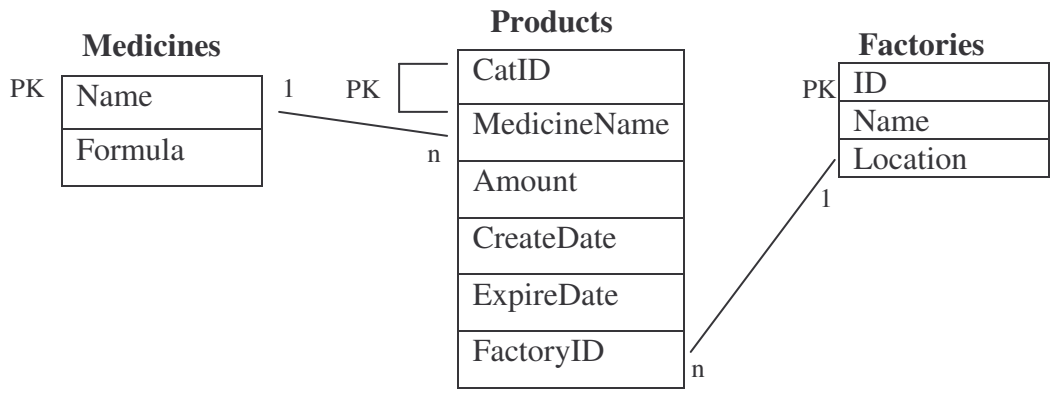
```
public class ConnectionAcceptor implements Runnable {
  protected Selector _selector;
  protected ServerSocketChannel _ssChannel;
  protected ExecutorService _pool;
  ...

  public void run() {
    try {
      accept();
    } catch (IOException  e) {
      e.printStackTrace();
    }
  }
  public void accept() throws IOException {
    // Get a new channel for the connection request
    SocketChannel sChannel = _ssChannel.accept();

    // If serverSocketChannel is non-blocking, sChannel may be null
    if (sChannel != null) {
        SocketAddress address = sChannel.socket().getRemoteSocketAddress();
        System.out.println("Accepting connection from " + address);
        sChannel.configureBlocking(false);
        sChannel.register(_selector, SelectionKey.OP_READ,
                                    new ConnectionReader(sChannel, _pool));
    }
  }
}
```

שאלה 7 (10 נקודות)

סעיף א (5 נקודות)



סעיף ב (5 נקודות)

```
SELECT DISTINCT Medicines.Name, Medicines.Formula
FROM   Medicines,Products, Factories
WHERE  Medicines.Name = Products. MedicineName AND
       Products.FactoryID = Factories.ID AND
       Factories.Location = 'Yerocham'
```