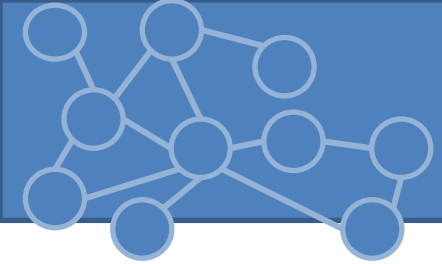


Laboratorio Reti di Calcolatori

Laurea Triennale in Comunicazione Digitale

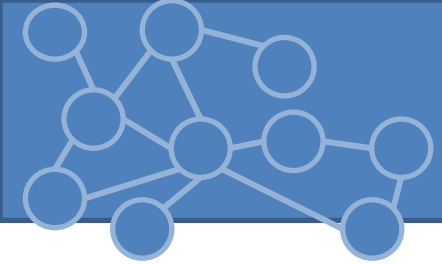
Anno Accademico 2013/2014

Servizio Web

- Analizziamo strategie per accedere al principale dei servizi su Internet: Web
  - Attenzione sullo scambio di messaggi per la fruizione del servizio
- Web non funzionale a scaricare e visualizzare pagine HTML MA è rivolto all'interrogazione di risorse a cui potrebbero essere associati contenuti
- Procedura della fruizione semplice:
  1. Client si collega al server e richiede l'accesso ad una risorsa identificata con URI
  2. Server fornisce una risposta forse corredata da un contenuto e chiude la connessione.
  - Porta: 80 e protocollo a livello applicazione: HTTP => web server
- Attenti: la risorsa può non
  - Essere associata ad una pagina web
  - Essere localizzata sullo stesso nodo del server
  - Esistere al momento della connessione

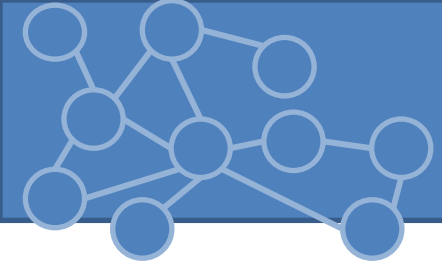
Protocollo HTTP

- HTTP (RFC 1945/2616) definisce modo in cui client richiede una risorsa al server web e come server trasferisce risorsa al client attraverso dei messaggi
- Utilizza TCP (☺)
- Server non memorizza informazioni di stato circa il client: **protocollo stateless.**
- 2 tipi di messaggi: richiesta e risposta
  - Composti da serie di stringhe ASCII terminate dalla sequenza «\r\n» seguite dal contenuto da associato o da associare alla risorsa
- Messaggio composto da:
  - Stringa iniziale
    - Distingue una richiesta da una risposta
  - Header (serie di opzioni)
    - Formato del tipo: <campo: valore>
    - Definiscono i parametri per il funzionamento del client e server. C e S possono definire nuove opzioni. I campi ricevuti ma non supportati vengono ignorati
  - Stringa vuota per segnalare la fine dell'header
  - Contenuto in formato binario



Messaggio di richiesta(1)

- Linea iniziale (riga di richiesta): 3 elementi intervallati da spazi
  - METODO URL VERSIONE
- Metodo di accesso: modalità con cui si richiede di accedere all'URL
- URL specificabile sia in maniera relativa sia assoluta.
  - Relativo: omessi protocollo (sottointeso) e nome dell'host (nodo verso cui aperta comunicazione)
  - Si raccomanda di usare sempre URL assoluti per favorire il server
- VERSIONE ha formato «HTTP/A.B» e specifica la versione del protocollo implementata
  - Quali opzioni il client riconosce e sulle modalità gestione del canale



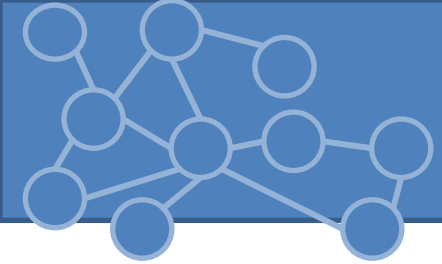
Messaggio di richiesta(2)

  - Ver 1.0 prevede la chiusura del canale appena terminato il messaggio di risposta. Ver 1.1 tiene la connessione aperta fino alla scadenza di un timeout (chiusura socket evento potenzialmente asincrono)
- Righe di intestazione: opzioni più comuni
  - **Host**: indirizzo e porta utilizzati per contattare il server (obbligatori con ver 1.1)
  - **Content-Type**: specifica il formato del contenuto. Identificatore secondo formato MIME (obbligatorio se invio al server del contenuto)
  - **Content-Length**: specifica dimensione in byte dei dati che seguiranno l'header (obbligatorio se inviati contenuti al server)
  - **User-Agent**: informazioni riguardo il client: tipi e versione browser e SO. Per personalizzazione servizio o raccolta statistiche.

Messaggio di risposta

- Linea iniziale: 3 elementi intervallati da spazi
  - VERSIONE CODICE COMMENTO
- VERSIONE: stessa funzione dell'equivalente del messaggio di richiesta
- CODICE: risultato della richiesta. Numero decimale di tre cifre. La prima identifica una classe di risposte

Codice	Significato della classe
1xx	Solo ver 1.1. Manda informazioni addizionali al client sotto forma di messaggi di risposta
2xx	Buon fine
3xx	Risorsa esistente ma raggiungibile tramite un URL diverso
4xx	Richiesta non appropriata: risorsa non esistente (404), non può accedere alla risorsa (403), formato non corretto (400)
5xx	Errore interno al server

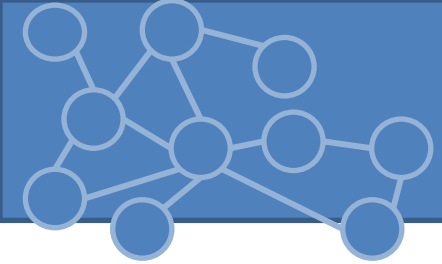


- Terzo elemento = stringa di commento fornita al client (se c'è errore fornisce una descrizione).
- Opzioni più usate
  - **Content-Type**: stesso significato del messaggio di richiesta
  - **Content-Length**: Facoltativo, se non specifico il contenuto si intende finito con la chiusura del canale.
  - **Last-Modified**: quando contenuto modificato l'ultima volta (uso o meno della cache)
  - **Server**: informazioni sul server

Acceso ad una risorsa

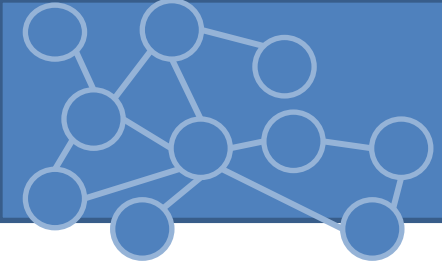
- Nella richiesta si deve indicare il metodo di accesso. Serie di metodi di accesso

Metodo	Operazione associata
Options	Richiede modalità fruizione di una risorsa (metodi di accesso utilizzabili,...)
Get	Accede al contenuto associato alla risorsa
Head	Messaggio di risposta è privo di contenuto
Post	Accede al contenuto fornendo informazioni aggiuntive
Put	Richiede che contenuto che segue header venga memorizzato
Delete	Richiede la cancellazione di una risorsa
Connect	Solo con un proxy. Richiesta di deviazione della connessione corrente verso altro indirizzo di trasporto
Trace	Il contenuto della risposta è il messaggio di richiesta

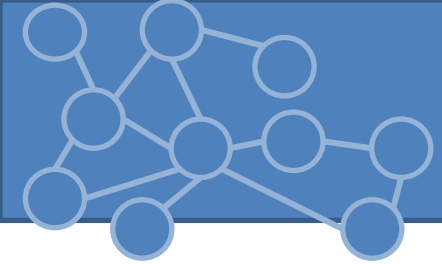


Contenuti statici e dinamici

- Contenuti cui posso fruire usando HTTP si dividono in due categorie
  - Statico: sempre lo stesso
  - Dinamico: varia nel tempo o in base al contesto
    - url di un motore di ricerca
    - I parametri influenzanti il contenuto possono essere indipendenti da utente o meno
- Fornisco parametri di funzionamento ad una risorsa dinamica tramite GET aggiungendoli in coda all'URL => aggiungo «?» seguito da coppie «parametro=valore» intervallate da «&»
- Rende gli URL illeggibili, bookmark non più validi

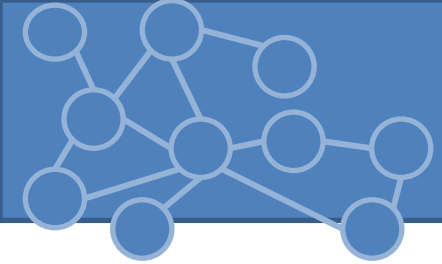


- POST usato da un web browser nel momento in cui l'utente sottomette un form all'interno di una pagina web
- I parametri vengono spediti all'interno del contenuto del messaggio
  - Content-type e Content-Length diventano necessarie
    - Content-type più usati: 'application/x-www-form-urlencoded' (sottomissione form) e 'application/octet-stream' (inviati dati binari)



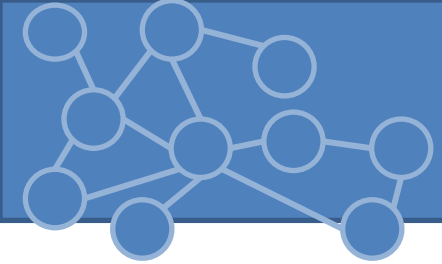
URLConnection

- *URLConnection* è una classe astratta che rappresenta una connessione attiva ad una risorsa specificata da un oggetto URL
- Scopi:
  1. Maggior controllo sull'interazione con un server rispetto alla classe URL
     - Ispeziono header inviato dal server
     - Modifico i campi nell'header di un messaggi di richiesta
     - Scaricare file binari
     - Posso inviare dati al server usando POST e PUT
  2. Parte meccanismo di protocol handler => separo dettagli relativi al processamento di un protocollo, dal processamento dati
- Maggior parte dei metodi e dei campi sono protected, come il costruttore.
- Si basa sulla classe Socket => posso fare essenzialmente le stesse cose ma URLConnection è più facile da usare e fornisce un'astrazione a più alto livello



Apertura URLConnection

- Sequenza base di passi
  1. Costruzione di un oggetto URL
  2. Invoco metodo openConnection() che mi restituisce un oggetto URLConnection
  3. Configuro una URL Connection
  4. Leggo i campi dell'header di risposta
  5. Uso input e output stream per leggere/scrivere dati
  6. Chiudo la connessione
- Non sempre tutti sono necessari
- Il costruttore di URLConnection è protetto, lo posso invocare se creo un a sottoclasse
- Posso usare l'oggetto restituito dal metodo openConnection() della classe URL



Apertura URLConnection(2)

- Tutti tranne un metodo sono implementati: *public abstract void connect() throws IOException*

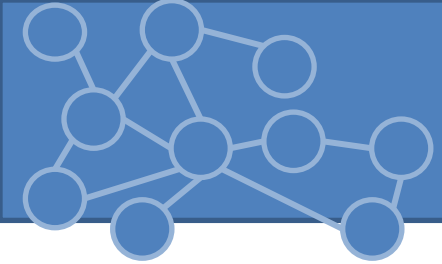
che esegue una connessione verso un server e quindi dipende dal tipo di servizio.

  - FileURLConnection converte URL in un nome di file e apre un FileInputStream verso il file
- Quando URLConnection viene costruita non è connessa, la connect() stabilisce una connessione
  - getInputStream, getContent() e altri metodi che richiedono una connessione aperta chiamano una connect() se la connessione non è ancora stata aperta

Leggere Dati dal Server

- Dopo la chiamata alla openConnection() mi faccio restituire uno stream con getInputStream() e poi leggo nel modo usuale

```
ublic class HTMLViewer {

    public static void main(String[] args) {
        if(args.length > 0){
            try{
                URL u = new URL(args[0]);
                URLConnection uc = u.openConnection();
                Reader r = new InputStreamReader(uc.getInputStream());
                int c;
                while((c=r.read())!=-1){
                    System.out.print((char) c);
                }
            }catch(MalformedURLException mue){
                mue.printStackTrace();
            }catch (IOException ioe) {
                ioe.printStackTrace();
            }
        }
    }

}
```

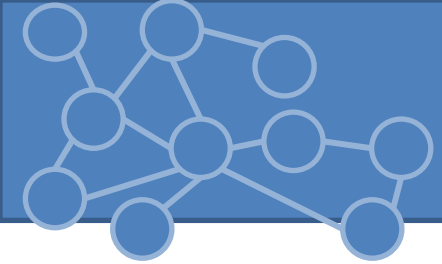


Lettura Header

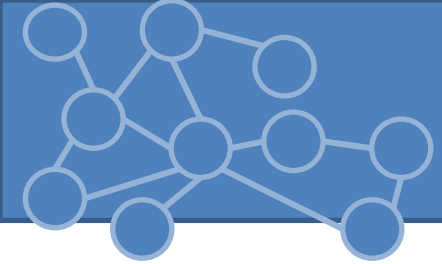
- Con URLConnection posso accedere agli header HTTP, configurare i parametri di richiesta da inviare al server, scrivere e leggere dati verso/dal server.
- **public String getContentType()**
  - Restituisce il tipo MIME dei dati
  - Nessuna eccezione, se l'opzione non è disponibile restituisce null
  - Esempi più comuni: text/plain, image/gif, application/xml, image/jpeg
  - Se il contenuto è un testo header può contenere un charset
    - Content-type: text/html; charset=UTF-8
    - Posso sfruttarlo per la codifica

- **public int getContentLength()**
  - Restituisce la dimensione in byte del contenuto. Molti server mandano l'opzione Content-length solo quando stanno trasferendo un file binario
  - Se non c'è il content-length viene restituito -1
  - Web server non sempre chiudono la connessione quando i dati sono finiti. Per scaricare un file binario leggo il numero di byte che mi vengono restituiti

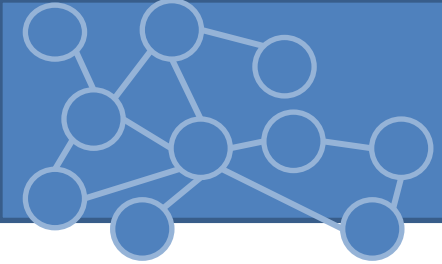
```
while(offset < contentLength){
    byteRead = is.read(data,offset,data.length-offset);
    if(byteRead == -1) break;
    offset +=byteRead;
}
is.close();
if(offset!=contentLength){
    throw new IOException("Letti "+offset+" bytes. Attesi "+contentLength+" bytes");
}
String filename = root.getFile();
filename = filename.substring(filename.lastIndexOf('/')+1);
FileOutputStream fout = new FileOutputStream(filename);
fout.write(data);
```



- **public String getContentEncoding()**
  - Restituisce una stringa contenente la modalità di codifica del contenuto, se il contenuto non è codificato (HTTP) il metodo restituisce null
  - Il valore più comune è x-gzip che può essere decodificato usando java.util.zip.GZipInputStream
- **public long getDate()**
  - Restituisce quando il documento è stato mandato (millisecondi dal 1 gennaio 1970) dal punto di vista del server
- **public long getExpiration()**
  - Data che indica quando un documento dovrebbe essere rimosso dalla cache e richiesto al server. Il valore long indica il numero di millisecondi dopo la solita data
- **public long getLastModified()**
  - Restituisce l'ultima data di modifica della risorsa

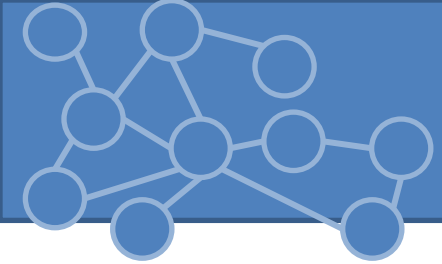


- **public String getHeaderField(String name)**
  - Restituisce il valore di un campo indicato dalla chiave name. Name non è case-sensitive e non include ‘:’
- **public String getHeaderFieldKey(int n)**
  - Restituisce la chiave dell’n-esimo campo. Il metodo di richiesta ha valore 0 e chiave nulla => Il primo campo utile è il numero 1
- **public String getHeaderField(int n)**
  - Restituisce il valore dell’n-esimo campo

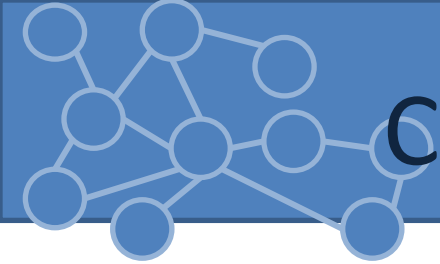


Configurare la connessione

- URLConnection ha sette campi protected che definiscono come il client esegue la richiesta al server. I valori possono essere modificati o letti usando metodi getter e setter
  - URL url
  - Boolean doInput
  - Boolean doOutput
  - Boolean allowUserInteraction
  - Boolean useCaches
  - Long ifModifiedSince
  - Boolean connected
- Si possono modificare questi campi solo PRIMA della connessione. Molti sollevano IllegalStateException
- URL
  - Solo un get => una volta creata non posso cambiare l'URL
- Connected
  - True se la connessione è aperta => ogni metodo che apre una connessione deve settare questo valore a true



- doInput
  - Settato a true se URLConnection può essere usata per l'input
  - Default: true
- doOutput
    - Client può usare URLConnection per fare output verso il server cambiando il valore del campo a true
  - Quando viene posto a true il campo con un URL http, il metodo di richiesta è cambiato da GET a POST

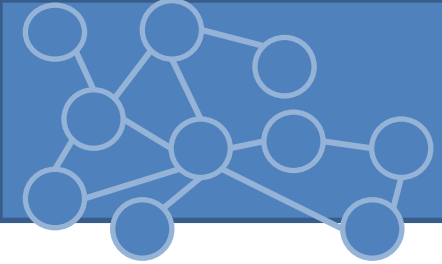


Configurare una richiesta HTTP

- Da Java 1.3 posso aggiungere campi all'header usando il metodo

  public void setRequestProperty(String name,String value)
- Può essere usato solo PRIMA l'apertura della connessione. Solleva una IllegalStateException se connessione già aperta
- HTTP permette che una chiave abbia più valori separati da un virgola. Per aggiungere una valore ad una chiave uso

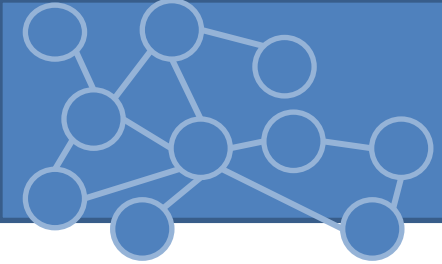
  public void addRequestProperty(String name,String value)
- I server ignorano i campi che non riconoscono + HTTP impone alcune restrizioni sulle chiavi e sui valori
  - Chiave non può contenere spazi
  - Valore non può contenere caratteri di terminazione



Il metodo

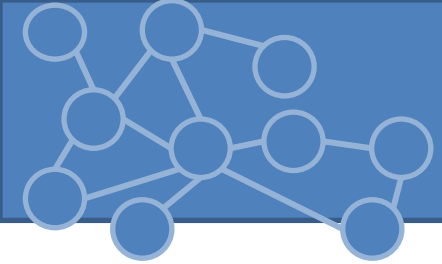
public OutputStream getOutputStream()

- Permette di scrivere dati sul server. Si deve invocare setDoOutput(true) prima di usare il metodo.
- Devo ricordarmi di settare i campi Content-type e content-length



HttpURLConnection

- E’ una sottoclasse astratta di URLConnection che rende disponibili alcuni metodi utili nel trattare URL di tipo HTTP
- Non posso creare direttamente un’istanza di HttpURLConnection. Costruendo un oggetto URL usando un URL HTTP e invocando openConnection() l’oggetto restituito è un’istanza di HttpURLConnection
  - Void setRequestMethod(String method)
    - In base al valore deve settare alcuni campi dell’header ed inserire un contenuto
  - String getResponseMessagge()
  - int getResponseCode()

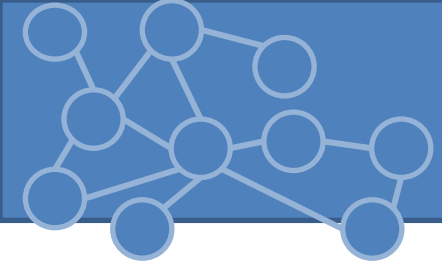


Messaggio di richiesta

- Ottenuta un'istanza di HttpURLConnection posso impostare i parametri con cui effettuare una richiesta.
- Da Java 1.3 posso aggiungere campi all'header usando il metodo

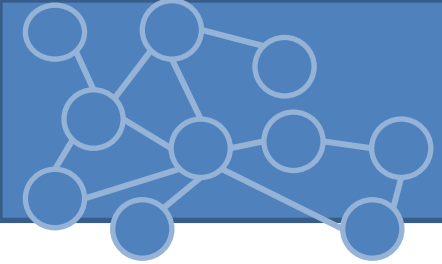
  public void setRequestProperty(String name,String value)
- Può essere usato solo PRIMA l'apertura della connessione. Solleva una IllegalStateException se connessione già aperta
- HTTP permette che una chiave abbia più valori separati da un virgola, punto e virgola o spazio. Per aggiungere una valore ad una chiave uso

  public void addRequestProperty(String name,String value)
- I server ignorano i campi che non riconoscono + HTTP impone alcune restrizioni sulle chiavi e sui valori (IllegalArgument)
  - Chiave non può contenere spazi
  - Valore non può contenere caratteri di terminazione
- Apro la connessione con il server invocando il metodo connect()



- Per ispezionare gli header di una HttpURLConnection:
  - String getRequestProperty(String name)

```
public class WebClient {

    public static void main(String[] args) {
        try{
            URL url = new URL(args[0]);
            HttpURLConnection huc = (HttpURLConnection) url.openConnection();
            huc.setRequestProperty("User-Agent", "Client fatto in casa v1.0");
            huc.connect();
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}
```



Messaggio di risposta

- Dopo la connect è stata effettuata la connessione al server + analizzati intestazione e headers
- Con HttpURLConnection posso accedere agli header HTTP
- **public String getContentType()**
  - Restituisce il tipo MIME dei dati
  - Nessuna eccezione, se l'opzione non è disponibile restituisce null
  - Esempi più comuni: text/plain, image/gif, application/xml, image/jpeg
  - Se il contenuto è un testo header può contenere un charset
    - Content-type: text/html; charset=UTF-8
    - Posso sfruttarlo per la codifica (esempio)

public int getContentLength()

- Restituisce la dimensione in byte del contenuto. Molti server mandano l'opzione Content-length solo quando stanno trasferendo un file binario
- Se non c'è il content-length viene restituito -1
- Web server non sempre chiudono la connessione quando i dati sono finiti. Per scaricare un file binario leggo il numero di byte che mi vengono restituiti

public String getContentEncoding()

- Restituisce una stringa contenente la modalità di codifica del contenuto, se il contenuto non è codificato (HTTP) il metodo restituisce null
- Il valore più comune è x-gzip che può essere decodificato usando java.util.zip.GZipInputStream

public long getDate()

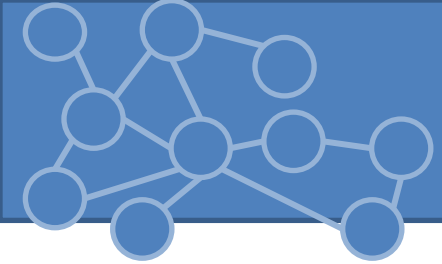
- Restituisce quando il documento è stato mandato (millisecondi dal 1 gennaio 1970) dal punto di vista del server

public long getExpiration()

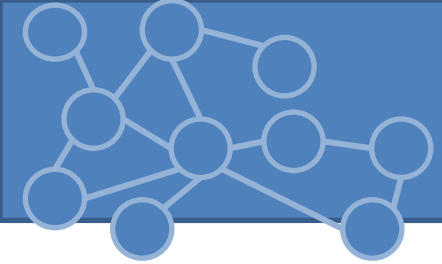
- Data che indica quando un documento dovrebbe essere rimosso dalla cache e richiesto al server. Il valore long indica il numero di millisecondi dopo la solita data

public long getLastModified()

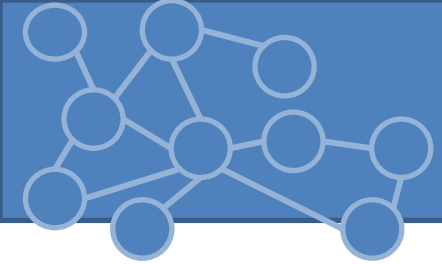
- Restituisce l'ultima data di modifica della risorsa



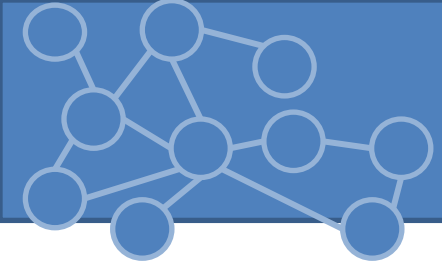
- **public String getHeaderField(String name)**
  - Restituisce il valore di un campo indicato dalla chiave name. Name non è case-sensitive e non include ‘:’
- **public String getHeaderFieldKey(int n)**
  - Restituisce la chiave dell’n-esimo campo. Il metodo di richiesta ha valore 0 e chiave nulla => Il primo campo utile è il numero 1
- **public String getHeaderField(int n)**
  - Restituisce il valore dell’n-esimo campo



- **int getResponseCode()**
  - Restituisce il codice di una risposta HTTP
- **String getResponseMessagge()**
  - Restituisce il messaggio di risposta HTTP eliminando il codice di risposta
    - HTTP/1.0 200 OK -> OK
    - HTTP/1.0 404 Not Found -> Not Found
  - *null* se non riesce ad estrarlo dalla risposta (il risultato non è un valido HTTP)



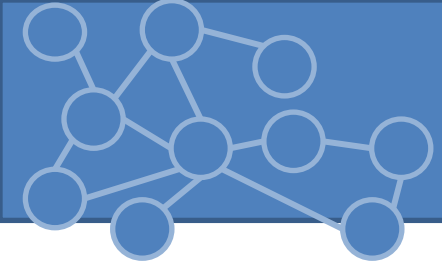
- **connected**
  - True se la connessione è aperta
  - Il suo valore iniziale (creazione HttpURLConnection) è false
  - ogni metodo che apre una connessione deve settare questo valore a true
  - Ogni metodo che chiude una connessione deve settare connected a false. Lo setta solo metodo disconnect()
- **allowUserInteraction**
  - Alcune HttpURL devono interagire con utente (username, password)
  - False per default
  - getAllowUserInteraction(), setAllowUserInteraction()
  - Modificabile solo prima della connessione



- **doInput**
  - HttpURL fornisce input ad un client (es: GET)
  - Settato a true se HttpURL può essere usata per l'input
  - Default: true
  - setDoInput(), getDoInput()
- **doOutput**
  - Necessità di fare output verso un server (es: POST)
  - Client può usare HttpURL per fare output verso il server cambiando il valore del campo a true
  - Quando viene posto a true il campo con un URL http, il metodo di richiesta è cambiato da GET a POST
  - Uso POST anziché GET se ho lunghi URL (vincoli browser o server)
- **ifModifiedSince**
  - Client mantengono in cache documenti reperiti in precedenza. Alcuni doc possono cambiare e cache non più consistente
  - Includo un campo If-Modified-Since nell'header, che include data e ora
  - Se documento è stato cambiato dopo il periodo specificato il server invia la risorsa altrimenti risponde con un messaggio 304
- **useCaches**
  - Indica se si può usare la cache. (classe ResponseCache)
  - Default: true

- Esigenze di scrittura su server (es: form o upload file)
  - OutputStream getOutputStream()
- Prima di invocare il metodo, devo chiamare setDoOutput(true) => il default è false

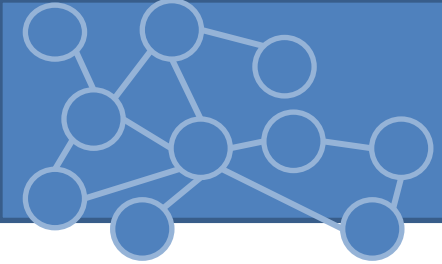
```
URL u = new URL("http://www.cafeaulait.org/books/jnp3/postquery.phtml");
HttpURLConnection huc = (HttpURLConnection) u.openConnection();
String richiesta = "username=Elliotte+Rusty+Harold&email=elharo%40metalab%2eunc%2eedu";
byte[] richiestaByte = richiesta.getBytes();

huc.setRequestMethod("POST");
huc.setRequestProperty("User-Agent", "CustomClient");
huc.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
huc.setRequestProperty("Content-Length", ""+richiestaByte.length);
huc.setDoOutput(true);
OutputStream os = huc.getOutputStream();
os.write(richiestaByte);
BufferedReader br = new BufferedReader(new InputStreamReader(huc.getInputStream()));
String risultati = "";
while((risultati = br.readLine())!=null){
    System.out.println(risultati);
```



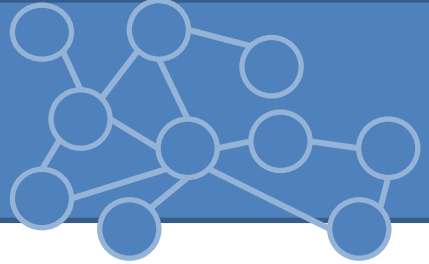
1. Decidere quali coppie chiave-valore
2. Creare una stringa query name1=value1&name2=value2&….
3. Aprire una HttpURLConnection verso il server che accetta i dati
4. Settare a true il campo doOutput
5. Scrivere la query nell'OutputStream
6. Leggere la risposta del server

```
URL u = new URL(args[0]);
HttpURLConnection huc = (HttpURLConnection) u.openConnection();
huc.setRequestMethod("GET");
int dimensione = Integer.parseInt(huc.getHeaderField("Content-length"));
InputStream is = huc.getInputStream();
int dimb = 1024;
byte[] buffer = new byte[dimb];
int r,da_leggere;
do{
    da_leggere = Math.min(dimensione,dimb-1);
    r = is.read(buffer, 0, da_leggere);
    System.out.println(new String(buffer, 0, r));
    dimensione-=r;
}while(dimensione > 0);
```

- Se non sono sicuri che Content-Length sia settato pongo dimensione = Integer.MAX_VALUE



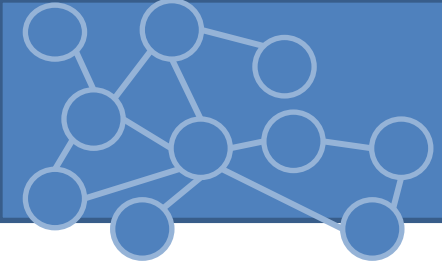
- HttpURL fortemente correlata ai meccanismi Java di protocol e content handler
- Protocol Handler: connessione, scambio headers, overhead del protocollo per la richiesta file
- Content Handler: gestisce i dati, input grezzo dopo gli header e conversione nel giusto oggetto
- **Object getContent()**
  - Chiama il metodo getContent di URLConnection.
  - VM deve riconoscere il tipo del contenuto
  - Funziona con protocolli come HTTP perché usa MIME, se content type sconosciuto UnknownServiceException
- Object getContent(Class[] classes)
  - Posso scegliere la classe del contenuto
  - Ordine della preferenza = ordine array



```
public class ArrayContenuto {

    public static void main(String[] args) throws IOException {
        URL u = new URL("http://www.ilsitodellamiavita.com");
        HttpURLConnection huc = (HttpURLConnection) u.openConnection();
        Class[] type = {String.class, Reader.class, InputStream.class};
        Object o = huc.getContent(type);
    }

}
```

Esercizi

1. Si stampino a video gli header HTTP di una risposta utilizzando prima i metodi delle slide 15-16-17e poi quelli della slide 18. Si confronti i risultati con quelli restituiti usando putty come client.
2. Si implementi il seguente modello. Il client C invia una richiesta HTTP con metodo di accesso HEAD ad un server S1, il quale inoltra la richiesta al server HTTP. S1 riceve la risposta e la rinvia al client cambiando il codice della risposta secondo una politica ch potete decidere voi
3. Utilizzare il metodo GET per ottenere il contenuto relativo ad un'immagine memorizzandolo su disco. Si verifichi che l'immagine sia stata salvata correttamente
4. Si analizzino i campi Date e Expiration date di un qualsiasi sito che fornisca un servizio con dati in tempo reale. Se il campo Expiration date è settato si richieda la pagina appena dopo la data della sua scadenza (qualcosa di molto simile ad un refresh automatico del sito)
5. Si utilizzi il metodo getContent(Class[]) per prelevare il contenuto della pagina del corso
6. Utilizzando il metodo GET si eseguano delle ricerche attraverso www.google.it e si estraggano tutti i link presenti nella pagina di risposta



7. Si implementi un ‘proto’-web server che riceve richieste HTTP solo di tipo GET e verifichi che la richiesta sia valida.
8. Si estenda il server precedente in modo tale che tenga delle statistiche sui tipi di client che hanno richiesto dei servizi.
9. Si utilizzi il metodo GET per eseguire ricerche su www.wikipedia.it. Si stampino il codice e il messaggio dell’intestazione della risposta.
10. Si stampino i messaggi completi di richiesta e risposta di un sito a vostra scelta e si ripeta la stessa richiesta utilizzando putty .
11. Si implementi una classe FormPoster la quale invia i dati di un form alla risorsa http://www.cafeaulait.org/books/jnp3/postquery.phtml. La pagina restituita contiene le coppie chiavi,valore impostati nella richiesta.