

פיתוח מערכות תוכנה מבוססות Java  
שיעור מספר 5:  
"תקשורת עוינת"

**אוהד ברזילי**  
**מאי 2007**

בית הספר למדעי המחשב  
אוניברסיטת תל אביב

# על סדר היום

- תקשורת בין מחשבים
- תקשורת וחוטים
- מיקומה של התקשורת ביישום
- RMI - תקשורת ללא תקשורת

# פרוטוקול תקשורת

■ פרוטוקול תקשורת הוא מסמך המתאר תחלופת הודעות בין מחשבים

■ המסמך מתאר את:

■ מבנה ההודעות השונות

■ סדר הודעות חוקי

■ הודעות שגיאה אפשריות

■ המסמך אינו מתייחס לשפת התכנות ששימשה לכתובת תוכניות המדברות בעזרת הפרוטוקול

# פרוטוקול תקשורת

- ניתן לשלב בין פרוטוקולים ע"י שליחת הודעות בתוך הודעות (protocol stack) כל אחת מההודעות מטפלת בהיבט אחר של המשלוח
- לדוגמא: במשלוח שורה ביישום chat ההודעה שנשלחת בפועל מכילה מידע גם על: איתור הכתובת, בקרת השגיאות, כרטיס הרשת ועוד....
- קיימים מספר ארגונים בעולם (ארגוני תקינה) אשר מרכזים את תהליך כתיבת המסמכים האלה. לדוגמא:  
<http://www.ietf.org/rfc.html>
- קיימים ארגונים מסחריים אשר אינם מפרסמים את הפרוטוקול שבו הם משתמשים. לדוגמא: <http://www.skype.com>

# Internet for dummies

- האינטרנט היא רשת מחשבים (= מחשבים + חוטים)
- לכל מחשב ניתנת כתובת (IP Address) המורכבת מ-4 מספרים בין 0 ל-255
- כל תוכנית הרצה במחשב מקבלת מספר שלוחה (port) בין 0 ל-65,535
- ניתן ליצור קשר עם מחשב המחובר לרשת ע"י ציון כתובתו ופניה אליו בהודעה המתאימה לפי הפרוטוקול
- בדרך כלל ע"י שימוש בתוכנית המממשת את הפרוטוקול

# תקשורת ב Java

■ תוכנית Java מתקשרת עם העולם החיצוני ע"י זרמים (streams)

■ עם המקלדת:

```
InputStreamReader in = new InputStreamReader(System.in);  
BufferedReader bin = new BufferedReader(in);  
String text = bin.readLine();
```

■ עם קובץ:

```
InputStreamReader in = new InputStreamReader(  
    new FileInputStream("foo.txt"));  
BufferedReader bin = new BufferedReader(in);  
String text = bin.readLine();
```

■ ננסה להשתמש באותה גישה בדיוק כדי לתקשר עם **תוכנית מחשב אחרת** (אולי במחשב אחר). לשם כך משתמש במחלקה **Socket** (שקע)

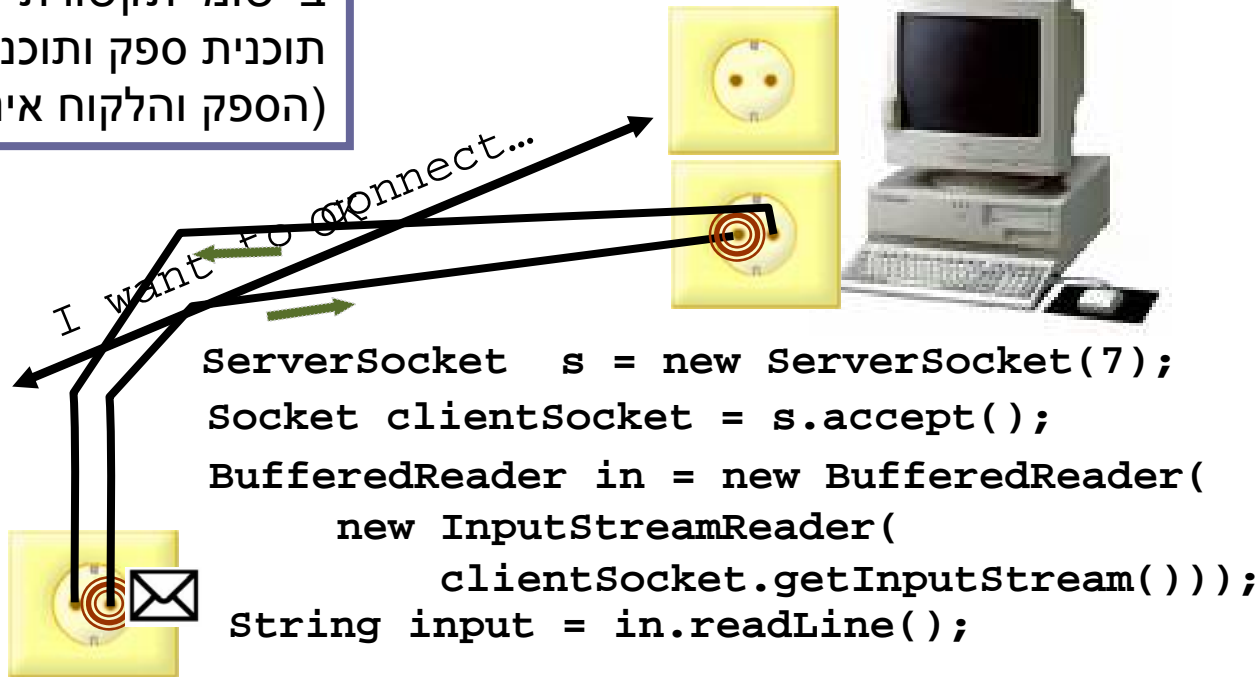
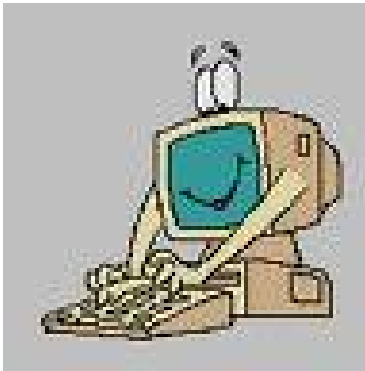
■ ב Java קיים חוסר סימטריה בין **יוזם ההודעה הראשונה** (לקוח - Client) ובין הצד השני (שרת - Server)



# דוגמא: איך נשלח מחרוזת בין שני מחשבים?

ביישומי תקשורת יש לכתוב שתי תוכניות:  
תוכנית ספק ותוכנית לקוח.  
(הספק והלקוח אינם מריצים את אותה התוכנית)

[www.soft1.co.il](http://www.soft1.co.il) (132.67.250.239)



```
Socket s = new Socket("www.soft1.co.il", 7);
PrintWriter out = new PrintWriter(s.getOutputStream(), true);
out.println("hello there!");
```

פיתוח מערכות תוכנה מבוססות Java  
אוניברסיטת תל אביב

```
package examples.sockets;
```

```
import java.io.*;  
import java.net.*;
```

# EchoClient

```
public class EchoClient {  
    public static void main(String[] args) throws IOException {  
  
        Socket echoSocket = null;  
        PrintWriter out = null;  
        BufferedReader in = null;  
  
        try {  
            String hostName = args[0];  
            echoSocket = new Socket(hostName, 7);  
            out = new PrintWriter(echoSocket.getOutputStream(), true);  
            in = new BufferedReader(  
                new InputStreamReader(echoSocket.getInputStream()));  
        } catch (UnknownHostException e) {  
            System.err.println("unkown host");  
            System.exit(1);  
        } catch (IOException e) {  
            System.err.println("Couldn't get I/O for "  
                + "the connection to host");  
  
            System.exit(1);  
        } catch (ArrayIndexOutOfBoundsException aiobe) {  
            System.err.println("wrong usage: enter hostname");  
            System.exit(1);  
        }  
    }  
}
```



```
// establish an input stream to read from the standard input
BufferedReader input =
    new BufferedReader(new InputStreamReader(System.in));
```

```
String userInput;
```

```
while ((userInput = input.readLine()) != null) {
    // writer line to output stream
    out.println(userInput);
    out.flush();

    // print received echo result
    System.out.println("echo: " + in.readLine());
}
```

**Buisness Logic**

```
// close streams and socket
out.close();
in.close();
input.close();
echoSocket.close();
```

```
}
```

```
class EchoServer {
```

EchoServer

```
    public static void main(String[] args) {
        ServerSocket serverSocket = null;
        try {
            serverSocket = new ServerSocket(7);
        } catch (IOException ioe) {
            System.err.println("Couldn't listen on port 7");
            System.exit(1);
        }

        Socket clientSocket = null;
        try {
            clientSocket = serverSocket.accept();
        } catch (IOException ioe) {
            System.out.println("Accept failed: 7");
            System.exit(-1);
        }

        try {
            PrintWriter out =
                new PrintWriter(clientSocket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));

            String input = in.readLine();
            out.println(input);

            out.close();
            in.close();
            clientSocket.close();
            serverSocket.close();
        } catch (IOException ioe) {
            System.err.println("Couldn't communicate with client");
        }
    }
}
```

Buisness Logic

פיתוח מערכות תוכנה מבוססות Java  
אוניברסיטת תל אביב

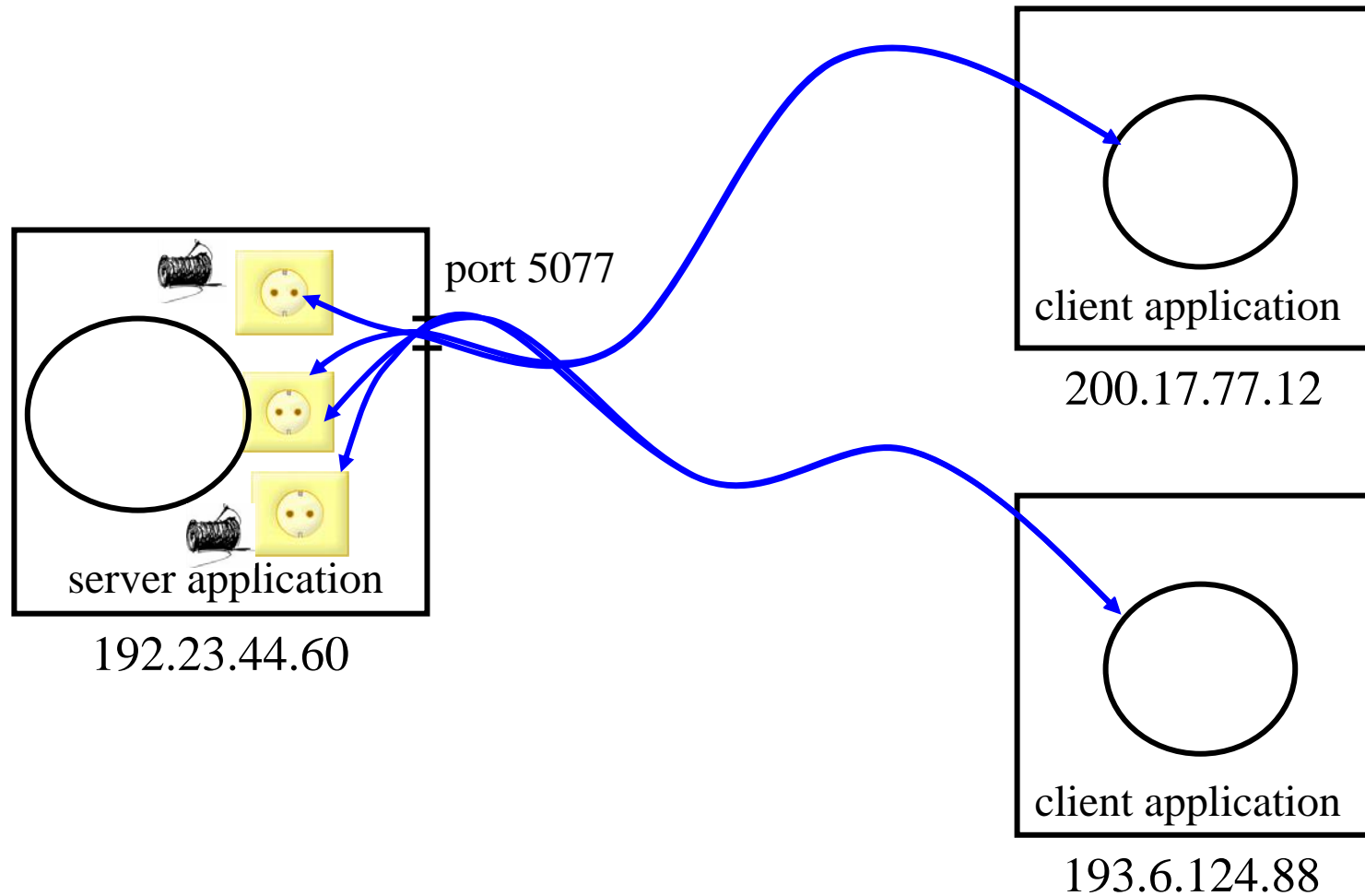
# שיפור המימוש

- שרת שלא יפול אחרי ההודעה הראשונה
- שרת שלא יפול אחרי הלקוח הראשון
- שרת שיודע לטפל בכמה לקוחות במקביל (כמעט שרת chat)
  
- בעיה: בעת הפעולות `accept` (המתנה ללקוח חדש),  
`readLine` (קריאת שורה מזרם) או `print` (הדפסה לזרם) –  
התוכנית נתקעת (`blocked`) ולא ניתן לבצע דברים אחרים  
"במקביל"
- כדי ליצור מקביליות יש להשתמש במנגנון החוטים (`threads`)

# שרת echo

- אז איך נכתוב שרת echo מרובה לקוחות?
- השרת יבצע לולאה של המתנה ללקוחות, עבור כל לקוח חדש ייצור חוט חדש שיטפל בו
- כל החוטים יריצו את אותו השרות – לולאה שממתינה להודעה מהלקוח (כל חוט והלקוח שלו) ועונה לו
- מה חסר כדי להפוך את השרת לשרת chat?

# Multiple clients



```

package examples.sockets;

import java.net.*;
import java.io.*;

class MultiClientEchoServer {

    public static void main(String[] args) {
        ServerSocket serverSocket = null;
        try {
            serverSocket = new ServerSocket(7);
        } catch (IOException ioe) {
            System.err.println("Couldn't listen on port 7");
            System.exit(-1);
        }

        while(true) {
            try {
                Socket clientSocket = serverSocket.accept();
                EchoClientHandler handler =
                    new EchoClientHandler(clientSocket);
                (new Thread(handler)).start();
            } catch (IOException ioe) {}
        }
    }
}

```

```

class EchoClientHandler implements Runnable {
    private Socket clientSocket;

    public EchoClientHandler(Socket clientSocket) {
        this.clientSocket = clientSocket;
    }

    public void run() {
        try {
            PrintWriter out =
                new PrintWriter(clientSocket.getOutputStream(), true);

            BufferedReader in = new BufferedReader(
                new InputStreamReader(clientSocket.getInputStream()));

            String input = null;

            while ((input = in.readLine()) != null) { // read from the client
                out.println(input);                // write to client
            }

            out.close();
            in.close();
            clientSocket.close();
        } catch (IOException ioe) {
            System.err.println("couldn't communicate with client");
            System.exit(-1);
        }
    }
}

```

# למה לא Sockets? (1)

■ **בעיה:** שימוש בפרוטוקול לא סטנדרטי –

- התקשורת אמנם עוברת מעל פרוטוקולים סטנדרטים כגון IP, TCP, Ethernet ואולי אחרים (מאחורי הקלעים)
- אולם היישום עצמו מגדיר פרוטוקול משלו (שלוחה מספר 7, תוכן ההודעה הוא מחרוזת...)
- כדי להשתמש בתוכנית שלנו לקוחות (אנושיים) יצטרכו להריץ בעצמם את תוכנית הג'אווה: EchoClient

■ **פתרונות אפשריים – הרצה בתוך הדפדפן**

- שימוש ביישומונים (Applets) או בטכנולוגיית Java Web Start



## למה לא Sockets? (2)

- הטכנולוגיה תופסת חלק נכבד מהקוד, בעוד הלוגיקה העסקית (מה שהתוכנית באמת עושה) זניחה
- הדבר בולט בישומים פשוטים
- התקשורת והמקביליות הן היבטים (aspects) של היישום ולא נרצה לערב אותן עם הלוגיקה העסקית

# במעלה מחסנית הפרוטוקולים

- Java מאפשרת עבודה ברמות שונות של פרוטוקול התקשורת בהתאם לנדרש ע"י היישום
- המחלקות `Socket` ו-`ServerSocket` לעבודה מעל `TCP`
- המחלקות `DatagramPacket` ו-`DatagramSocket` לעבודה מעל `UDP`
- המחלקה `URL` לעבודה מעל `HTTP` או `FTP`
- מחלקות לעבודה מעל פרוטוקולים מוצפנים כגון `SSLSocket` או `URL`

# המחלקה URL

כדי להקל על כתיבת לקוחות בפרוטוקולים סטנדרטים (http, ftp וכיו"ב) מספקת Java את המחלקה `java.net.URL`

```
// Create some URL objects
URL url = null, url2 = null, url3 = null;
try {
    url = new URL("http://www.oreilly.com"); // An absolute URL
    url2 = new URL(url, "catalog/books/javanut4/"); // A relative URL
    url3 = new URL("http:", "www.oreilly.com", "index.html");
} catch (MalformedURLException e) { /* Ignore this exception */
}

// Read the content of a URL from an input stream
InputStream in = url.openStream();
```

# המחלקה URL

■ ע"י שימוש במחלקה URLConnection ניתן לקבל פרטים נוספים על ההודעה כגון:

```
// For more control over the reading process, get a URLConnection object
URLConnection conn = url.openConnection();

// Now get some information about the URL
String type = conn.getContentType();
String encoding = conn.getContentEncoding();
Date lastModified = new Date(conn.getLastModified());
int len = conn.getContentLength();

// If necessary, read the contents of the URL using this stream
InputStream in = conn.getInputStream();

Scanner s = new Scanner(in);
while(s.hasNextLine())
    System.out.println(s.nextLine());
}
```

# למה לא Sockets? (3)

- למרות העושר שמספקת Java, ביישומים רבים זה אינו מוקד הפיתוח (פרט לשיקולי ביצועים)
- אנו מעוניינים לכתוב את היישום ללא תלות בפרוטוקול התקשורת שמעליו אנו עובדים
- יתר על כן, אנו מעוניינים לתאר את האינטראקציה בין שני עצמים **באותה צורה** בין אם הם נמצאים על אותה מכונה ובין אם הם מתקשרים דרך האינטרנט
- נזכיר כי אחד מיסודות התכנות מונחה העצמים הוא **העברת הודעות** (message passing), כאשר הכוונה להפעלה של מתודות על עצמים

# RMI - הפתרונות - אחד

■ במצגת הבאה נסקור גישה חלופית למימוש תקשורת ביישומי Java

■ בגישת RMI, הטיפול בהבטי התקשורת של המערכת מתבצע ע"י תשתית המסופקת ע"י שפת התכנות

■ בצורה זו, נדרש מהמתכנתת מאמץ מזערי למימוש התקשורת