אוניברסיטת בן-גוריון

מדור בחינות

מספר נבחן:_____

רשמו תשובותיכם בגיליון התשובות בלבד
תשובות מחוץ לגיליון לא יבדקו.

בהצלחה!

תאריך הבחינה: 22.2.2010
שם המורה: ד"ר מיכאל אלחדד
ד"ר מני אדלר
מר אוריאל ברגיג
שם הקורס: תכנות מערכות
מספר הקורס: 202-1-2031
מיועד לתלמידי: מדעי המחשב, הנדסת תוכנה
שנה: תש"ע
סמסטר: א'
מועד: ב'
משך הבחינה: שלש שעות
חומר עזר: אסור

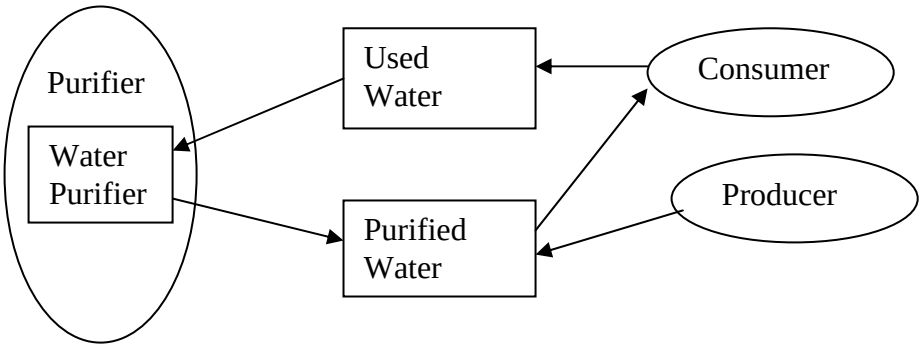
שאלה 1 (30 נקודות)

בשאלה זו נמשיך לעסוק במערכת לטיהור המים בה עסקנו במועד א'.
המערכת מורכבת, כזכור, מהאובייקטים הפסיביים הבאים:

- שני מיכלים (**WaterPool**) - עבור מים מטוהרים ועבור מים משומשים - המממשים את הממשק **Pool**.
- מטהר מים (**WaterPurifier**) המממש את הממשק **Purifier**. בממשק מוגדרת המתודה **getQuality()** המחזירה את כמות המים באחוזים שניתן לטהר באיכות טובה והמתודה **purify** המקבלת כמות של מים משומשים ומחזירה כמות של מים מטוהרים.

בסימולציה רצים 3 אובייקטים אקטיביים:

- **Producer**: ת'רד הרץ מעל המשימה **ProducingTask**: הוספת מים למיכל המים המטוהרים (למשל זרימת המים מהקו העירוני לצנרת הבית).
- **Consumer**: ת'רד הרץ מעל המשימה **ConsumingTask**: צריכת מים ממיכל המטהורים והעברתם בתום השימוש למיכל המים המשומשים (למשל צרכן הפותח ברז, צורך מים, והמים מתנקזים למאגר המים המיועדים לטיהור).
- **Purifier**: ת'רד הרץ מעל המשימה **PurifyingTask**: לקיחת מים ממיכל המים המשומשים, טיהורם בעזרת מטהר המים, והוספתם למיכל המים המטוהרים (כמו פעולת המערכת לטיהור המים האפורים).



הקוד המקורי של המערכת, בדיוק כפי שהופיע במועד א', ניתן בנספח א'.

א. במועד א' נוכחנו לדעת כי המערכת בטוחה. האם ההרצה של המערכת מקיימת נכונות? (כזכור, הרצת מערכת תוגדר כנכונה, אם לכל תוצאה של ביצוע מקבילי קיימת תוצאה סדרתית מתאימה, בסדר כל שהוא של פעולות). נמקו. [5 נקודות]

ב. הסבירו מדוע ביצוע interrupt() על כל אחד משלושת הת'רדים במערכת, לא יגרום בהכרח להפסקת ריצתו. ועדכנו את הקוד כך שעבודתו תופסק בכל מקרה. אין להשתמש ב **System.exit** וכן אין לשנות את חתימות המתודות ב **WaterPool**. [10 נקודות]

ג. הסבירו מדוע ה Producer, הממתין להתרוקנות מיכל המים המטוהרים, יתעורר גם אם נוספו מים למיכל על ידי ה Purifier? [3 נקודות]

ד. עדכנו את המחלקה **WaterPool** כך שת'רד הממתין לירידה בכמות המים לא יתעורר בעקבות הוספת מים על ידי ת'רד אחר. וכן להפך, ת'רד הממתין לתוספת מים לא יתעורר בעקבות הורדת מים על ידי ת'רד אחר. רמז: השתמשו במוניטורים שונים עבור ההמתנות. [12 נקודות]

שאלה 2 (30 נקודות)

נתונות המחלקות הבאות המייצגות מחסנית של אירועים. הקוד תקין:

```
class Event {
public:
  Event(int id) : _id(id) {};
  virtual ~Event() {}
private:
  const int _id;
};

class SellingEvent : public Event {
public:
  SellingEvent(int id, int price) : Event(id), _price(price) {};
  virtual int price() {return _price;}
private:
  int _price;
};

// NOTE: nodes are double linked with next and prev
// all field are public
class EventNode {
public:
  EventNode(){};
  Event *data;
  EventNode *next;
  EventNode *prev;
};

// EventsStack manages a double linked stack based on EvenNode
class EventsStack {
private:
  EventNode *_top;
```

```
  EventNode *_last;
public:
  EventsStack();
  void push(Event *data);
  void pop();
};
```

```
EventsStack::EventsStack():_top(0),_last(0){};

// Note: take your time to understand how push works
void EventsStack::push(Event *e) {
  EventNode *q = new EventNode();
  q->data = e;
  q->next = _top;
  q->prev = 0;
  if (_top != 0)
    _top->prev = q;
  _top = q;
  if (_last == 0)
    _last = q;
}
```

בסעיפים א-ב יש לצייר את תמונת הזיכרון של התהליך הכוללת ערכים הנמצאים במחסנית, ערכים הנמצאים בערימה ומצביעי **vtable** אם יש. אין לאייר את ה **vtables**. אין לאייר ערכים שמתחת ל **stack pointer** ("נמחקו מהמחסנית"). כל סעיף יש לאייר מחדש. לכל תא זיכרון באיור יש לכלול מספר המייצג את כתובתו ומספר המייצג את ערכו.

גודלם של מצביעים ו **int** הוא 4. נקבע כי כתובות בערימה הם בתחום 1000 עד 3000 וכי כתובת המחסנית מתחילה ב 5000 ועולה כשהמחסנית גדלה.

א. ציירו את תמונת הזיכרון לאחר ביצוע השורות הבאות [7 נקודות].

```
void main() {
  EventsStack eventStack;
  Event e1(1);
  eventStack.push(&e1);
```

ב. ציירו את תמונת הזיכרון לאחר ביצוע השורות הבאות [8 נקודות]

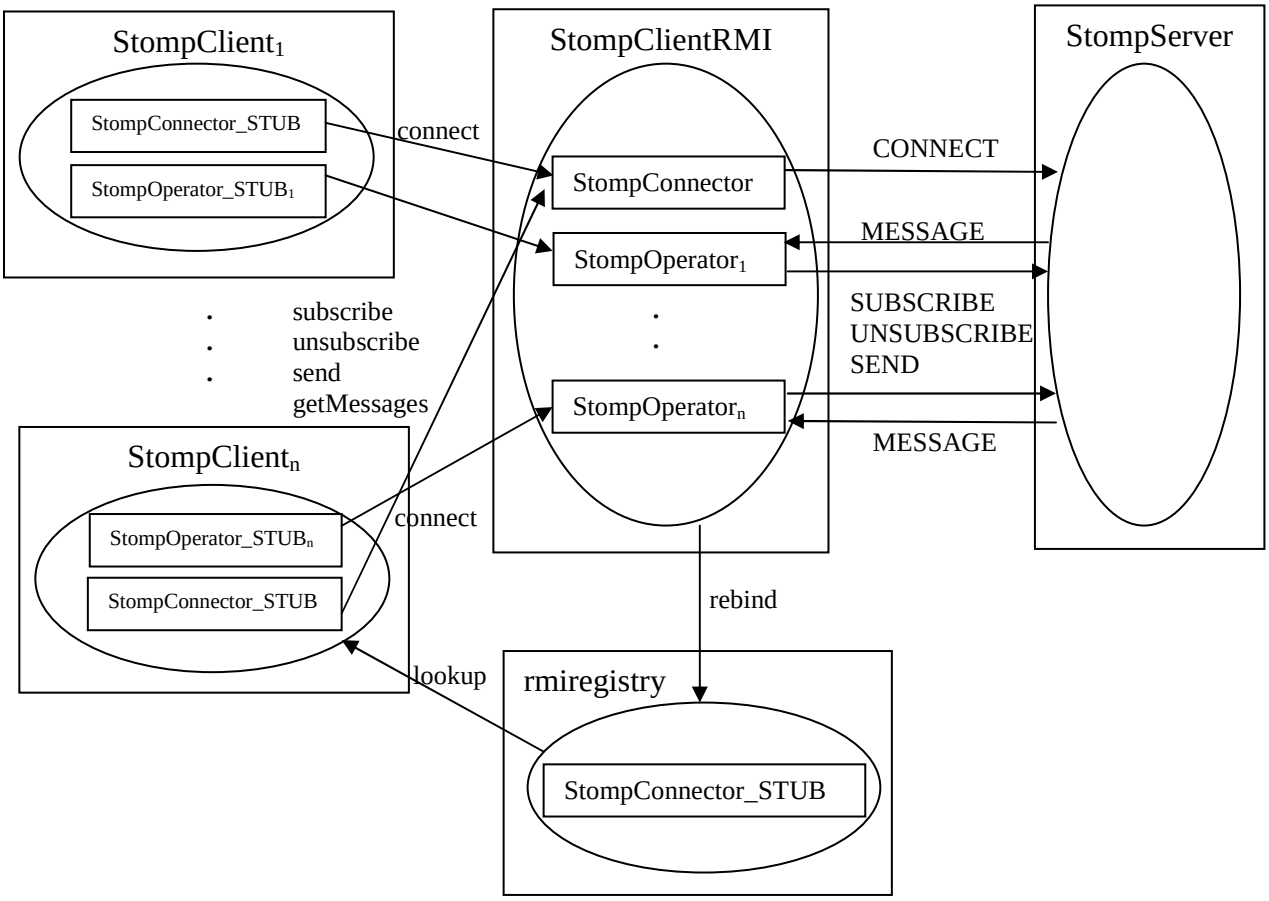
```
void main() {
  EventsStack eventStack;
  Event *e1 = new Event(1);
  eventStack.push(e1);
  SellingEvent e2(2,20);
  eventStack.push(&e2);
```

ג. הוסיפו מימוש לבנאי מעתיק עבור **EventsStack**. פעולת ההעתקה צריכה להיות עמוקה. מחסנית האירועים צריכה להיות מועתקת כולל כל ה- **EventNodes** כך שיווצרו הצבעות כפולות ל Event-ים. ניקוד מלא יינתן לפתרון אלגנטי (כ - 4 שורות קוד).
[7 נקודות]

ד. הוסיפו את המימוש לשיטה **void EventsStack::pop()** המוציאה את האיבר הראשון מהמחסנית. יש לשים לב לעדכון ערכי **prev,next,_top,_last** במידת הצורך במקרים בהם יש איבר יחיד או במקרים בהם יש מספר אברים.
[8 נקודות]

שאלה 3 (30 נקודות)

בשאלה זו נמשיך לעסוק במערכת ממועד א', המפשטת את העבודה של הלקוח מול לשרת ה Stomp, בעזרת תהליך ביניים המספק ממשקי RMI.



RMI Connection **TCP Connection**

<u>הקוד של המערכת מהפיתרון של מועד א' ניתן בנספח ב'.</u>

במימוש הנוכחי משיכה של ההודעות שהגיעו עד כה נעשית באופן יזום, על ידי הפעלת המתודה **getMessages()** של **StompOperator**.

כעת, אנו מעוניינים לקבל הודעות מהשרת באופן שוטף, מבלי לקרוא 'מידי פעם' למתודה **getMessages()**.

כדי לממש זאת, הציע אחד המתרגלים בקורס, להגדיר אצל הלקוח (**StompClient**) **RemoteObject** הממשמש את הממשק **MessageReceiver**. ממשק זה מאפשר להעביר הודעה על ידי הפעלת המתודה **message**.

```
public interface MessageReceiver extends java.rmi.Remote {
  void message(String str) throws java.rmi.RemoteException;
}
```

כאשר לקוח (**StompClient**) רוצה להתחבר לשרת, הוא מייצר מראש אובייקט הממשמש **MessageReceiver**, ומעבירו כפרמטר למתודה **connect**.

הודעות מהשרת יועברו ללקוח ב **StompClientRMI**, על ידי הפעלת המתודה **message** של אובייקט זה (מימוש המתודה הינו למעשה תגובת הלקוח להודעה שנקבלה).

חתימת המתודה **connect** בממשק **StompConnector** שונתה בהתאם - פרמטר נוסף מסוג **MessageReceiver**:

```
public interface StompConnector extends java.rmi.Remote {
  StompOperator connect(String login, String passcode, MessageReceiver messageReceiver)
    throws java.rmi.RemoteException,IOException;
}
```

כעת לא נדרשת יותר המתודה **getMessages()** בממשק **StompOperator**:

```
public interface StompOperator extends java.rmi.Remote {
  void subscribe(String groupName) throws java.rmi.RemoteException,IOException;
  void unsubscribe(String groupName) throws java.rmi.RemoteException,IOException;
  void send(String groupName, String str) throws java.rmi.RemoteException,IOException;
  List<String> getMessages() throws java.rmi.RemoteException,IOException;
}
```

להלן מימוש של ממשקים אלו:

```
public class MessageReceiverImp extends java.rmi.server.UnicastRemoteObject implements MessageReceiver {
  MessageReceiverImpl() throws java.rmi.RemoteException { }
  public void message(String str) throws java.rmi.RemoteException {
    System.out.println(str);
  }
}
```

```
public class StompConnectorImpl extends java.rmi.server.UnicastRemoteObject implements StompConnector {
  protected String _stompServerHost;
  protected int _stompServerPort;

  public StompConnectorImpl(String stompServerHost, int stompServerPort)  throws java.rmi.RemoteException {
    _stompServerHost = stompServerHost;
```

```
    _stompServerPort = stompServerPort;
  }

  public StompOperator connect (String login, String passcode, MessageReceiver messageReceiver)
      throws java.rmi.RemoteException,IOException {
    Socket socket = new Socket(_stompServerHost, _stompServerPort);
    PrintWriter out = new PrintWriter(new OutputStreamWriter(socket.getOutputStream(),"UTF-8"));
    out.print("CONNECT\nlogin: " + login + "\npasscode: " + passcode + "\n\n" + '\0');
    return new StompOperatorImpl(socket.getInputStream(),socket.getOutputStream(),messageReceiver);
  }
}
```

א. עדכנו את הקוד המקורי של המחלקה **StompOperatorImpl** כך שיממש את האופן החדש של קבלת ההודעות מהשרת. [6 נקודות]

ב. עדכנו את הקוד ב StompClient כך שיתאים למערכת החדשה.
יש לדאוג לכך שההודעות המתקבלות מהשרת יודפסו על המסך של התהליך StompClient מיד עם קבלתן. [6 נקודות]

ג. עדכנו את הקוד מסעיף ב' (אם לא עניתם על סעיפים א-ב, התייחסו לקוד המקורי), כך שבתגובה לקבלת הודעה, יִשָלַח מהזיכרון של תהליך הלקוח **(StompClient)**, הודעת תגובה, המאשרת את קבלת ההודעה: **"Got your message: <received message>"**. ניתן לשנות את חתימת המתודות בממשקים. [10 נקודות]

ד. ציינו נכון/ לא נכון עבור הקביעות הבאות:

I הוספת תהליכי הביניים* יקרה יותר מבחינת מספר פעולות תקשורת
II הוספת תהליכי הביניים* מורידה את רמת האמינות של שליחת ההודעות
III מימוש ה StompServer עם reactor חוסך משאבי זיכרון עבור מספר רב של לקוחות
IV מימוש ה StompServer עם reactor משרת באופן <u>הוגן</u> יותר מספר רב של לקוחות

* תהליכי הביניים הם ClientRMI, rmiregistry שנוספו למערכת המקורית של תרגיל 3, שהתבססה רק על StompClient ו StompServe .

[8 נקודות]

שאלה 4 (10 נקודות)

במועד א' בנינו מודל נתונים עבור לוח זמני הטיסות של נמל התעופה בן גוריון.
כזכור, לכל טיסה יש מספר זיהוי יחודי. בנוסף, מתאפיינת הטיסה על פי היעד שלה, מספר הטרמינל, שער היציאה, תאריך ושעת הטיסה, והמטוס המבצע את הטיסה. מטוס מתאפיין על פי מספר יחודי, וציון הסוג שלו.
לכל סוג מטוס יש מחרוזת המתארת את המפרט הטכני שלו, וכן שדה המציין את קיבלת הנוסעים המקסימאלית.

להלן תאור של המודל כטבלאות ומפתחות ב SQL:

```
CREATE TABLE PlaneModels (
   Model varchar(20) PRIMARY KEY,
   TechnicalSpec varchar(10000)
   Capacity integer)

CREATE TABLE Planes (
    ID integer PRIMARY KEY,
    Model varchar(20) FOREIGN KEY REFERENCES PlaneModels(Model))

CREATE TABLE Flights (
   ID integer PRIMARY KEY,
   Destination varchar(100),
   Terminal varchar(20),
   ExitGate varchar(20),
   ExitDate date,
   ExitTime time,
   PlaneId integer FOREIGN KEY REFERENCES Planes(ID))
```

א. הרחיבו את מודל הנתונים כך שיכלול את הנתונים הבאים:
- שם הטייס המטיס את המטוס
- תאריך הטיפול הבא של המטוס
- קיבולת מחלקת העסקים

[6 נקודות]

ב. כתבו שאילתת SQL המחזירה את מספרי הטיסות ואת קיבולת מחלקת העסקים שלהם, ממוינים על פי היעדים בסדר עולה [4 נקודות]

נספח א': הקוד המקורי של המערכת של השאלה הראשונה, כפי שהופיע במועד א'.

```
interface Pool {
  long getCapacity();
  float getContent();
  float remove();
  void add(float addition);
}

class WaterPool implements Pool {
  private final long _capacity;
  private float _content;

  WaterPool(float content, long capacity)  { _content = content; _capacity = capacity; }

  public long getCapacity() { return _capacity; }
  public synchronized float getContent () { return _content; }
  public synchronized float remove() {
    while (_content == 0)
      try { wait(); } catch (InterruptedException e) { return 0; }
    float ret = _content;
    _content = 0;
    notifyAll();
    return ret;
  }

  public synchronized void add(float addition) {
    while (_content + addition  > _capacity)
      try { wait(); } catch (InterruptedException e) { return; }
     _content += addition;
    notifyAll();
  }
}
```

```
interface Purifier {
  float getQuality();
  float purify(float capacity);
}

class WaterPurifier implements Purifier {
  private final float _quality;
  WaterPurifier(float quality) { _quality = quality; }
  public float getQuality() { return _quality; }
  public float purify (float content) {
```

```
    try {
        Thread.sleep((long)content);        // sleep to simulate using the water
        } catch (InterruptedException e) { return 0; }
    return _quality * content;
  }
}
```

```
class ProducingTask implements Runnable {
  private Pool _purifiedPool;
  ProducingTask (Pool purifiedPool) {
    _purifiedPool = purifiedPool;
  }
  public void run() {
    while (true)
      _purifiedPool.add(_purifiedPool.getCapacity());
  }
}
```

```
class PurifyingTask implements Runnable {
  private Pool _usedPool, _purifiedPool;
  private Purifier _purifier;
  PurifyingTask (Pool usedPool, Pool purifiedPool, Purifier purifier) {
    _usedPool = usedPool; _purifiedPool = purifiedPool; _purifier = purifier;
  }
  public void run() {
    while (true)
      _purifiedPool.add(_purifier.purify(_usedPool.remove()));
  }
}
```

```
class ConsumingTask implements Runnable {
  private Pool _purifiedPool, _usedPool;
  ConsumingTask (Pool purifiedPool, Pool usedPool) {  _purifiedPool = purifiedPool; _usedPool = usedPool; }
  public void run() {
    while (true) {
      float w = _purifiedPool.remove();
      try { Thread.sleep((long)w); } catch (InterruptedException e) { return; }
      _usedPool.add(w);
    }
  }
}
```

```
class Test {
  public static void main(String[] args) {
```

```
    Pool purifiedWaterPool = new WaterPool(0, 1000);
    Pool usedWaterPool = new WaterPool(0, 1000);
    Purifier waterPurifier = new WaterPurifier(0.8);
    new Thread(new ProducingTask(purifiedWaterPool)).start();
    new Thread(new PurifyingTask(usedWaterPool, purifiedWaterPool, waterPurifier)).start();
    new Thread(new ConsumingTask(purifiedWaterPool, usedWaterPool)).start();
  }
}
```

נספח ב': הקוד המקורי של המערכת של השאלה השלישית, כפי שהופיע בפיתרון מועד א'.

```
public interface StompConnector extends java.rmi.Remote {
  StompOperator connect(String login, String passcode)  throws java.rmi.RemoteException,IOException;
}
```

```
public interface StompOperator extends java.rmi.Remote {
  void subscribe(String groupName) throws java.rmi.RemoteException,IOException;
  void unsubscribe(String groupName) throws java.rmi.RemoteException,IOException;
  void send(String groupName, String str) throws java.rmi.RemoteException,IOException;
  List<String> getMessages() throws java.rmi.RemoteException,IOException;
}
```

```
public class StompConnectorImpl extends java.rmi.server.UnicastRemoteObject implements StompConnector {
  protected String _stompServerHost;
  protected int _stompServerPort;

  public StompConnectorImpl(String stompServerHost, int stompServerPort)  throws java.rmi.RemoteException {
    _stompServerHost = stompServerHost;
    _stompServerPort = stompServerPort;
  }

  public StompOperator connect (String login, String passcode) throws java.rmi.RemoteException,IOException {
    Socket socket = new Socket(_stompServerHost, _stompServerPort);
    PrintWriter out = new PrintWriter(new OutputStreamWriter(socket.getOutputStream(),"UTF-8"));
    out.print("CONNECT\nlogin: " + login + "\npasscode: " + passcode + "\n\n" + '\0');
    return new StompOperatorImpl(socket.getInputStream(),socket.getOutputStream());
  }
}
```

```
public class StompOperatorImpl extends java.rmi.server.UnicastRemoteObject , implements StompOperator {
```

```
  protected PrintWriter _writer;
  protected BufferedReader _reader;
  private List<String> _msgs;

  // Define a listener task to receive the messages from the Stomp server.
  private class Listener implements Runnable {
    private MessageTokenizer _tok;
    public Listener() {  _tok = new MessageTokenizer(_reader, '\0'); }
    public void run() throws IOException {
      while (_tok.isAlive())
        synchronize (_msgs) { _msgs.add(_tok.nextToken()); }
    }
   }

  public StompOperatorImpl(InputStream in, OutputStream out) throws java.rmi.RemoteException, IOException {
    _reader = new BufferedReader(new InputStreamReader(in,"UTF-8"));
    _writer = new PrintWriter(new OutputStreamWriter(out,"UTF-8"));
    _msgs = new ArrayList<String>();
    // Construct listener
    new Thread(new Listener(_reader)).start();
  }

  public void subscribe (String group)   throws java.rmi.RemoteException, IOException {
      synchronized (_writer) { _writer.print("SUBSCRIBE\ndestination: " + group + "\n\n" + '\0'); }
  }

  public void unsubscribe (String group)  throws java.rmi.RemoteException, IOException {
      synchronized (_writer) { _writer.print("UNSUBSCRIBE\ndestination: " + group+ "\n\n" + '\0'); }
  }

  public void send (String group, String str)   throws java.rmi.RemoteException, IOException {
      synchronized (_writer) { _writer.print("SEND\ndestination: " + group+ "\n\n" + str + "\n" + '\0'); }
  }

  public List<String> getMessages() throws java.rmi.RemoteException {
    // Snapshot copy of the messages and reset it.
    List<String> res;
    synchronized (_msgs) {
      res = new ArrayList<String>(_msgs);
      _msgs.clear();  // reset the accumulator to receive new messages.
    }
    return res;
  }
}
```

```
public class StompClientRMI {
   public static void main(String[] args) {
      try {
         // Get the address IP/port of the Stomp server from args.
         String stompHost = args[0];
         int port = Integer.parseInt(args[1]);
         StompConnector c = new StompConnectorImpl (stompHost, port);
         // Publish the connector to the rmiRegistry at the given address
         Naming.rebind("rmi://132.23.5.8:2010/StompConnector", c);
      } catch (Exception e) { e.printStackTrace(); }
   }
}
```

```
public class StompClient {
  public static void main(String[] args) {
     try {
       StompConnector c = (StompConnector)Naming.lookup("rmi://132.23.5.8:2010/StompConnector");
       StompOperator s = c.connect("user", "password");
       s.subscribe("q1");  s.subscribe("q2"); s.send("q3", "Suzy Surprise");
       Thread.sleep(60000);
       for (String s : c.getMessages())
         System.out.println(s);
     } catch (Exception e) { e.printStackTrace(); }
  }
}
```