

תוכנה 1 בשפת Java  
שיעור מספר 13:  
"רשת – לא זזים מהבית"

**סיון טולדו**  
**אוהד ברזילי**

בית הספר למדעי המחשב  
אוניברסיטת תל אביב

# על סדר היום

■ תקשורת בין מחשבים

■ סקירה זריזה ופשטנית – פרטים בקורס מערכות הפעלה ובקורסי תקשורת

■ מבוא לתכנות מרובה חוטים

■ סקירה זריזה ופשטנית – פרטים בקורס מערכות הפעלה

■ שרתי אינטרנט

# 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 לכתיבת תוכניות המדברות בעזרת הפרוטוקול

# פרוטוקול תקשורת

- ניתן לשלב בין פרוטוקולים ע"י שליחת הודעות בתוך הודעות (protocol stack) של המשלוח כל אחת מההודעות מטפלת בהיבט אחר
- 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

- כבר ראינו בתרגול I/O כי ב-Java תוכנית 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();
```

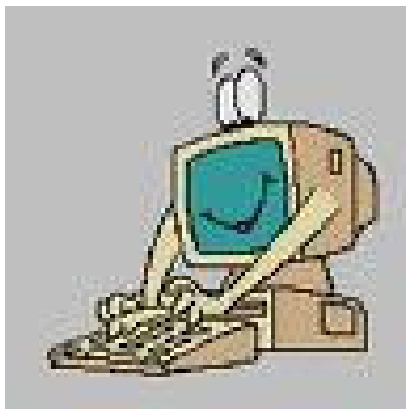
- ננסה להשתמש באותה גישה בדיוק כדי לתקשר עם תוכנית מחשב אחרת (אולי במחשב אחר). לשם כך משתמש במחלקה Socket (שקע)

- 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)



```
ServerSocket s = new ServerSocket(7);  
Socket clientSocket = s.accept();  
BufferedReader in = new BufferedReader(  
    new InputStreamReader(  
        clientSocket.getInputStream()));  
String input = in.readLine();
```

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

```
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 {

    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");
        }
    }
}

```

**EchoServer**

**Buisness Logic**

תוכנה 1 בשפת 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`)

# תכנות מרובה חוטים על קצה המזלג

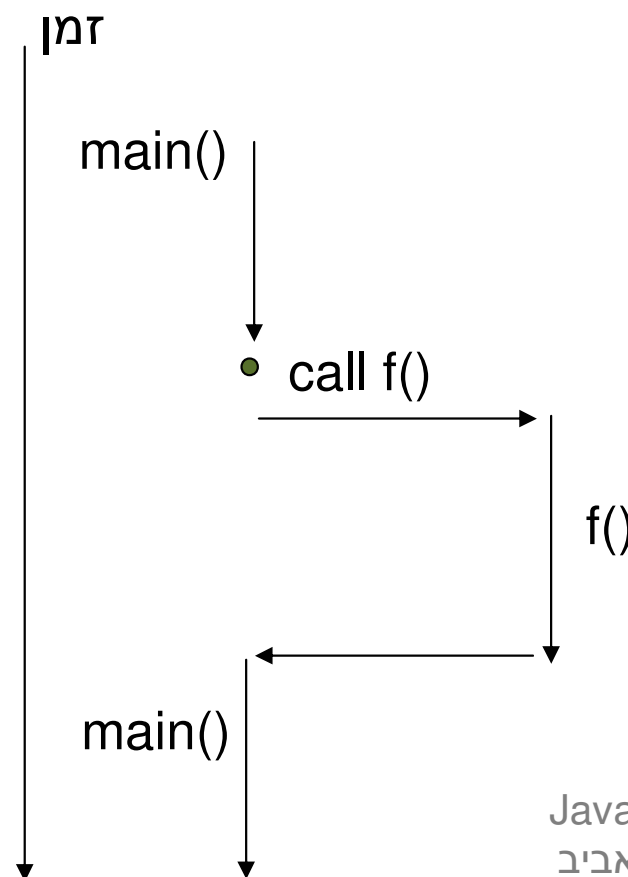


# מקביליות

- רמת התהליך (multithreading) לעומת רמת מערכת ההפעלה (multi processes)
- ריבוי מעבדים (multi processors) לעומת חלוקת זמן עיבוד (time slicing)

# חוטאים

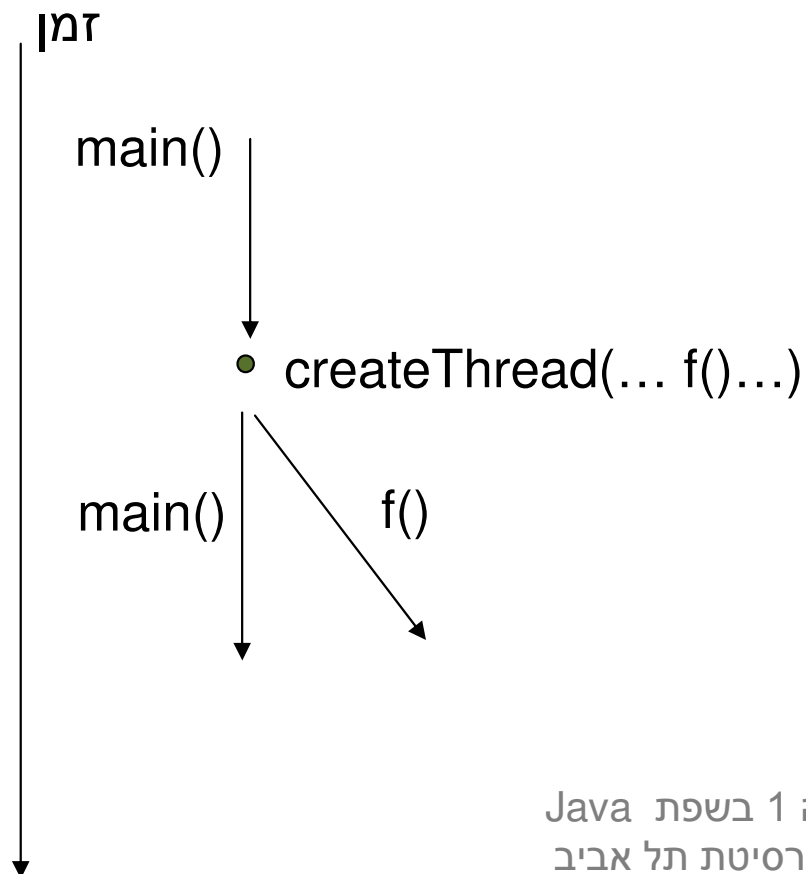
- עד עכשיו ראינו כי שני שרותים אינם רצים ביחד
- כאשר שרות קורא לשרות אחר, מצביע התוכנית "קופץ" לשרות האחר וכאשר הוא מסיים הוא חוזר לשרות הקורא



- למשל אם ב main מופיעה קריאה ל f() ביצוע main נעצר והוא ממשיך רק אחרי סיום f()

# חוטאים

- לעומת זאת אם main יריץ את f() בחוט נפרד – ריצות main ו-f ימשיכו במקביל (!)



# למה חוטים?

- הנדסת תוכנה: מודלריות, הכמסה

- שימוש יעיל במשאבים

- חישוב מבוזר

- משימות אופיניות:

- Non blocking I/O

- Timers

- משימות בלתי תלויות

- אלגוריתמים מקביליים

- דברים ש"רצים ברקע" (Garbage Collection)



# חוטרים ב-Java

■ כמו כל דבר ב-Java, גם חוט ב-Java הוא עצם  
■ מופע של המחלקה Thread

■ חוט תמיד מריץ מתודה עם חתימה קבועה:  
■ `public void run()`  
■ חוץ מהחוט הראשי שמריץ את `main()`

■ מחלקה שממשת את `run()` בעצם מממשת את המנשק  
**Runnable**

■ כלומר, חוט ב-Java הוא מופע של המחלקה Thread שהועבר לו  
כארגומט (למשל בבנאי) עצם ממחלקה שהיא `implements`  
**Runnable**

```

public class ThreadTester {
    public static void main(String args[]) {
        HelloRunner r = new HelloRunner();
        Thread t = new Thread(r);
        t.start();
        // do other things...
    }
}

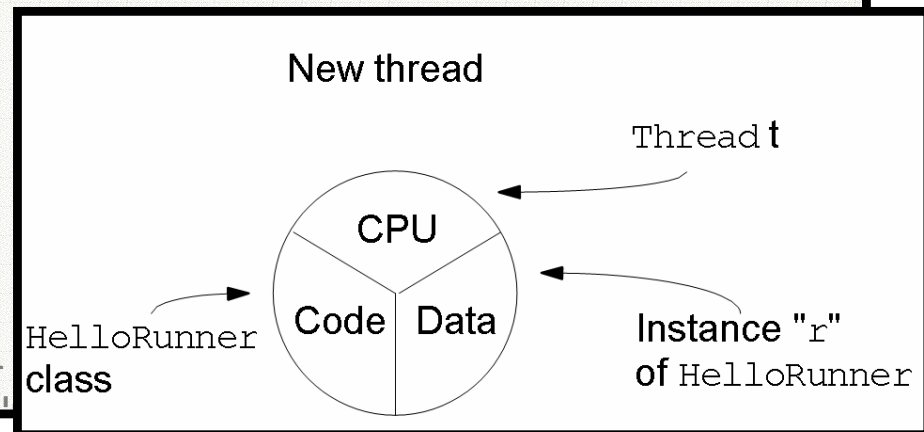
```

```

class HelloRunner implements Runnable {
    int i;

    public void run() {
        i = 0;
        while (true) {
            System.out.println("Hello " + i++);
            if (i == 50) {
                break;
            }
        }
    }
}

```



בשפת Java  
ניתן לרץ אובייקט

# שיתוף מידע

- ההבחנה עדינה – על אותו עצם `Runnable` יכולים לרוץ במקביל 2 חוטים
- הדבר מאפשר לשני החוטים לחלוק מידע ביניהם

```
public class TwoThreadTester {  
    public static void main(String args[]) {  
        HelloRunner r = new HelloRunner();  
        Thread t1 = new Thread(r);  
        Thread t2 = new Thread(r);  
        t1.start();  
        t2.start();  
    }  
}
```

- מה יודפס ?
- כדי להבין טוב יותר מי מדפיס מה, נוסיף פרטים להדפסה

```

public class TwoThreadTesterId {
    public static void main(String args[]) {
        HelloRunnerId r = new HelloRunnerId();
        Thread t1 = new Thread(r, "first");
        Thread t2 = new Thread(r, "second");
        t1.start();
        t2.start();
    }
}

```

```

class HelloRunnerId implements Runnable {
    int i;

    public void run() {
        i = 0;
        while (true) {
            System.out.println(
                Thread.currentThread().getName() +
                    ": Hello " + i++);

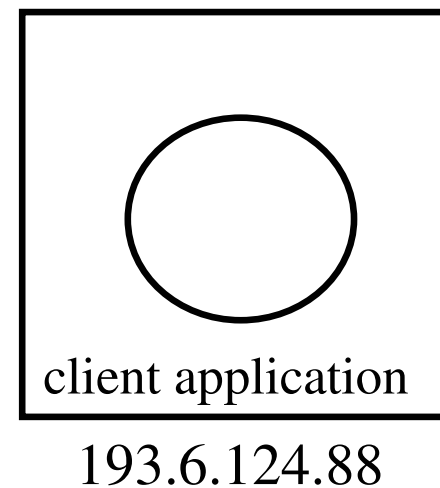
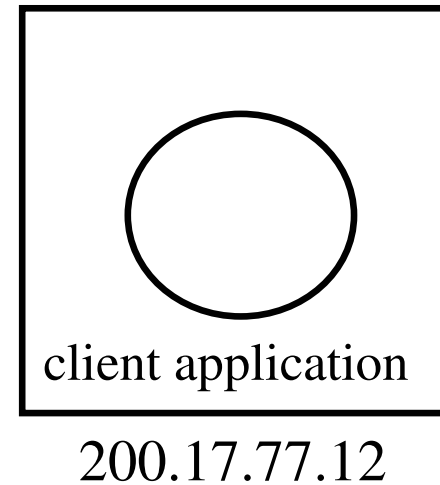
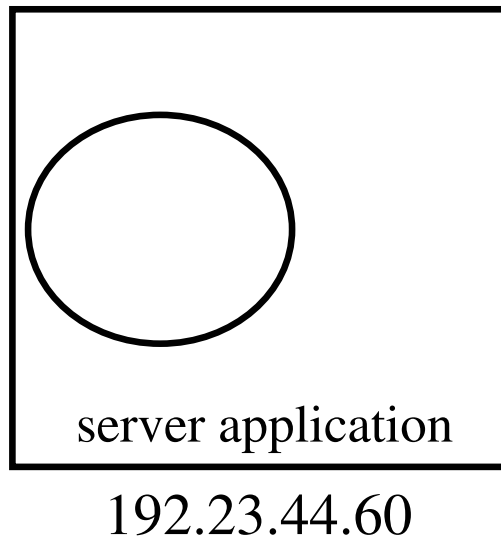
            if (i == 50)
                break;
        }
    }
}

```

# 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
  - לא נדון בהם בקורס אבל אתם מוזמנים לבדוק ב-
    - <http://java.sun.com/products/javawebstart/>
    - <http://java.sun.com/applets/>

## 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

# פתרון אפשרי אחר – עבודה מעל פרוטוקול סטנדרטי

- שימוש בפרוטוקול HTTP (לגלישה ב WEB) יספק לנו תוכנית לקוח חינם - הדפדפן
- שרתי אינטרנט ודפדפנים מתנהגים בצורה דומה מאוד לשתי התוכניות שראינו קודם (עד כדי רדוקציה פשוטה...)
- פרוטוקול HTTP מגדיר את מבנה ההודעות הבא (קיימות גם אחרות):  
`GET <filename> <version>`  
<שורת רווח>
- תוכניות שרת (web servers) משתמשות בשלוחה 80
- כאשר הם מקבלים הודעת GET הם פותחים את הקובץ המתאים ושולחים אותו ללקוח

# אז מה זה HTTP?

- שרת HTTP מחכה לבקשות על Port 80
- בהינתן בקשה, השרת מחזיר תשובה... בעצם קובץ

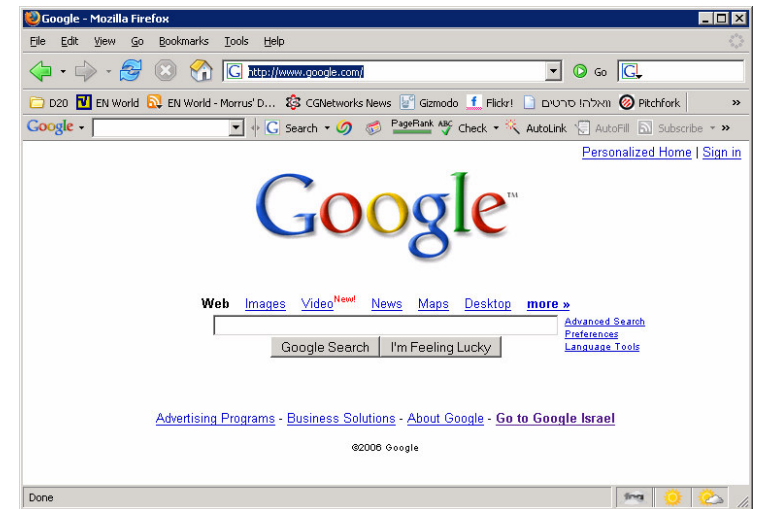


משתמש מקליד כתובת אינטרנט

השרת של GOOGLE מטפל בבקשה



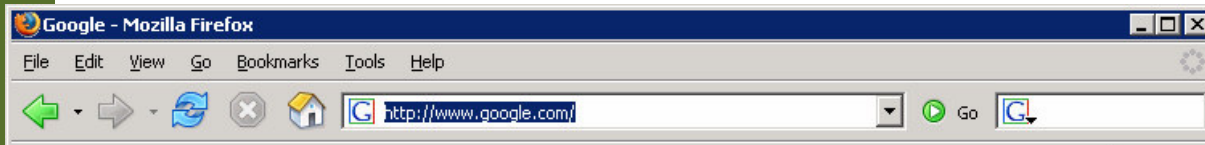
תוכנה 1 בשפת Java  
אוניברסיטת תל אביב



מחזיר דף HTML המוצג בדפדפן

# אז מה זה HTTP?

- שרת HTTP מחכה לבקשות על Port 80
- בהינתן בקשה, השרת מחזיר תשובה... בעצם קובץ



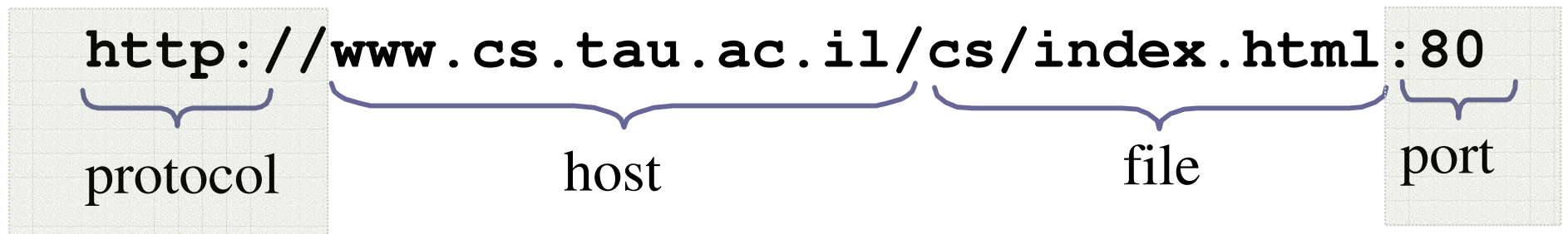
↑  
URL (Uniform Resource Locator)

מייצג משאב כלשהוא ברשת, דף אינטרנט, קובץ, שירות כלשהוא...

# עבודה מעל HTTP

■ איך עובד דפדפן?

■ דפדפנים מחלקים את הכתובת שהוכנסה בשורת הכתובת (URL) ל-5 חלקים, ובונים את ההודעה המתאימה. לדוגמא:



ברירת מחדל

ברירת מחדל

החלק החמישי עשוי מכיל הגדרות משתנים כגון:

<http://www.google.co.il/search?hl=iw&q=cow&btn&meta=>

# שרת ולקוח echo מעל HTTP

- איך נמיר את תוכניות השרת-לקוח שבצעו echo לעבוד מעל HTTP?
- נעשה מניפולציה על פורמט הודעת GET - במקום המיועד להזנת שם הקובץ, נזין את שם המחרוזת שיש להדפיס

■ לקוח:

■ שולח לשלוחה 80 הודעה בפורמט:

`GET /<strToBePrinted> HTTP/1.1`

`<שורה ריקה>`

■ שרת:

■ מאזין על שלוחה 80

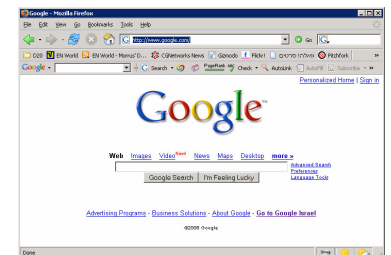
■ מפשיט מההודעה המגיעה את התווים המיותרים ("GET",  
"HTTP/1.1")

■ מחזיר את מה שנשאר בתור התוכן של עמוד html ריק

# HTTP Response

- התשובה המוחזרת מהשרת בעצם קצת יותר מורכבת וכוללת שורות כותרת (headers) אשר אינן מוצגות בדפדפן

```
HTTP/1.1 200 OK
Date: Mon, 20 Feb 2006 03:47:44 GMT
Server: Apache
Last-Modified: Wed, 15 Feb 2006 01:17:09 GMT
ETag: "9b2b1c-948-1222af40"
Accept-Ranges: bytes
Content-Length: 2376
Connection: close
Content-Type: text/html; charset=ISO-8859-1
<שורה ריקה>
<תוכן ההודעה>
```



תוכנה 1 בשפת Java  
אוניברסיטת תל אביב



# למה לא Sockets? (3)

- כעת ניתן להשתמש במוצר מדף – הדפדפן הסטנדרטי
  - את המחרוזת נשרשר לסוף כתובת האינטרנט (יש דרכים אלגנטיות יותר כגון forms שנראה בהמשך)
- האם ניתן להשתמש גם בשרת מדף?
- **בעיה:** שרת האינטרנט אינו דינאמי – הוא יכול להציג רק דפים שהוכנו מראש
  - תיבת הדואר שלי ב Gmail למשל נוצרת דינאמית
- היינו רוצים שרת שיענה על 2 הדרישות הבאות:
  - שידע לטפל בהבטי התקשורת (ואולי גם הבטים אחרים)
  - שיאפשר לנו לכתוב לוגיקה נוספת (ולא רק להחזיר דפים שהוכנו מראש)

# הרעיון

## ■ הפרדה בין:

■ **המסגרת (framework):** החוזרת על עצמה

■ "ההבטים": טיפול בלקוח חדש, יצירת חוט, יצירת שקע ועוד...

■ **הלוגיקה העסקית:** מה התוכנית עושה

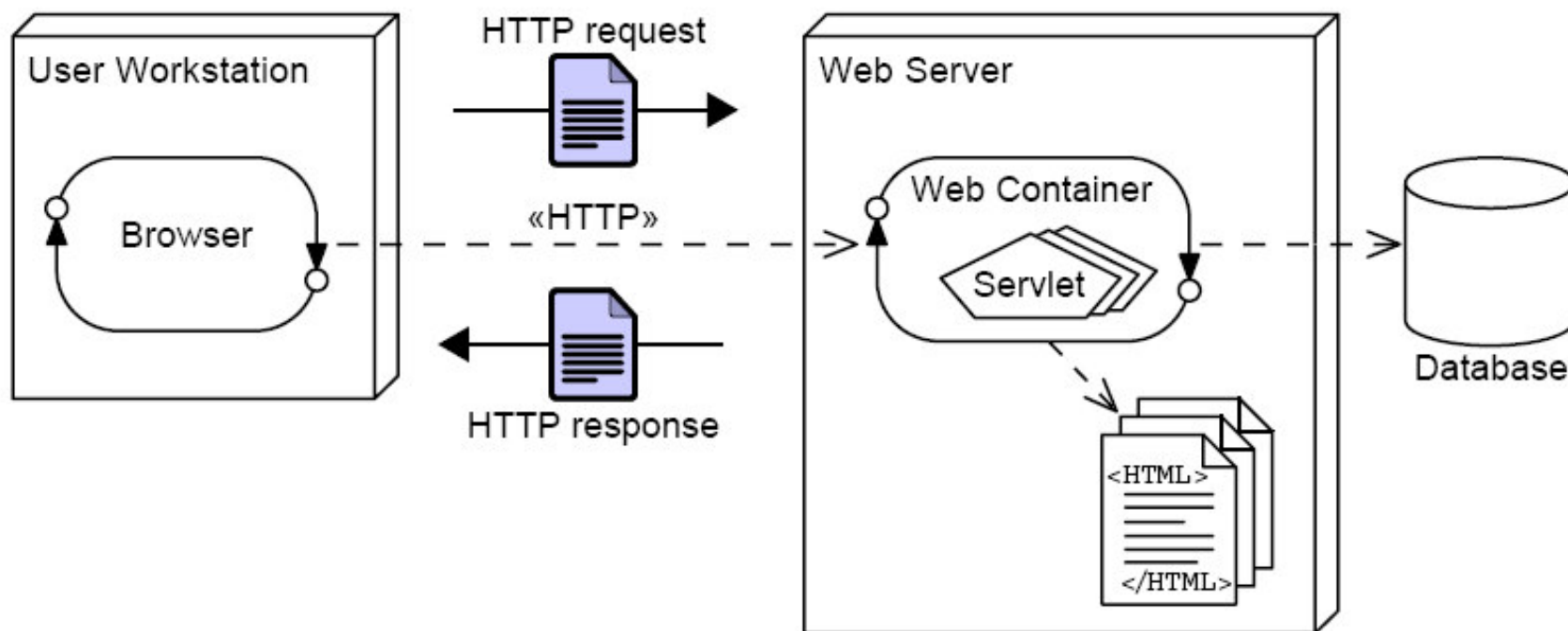
■ בדוגמא בינתיים Echo אבל אפשר לעשות כל דבר

■ את השרת (web container) נכתוב פעם אחת ונטען לתוכו מחלקות Java בשם servlets (שרותונים?) אשר מממשות כל אחת לוגיקה עסקית משלה

■ בפרט נוכל להתקין על אותו שרת כמה שרותים במקביל

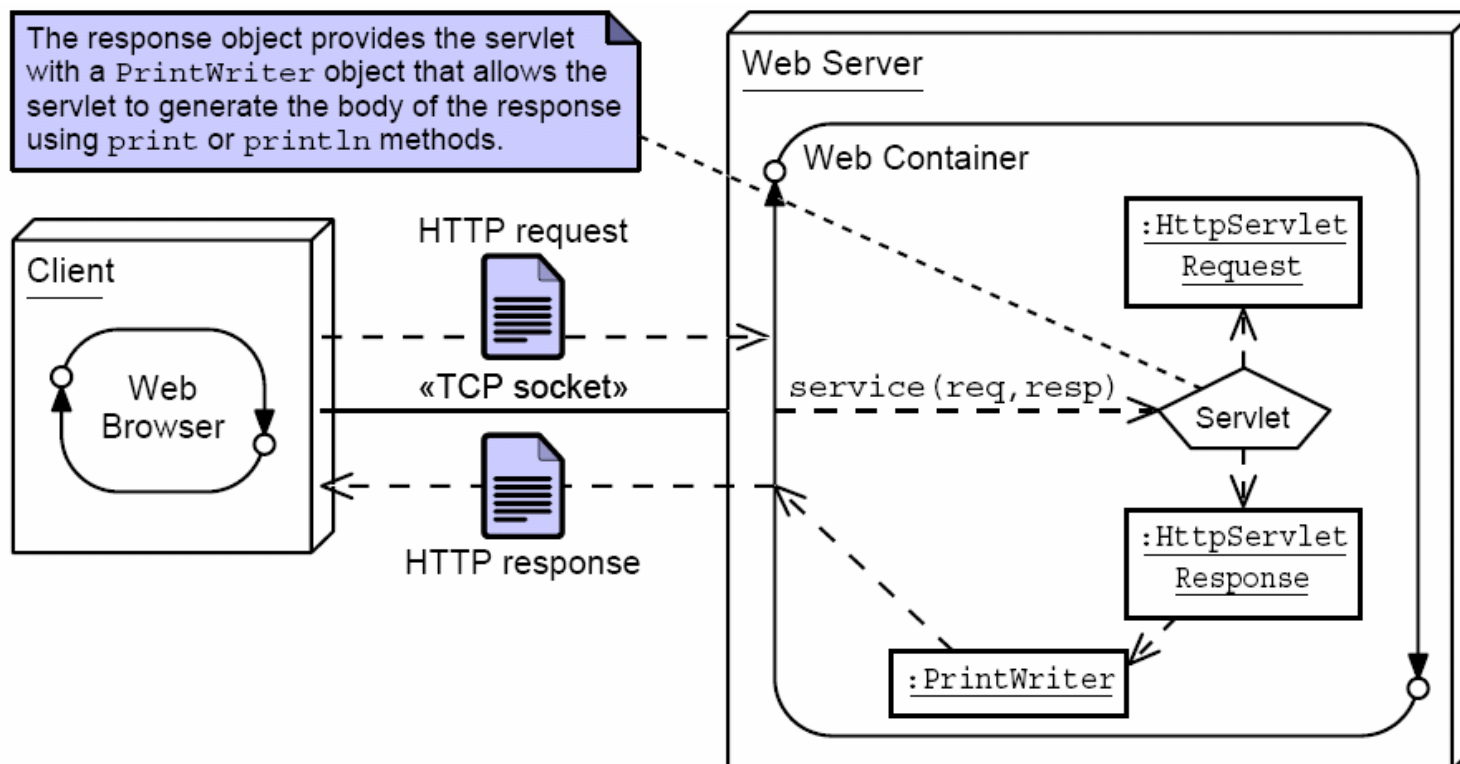
■ כמו כן, נוכל בהמשך להחליף את השרת שכתבנו בשרת מקצועי (Tomcat, Jboss ואחרים) מבלי להחליף את השרותונים

# ארכיטקטורת Web Container



■ גם ה- web container עושה מניפולציה על פורמט הודעת GET - במקום המיועד להזנת שם הקובץ, הוא מצפה לקבל שם מחלקה

# ארכיטקטורת Web Container



על המחלקה הזו הוא יפעיל את השרות `doGet`, (במקרה הכללי `service`) וידאג להעביר לה כפרמטרים **מחלקות עזר** שבעזרתם תקרא את הפרמטרים אם הועברו כאלה בשורת הכתובת, ותייצר הודעת תשובה

# HttpServletRequest, HttpServletResponse

■ שרותי מחלקות העזר מוגדרים ע"י המנשקים הבאים:

```
public interface HttpServletRequest {  
    public java.util.Set<String> getParameterNames();  
    public String getParameter(String name);  
}
```

```
public interface HttpServletResponse {  
    public java.io.PrintWriter getWriter();  
    public void setContentType(String type);  
}
```

# מימוש מחלקות העזר

```
public class ServletInvocation
    implements HttpServletRequest, HttpServletResponse {

    private java.util.Map<String,String> parameters;
    private PrintWriter writer;

    public ServletInvocation(java.util.Map<String,String> parameters,
        PrintWriter writer) {
        this.parameters = parameters;
        this.writer      = writer;
    }

    public String getParameter(String name) { return parameters.get(name); }

    public Set<String> getParameterNames() { return parameters.keySet(); }

    public PrintWriter getWriter() { return writer; }

    public void setContentType(String type) {
        writer.print("HTTP/1.1 200\r\n");
        writer.print("Content-Type: "+type+"\r\n");
        writer.print("Connection: close\r\n");
        writer.print("\r\n"); // end of header
    }
}
```

# TrivialServlet

- נתחיל בדוגמא פשטנית: נכתוב את השרת `HttpServer` (שנראה אחר כך) ונתקין בו את השרותון `TrivialServlet` אשר מדפיס את רשימת הפרמטרים שקיבל
- שרותון הוא Component – הוא מקבל במתנה מהמיכל (מה-`web container`) את כל הדברים שהוא זקוק להם (`aspects`)
- עליו לממש רק לוגיקה עסקית בתוך השרות `doGet` (`hook`)

# TrivialServlet

```
package examples.servletserver;

public class TrivialServlet extends HttpServlet {

    public TrivialServlet() {
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) {
        java.io.PrintWriter writer = resp.getWriter();
        resp.setContentType("text/plain");

        java.util.Set<String> parameters = req.getParameterNames();

        writer.println("This is servlet "+this.getClass().getName());
        for (String pname: parameters) {
            writer.println("parameter "+pname+" = "+req.getParameter(pname));
        }
    }
}
```



```

public class FileServlet extends HttpServlet {

    public FileServlet() {
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) {
        java.io.PrintWriter writer = resp.getWriter();
        String path = req.getParameter("path");
        if (path == null)
            return;

        // we really should be more clever
        if (path.endsWith(".html"))
            resp.setContentType("text/html");
        else
            resp.setContentType("text/text");

        try {
            BufferedReader in = null;
            String servlet = null;
            java.util.Map<String,String> parameters = null;

            in = new BufferedReader(new InputStreamReader(new FileInputStream(path)));

            while (true) {
                String line = in.readLine();
                if (line==null) break; // end of file
                writer.println(line);
            }
        } catch (IOException ioe) {
            System.err.println(ioe.getMessage());
        }
    }
}

```

מה עושה ה Servlet הזה?

# ChatServlet

■ השרותונים הם Singletons

■ אם שני לקוחות (דפדפנים) ניגשים לאותו ה Servlet הפניות מופנות לאותו עצם בזכרון

■ ננצל עובדה זו כדי לכתוב ChatServlet

■ השרותון מציג טופס html (form)

■ ערכי השדות שהמשתמש מקליד משורשרים כפרמטרים לשורת הכתובת והעמוד נטען בשנית

■ השרותון מציג בנוסף לטופס את היסטורית השיחה (השמורה בפרמטרים)

```

public class ChatServlet extends HttpServlet {

    private String[] lastMessages = new String[10];
    private String[] lastNames    = new String[10];

    public ChatServlet() {
    }

    @Override
    protected void doGet (HttpServletRequest req, HttpServletResponse resp) {
        java.io.PrintWriter writer = resp.getWriter();
        String name = req.getParameter("name");
        String text = req.getParameter("text");

        resp.setContentType("text/html");

        if (text != null && text.length() > 0) {
            for (int i=9; i>0; i--) {
                System.out.println("i="+i);
                lastMessages[i] = lastMessages[i-1];
                lastNames      [i] = lastNames      [i-1];
            }
            lastMessages[0]= text;
            lastNames[0]   = name == null || name.length()==0 ? "anonymous" : name;
        }
    }
}

```

```

writer.println("<HTML><HEAD>");
writer.println("<TITLE>Chat Servlet</TITLE>");
writer.println("</HEAD><BODY>");
writer.println("<H1>Let 's Chat!</H1>");

writer.println("<form name=\"input\" action=\"\"
                +this.getClass().getName()
                +\"\" method=\"get\">");
writer.println("Name: <input type=\"text\" name=\"name\"");

if (name != null && name.length() > 0)
    writer.println(" value=\""+name+"\"");
else
    writer.println(">");

writer.println(" Message: <input type=\"text\" name=\"text\"");
writer.println(" <input type=\"submit\" value=\"Send\"");
writer.println("</form>");

for (int i=0; i<10; i++)
    if (lastNames[i] != null && lastMessages[i] != null)
        writer.println("<P>"+lastNames[i]+": " +lastMessages[i]+"</P>");

writer.println("</BODY></HTML>");
}
}

```

# מכיוון שכל השרותונים הם Singletons הנטענים דינאמית, מופיע מימוש תכונות אלו במחלקת הבסיס, `HttpServlet`

```
public abstract class HttpServlet {

    private static java.util.Map<String, HttpServlet> servlets =
        new java.util.TreeMap<String, HttpServlet> ();

    static HttpServlet getInstance(String class_name) {
        HttpServlet servlet = servlets.get(class_name);
        if (servlet != null)
            return servlet;

        try {
            java.lang.reflect.Constructor constructor =
                Class.forName(class_name).getConstructor(new Class[] { });

            servlet = (HttpServlet) constructor.newInstance(new Object[] { });
        } catch (Exception e) {
            return null;
        }

        servlets.put(class_name, servlet);
        return servlet;
    }

    protected HttpServlet() {}

    protected abstract void doGet(HttpServletRequest req, HttpServletResponse resp);
}
```

# פעולת השרת: `HttpServer`

1. יצירת שקע והמתנה ללקוחות
2. עבור כל לקוח חדש (`accept`) בצע:
  1. קרא את השורה הראשונה בהודעה (`header`)
  2. אם מדובר בהודעת `GET`:
    1. נתח את המחרוזת במקום המיועד לשם הקובץ וחלץ ממנה את שם ה `Servlet` המבוקש ואת שמות המשתנים וערכיהם (לפי התווים: `'/'`, `'='`, `'&'`, `'?'`)
    2. קבל הפנייה לשרותון המתאים (`getInstance`)
    3. צור את מחלקות העזר (`HttpServletRequest`, `HttpServletResponse`)
    4. אם השרותון נמצא:
      - קרא ל `doGet` על השרותון עם מחלקות העזר המתאימות אחרת
      - הצג הודעת שגיאה

# יצירת שקע והמתנה ללקוחות

```
public class HttpServer {
    public static void main(String[] arguments) {
        int port = 8888;

        ServerSocket server_socket = null; // dummy initialization
        try {
            server_socket = new ServerSocket(port);
        } catch (IOException ioe) {
            System.out.println("Failed to bind to port "+port+": "+ioe.getMessage());
            System.exit(1);
        }

        while (true) {
            Socket connection_socket = null;
            try {
                connection_socket = server_socket.accept();
            } catch (IOException ioe) {
                System.out.println("Accept failed: "+ioe.getMessage());
                System.exit(1);
            }

            BufferedReader in = null;
            String servlet = null;
            java.util.Map<String,String> parameters = null;

            try {
                in = new BufferedReader(
                    new
InputStreamReader(connection_socket.getInputStream()));
            }
```

# ניתוח שורת הכותרת

```
String line;

while (true) {
    line = in.readLine();
    if (line==null)          break;
    if (line.length() == 0) break;

    if (line.startsWith("GET ")) {
        String get_params[] = line.split(" ");
        String path = get_params[1];

        servlet = path.substring(1);
        int index_of_question = servlet.indexOf('?');
        int index_of_slash    = servlet.indexOf('/');

        parameters = new java.util.TreeMap<String, String>();
        if (index_of_question != -1) {
            String params = servlet.substring(index_of_question+1);
            String[] params_values = params.split("&");
            for (String pv: params_values) {
                String[] p_and_v=pv.split("=");
                parameters.put(p_and_v[0],
                               p_and_v.length==2 ? p_and_v[1] : "");

                servlet = servlet.substring(0,index_of_question);
            }
        } else if (index_of_slash != -1) {
            String path_param = servlet.substring(index_of_slash+1);
            parameters.put("path",path_param);
            servlet = servlet.substring(0,index_of_slash);
        }
    }
}
```



```

PrintWriter out = null;
try {
    out = new PrintWriter(connection_socket.getOutputStream());

    } catch (IOException ioe) {
        System.out.println("Could not send the output: "+ioe.getMessage());
        System.exit(1);
    }

HttpServlet s = HttpServlet.getInstance(servlet);

if (s != null) {
    ServletInvocation invocation = new ServletInvocation(parameters, out);
    s.doGet(invocation, invocation);
} else {
    out.print("HTTP/1.1 404\r\n");
    out.print("Content-Type: text/html\r\n");
    out.print("Connection: close\r\n");
    out.print("\r\n"); // end of header
    out.println("<HTML><HEAD>");
    out.println("<TITLE>404 Not Found</TITLE>");
    out.println("</HEAD><BODY>");
    out.println("<H1>Not Found</H1>");
    out.println("<P>The requested URL was not found on this server.</P>");
    out.println("</BODY></HTML>");
}

try {
    out.close();
    in.close();
    connection_socket.close();
} catch (IOException ioe) {
    // who cares?
}
}

```

## הפעלת השרותון המתאים

# J2EE Web Containers

■ שרתי Web דומים מאוד לאלו שהצגנו כאן, קיימים בצורה מסחרית (חלקם בתשלום וחלקם חופשיים)

■ לדוגמא:

- Apache Tomcat
- Sun Java System Web Server
- Oracle Containers for Java (OC4J)
- Many more...

■ שרתים אשר מממשים את תקן ה Servlet במלואו, מאפשרים יבילות לכותבי השרותונים

■ אנו נוכל להריץ את ה Servlets שכתבנו על כל שרת הממלא את התקן Servlet 2.4 Specification JSR-000154 JavaTM