CSI 3540

Structures, techniques et normes du Web

Programmation côté serveur : Servlets

Objectif:

- Introduction à la programmation côté serveur
- Introduction aux **Servlets**

Lectures:

- Web Technologies (2007) § 6 Pages 323-350

Plan

1. Traitements parallèles
2. Autres méthodes du Servlet

Traitements parallèles

Traitements parallèles (//)

- **Côté client** : les fureteurs téléchargent plusieurs images simultanément ou encore plusieurs onglets (tab) téléchargent plusieurs pages simultanément

- **Côté serveur** : des centaines, voir même des milliers, d'usagers accèdent le serveur simultanément

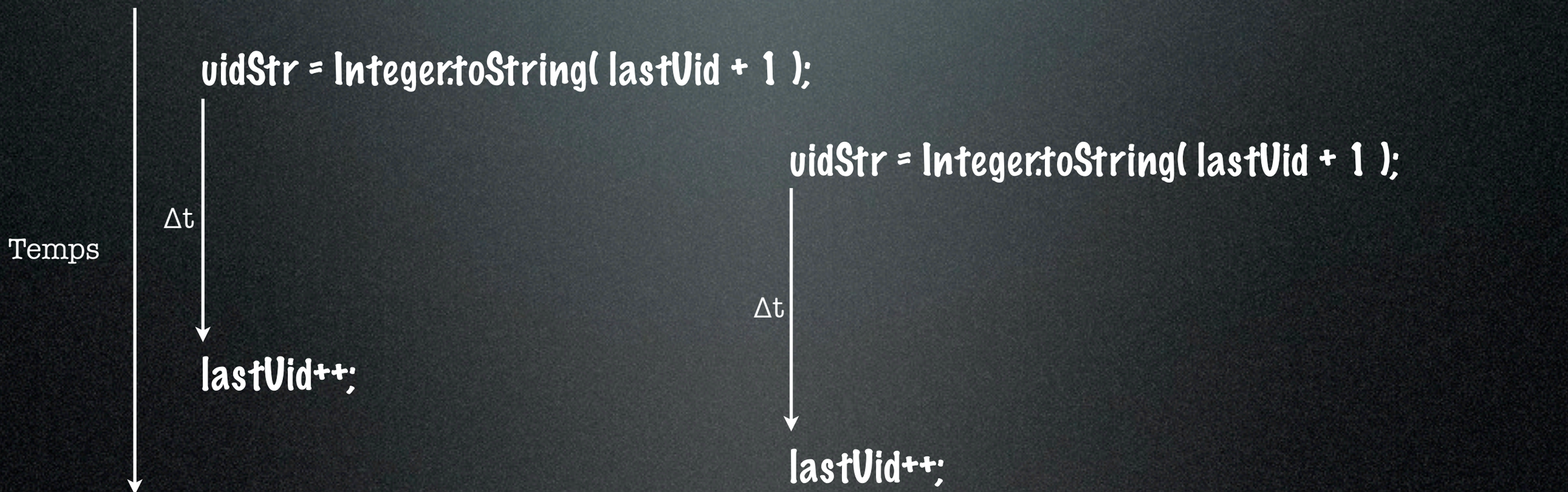
Traitements parallèles en Java

- Pour nos besoins, il n'y a qu'**une instance du Servlet**
- Mais chaque requête de l'usager est traitée par un **filin d'exécution** qui lui est propre. Ainsi, plusieurs filins exécutent **counter.doGet()** simultanément

```
public class SlowGetUid extends HttpServlet {

   private int lastUid = 1000;   // variable d'instance

   public void doGet( HttpServletRequest requete, HttpServletResponse reponse )
      throws ServletException, IOException {

      ...

⇨     String uidStr = Integer.toString( lastUid + 1 );

      try {
         Thread.sleep( 1000 * 5 ); // on simule on long traitement
      } catch ( InterruptedException e ) {
         System.err.println( "** InterruptedException caught **" );
      }

⇨     lastUid++;

      // génération du document

      doc.println( "<!DOCTYPE html" );
      ...
⇨     doc.println( "      Votre code d'acc&egrave;s est le : <b>" + uidStr + "</b>!" );
      ...
   }
}
```

Traitements parallèles

Traitements parallèles



Traitements parallèles en Java

- **Chaque filin** d'exécution possède sa propre **pile d'appels** (variables locales, paramètres, compteur de programme, etc.)

- Par contre, les **variables** d'**instanc**e et de **classe**, ainsi que les **ressources externes**, par exemple les fichiers, sont **partagées**

Traitements parallèles en Java

- **Accès en lecture**. Certains accès simultanés ne posent aucun problème. Les objets accédés en lecture seulement ou immuables. Par exemple, un dictionnaire de mots.

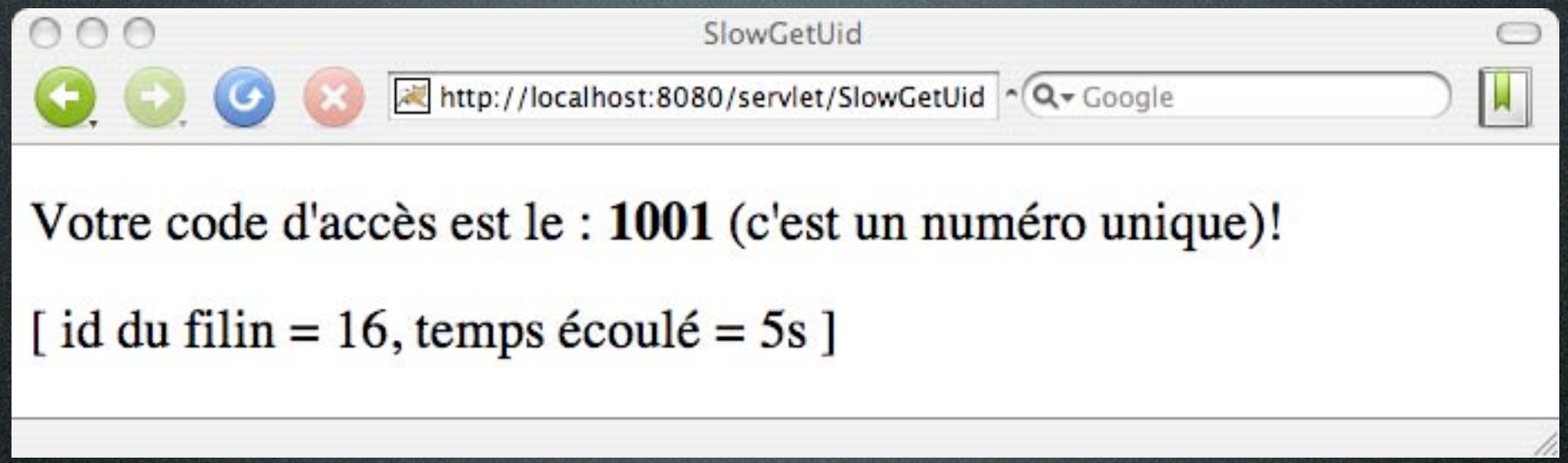
synchronized

- Méthode «**synchronized**»
- La JVM crée automiquement un **verrou** pour chaque objet ou classe
- Lorsqu'un filin exécute une méthode «**synchronized**» il détient le verrou de l'objet ou de la classe
- Aucun autre filin ne peut exécuter une méthode de cet objet ou de cette classe

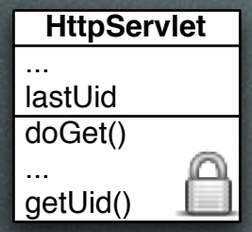
synchronized

```
private synchronized String getUid() {

  String uidStr = Integer.toString( lastUid + 1 );

  try {
     Thread.sleep( 1000 * 5 );
  } catch ( InterruptedException e ) {
     System.err.println( "** InterruptedException caught **" );
  }

  lastUid++;

  return uidStr;
}
```

synchronized



synchronized



Filin 1

Filin 2

getUid()

Δt

Temps

getUid()

Δt

Attention!

- Imaginez deux servlets faisant la mise à jour du nombre de pièces en inventaire
- L'un s'occupe des achats
- L'autre s'occupe des ventes
- Le nombre de pièces est partagé et sauvegardé dans un fichier
- **Comment éviter les problèmes de synchronisation (entre servlets)?**

Solution

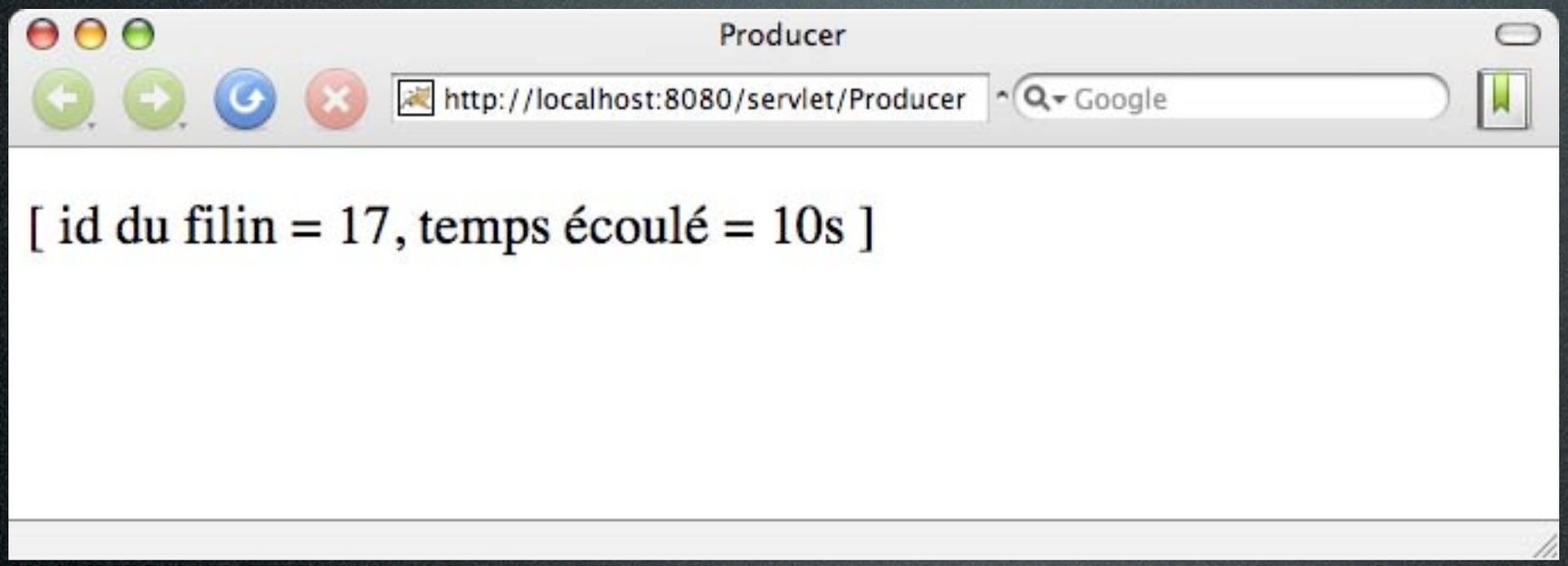
- Les traitements liés aux entrées-sorties sont placés dans une classe à part (**InventoryIO**)
- Les méthodes sont des méthodes de classe et **synchronized**
- Les servlets utilisent les méthodes de la classe **InventoryIO** afin de mettre à jour l'inventaire

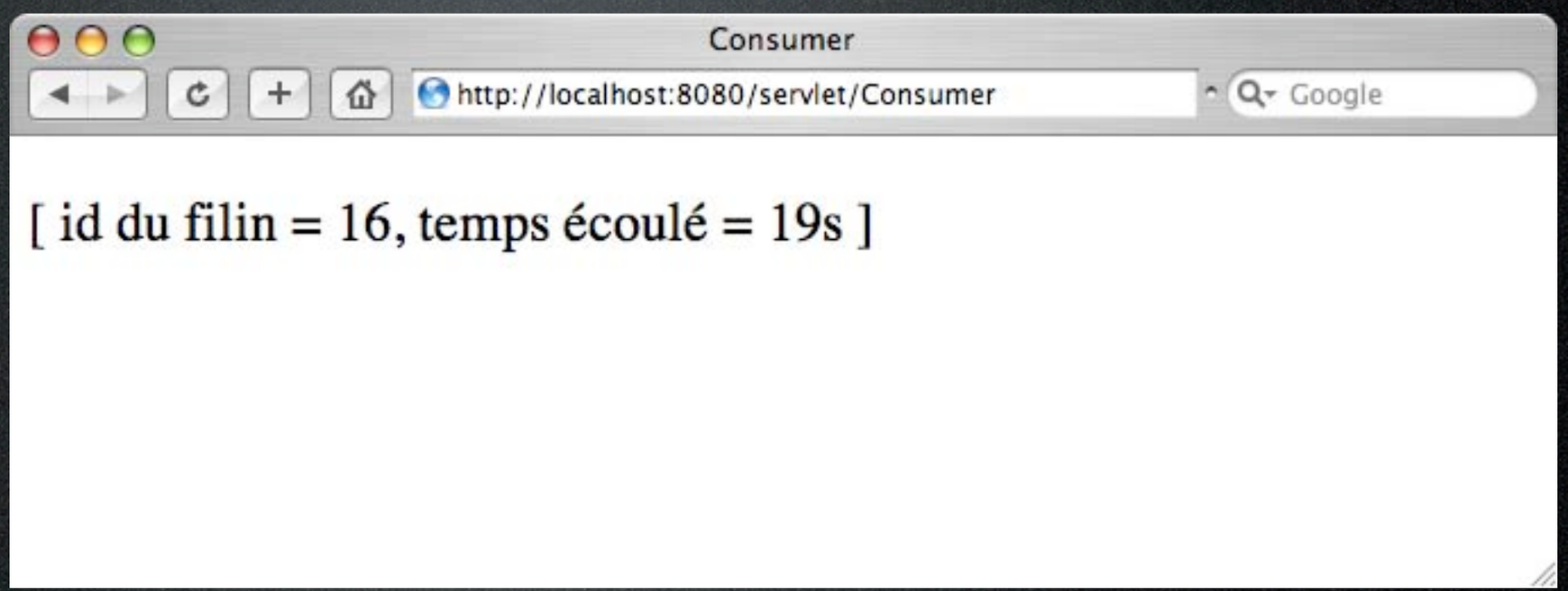
```
public class Utils {

    public static synchronized void read() {
        try {
            Thread.sleep( 1000 * 10 );
        } catch ( InterruptedException e ) {
            System.err.println( "** InterruptedException caught **" );
        }
    }

    public static synchronized void write() {
        try {
            Thread.sleep( 1000 * 10 );
        } catch ( InterruptedException e ) {
            System.err.println( "** InterruptedException caught **" );
        }
    }

}
```

```
public class Producer extends HttpServlet {
    public void doGet( ... ) {
        ...
        Utils.write();
        ...
    }
}
```

```
public class Consumer extends HttpServlet {
    public void doGet( ... ) {
        ...
        Utils.read();
        ...
    }
}
```

synchronized – 2 servlets





Solution (alternative)

- Les traitements liés aux entrées-sorties sont placés dans une classe à part (**InventoryIO**)
- Utilisez le modèle de conception (« design pattern ») **singleton**
- La classe **InventoryIO** retourne toujours le même objet pour un même nom de fichier et les Servlets utilisent les méthodes de cet objet

Attention! (suite)

- S'il y a plusieurs verrous, les filins peuvent se retrouver en situation d'interblocage (**deadlock**)
- Voir «Systèmes d'exploitation» (CSI 3531) ou encore «Bases de données II» (CSI 3530)

Remarques

Remarques

- Un Servlet en mode production devrait traiter proprement les **exceptions**
- **ServletException** : cas général
- **UnavailableException** : le Servlet n'est pas disponible, temporairement ou de façon permanente (isPermanent())

Remarques

```
public void init()
    throws ServletException
{
    try {
        BufferedReader br =
            new BufferedReader(new FileReader("aFile"));
        visits = (new Integer(br.readLine())).intValue();
    }
    catch (FileNotFoundException fnfe) {
        throw new UnavailableException("File not found: " +
                                       fnfe.toString());
    }
    catch (Exception e) {
        throw new UnavailableException("Data problem: " +
                                       e.toString());
    }
}
```

Remarques

- L'élément (balise) **INPUT** de type **file** n'est pas supporté à haut niveau par l'API des Servlets (Jackson pages 320)
- Il existe un certains nombre de bibliothèques, dont celle-ci :

  http://commons.apache.org/fileupload

Remarques

- Nous avons fait l'hypothèse que le serveur exécute **une seule instance** d'un Servlet (ce n'est pas nécessairement le cas pour une application réelle)
- Ne pas appeler la méthode **getWriter** avant un appel à **setContentType**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Date;

public class GetCount extends HttpServlet {

    private int count = 0;

    public void doGet( HttpServletRequest requete, HttpServletResponse reponse )
        throws ServletException, IOException {

        reponse.setContentType( "text/html; charset=\"UTF-8\"" );
        PrintWriter doc = reponse.getWriter();

        count++;

        doc.println( "<!DOCTYPE html" );
        // ...
        doc.println( "      <b>" + count + "</b>" );
        // ...
        doc.println( "</html>" );

        doc.close();
    }
}
```

Remarques

- Plusieurs éléments **checkbox** ont souvent le même nom (attribut **name**)
- Ainsi, la chaîne requête contiendra plusieurs paires **attribut/valeur** ayant le même nom d'attribut
- Il faudra donc utiliser la méthode **requete.getParameterValues(name);**

```
<form action="http://localhost:8080/QueryString/get" method="get">
  <table border="0" cellpadding="5">
    <tr>
      <td>Nom :</td>
      <td><input type="text" size="30" name="nom" /></td>
    </tr>
    ...
    <tr>
      <td>Couleur :</td>
      <td>
        <label>
          <input type="checkbox" name="BoiteCouleur" value="bleu">Bleu
        </label>

        <label>
          <input type="checkbox" name="BoiteCouleur" value="blanc">Blanc
        </label>

        <label>
          <input type="checkbox" name="BoiteCouleur" value="rouge">Rouge
        </label>
      </td>
    </tr>
    <tr>
      <td><input type="submit" value="Soumettre" /></td>
      <td></td>
    </tr>
  </table>
</form>
```

CSI3540. Structures, techniques et normes du Web

index.html

Google

Nom :

Code postal :

Pays : Sélectionnez un pays :

Couleur : Bleu Blanc Rouge

Soumettre

GetParameterNames

http://localhost:8080/QueryString/get?nom=&code=&pays=none&BoiteCouleur=blanc&BoiteCouleur=rouge

Google

nom=&code=&pays=none&BoiteCouleur=blanc&BoiteCouleur=rouge

nom = []

code = []

pays = [none]

BoiteCouleur = [blanc][rouge]

« Cross-site scripting (XSS) attack »

- Lorsqu'on crée une page Web à partir d'informations transmises par l'usager, il faut traiter les données de l'usager afin de retirer les éléments potentiellement dangereux (JavaScript, redirection...)

  www.cert.org/advisories/CA-2000-02.html
  www.ibm.com/developerworks/tivoli/library/s-csscript/

- Changer tous les **<** par **<** **&** par **&** ...

Utils.escapeXML(...)

```
public class Utils {
    static private Pattern pAmp = Pattern.compile( "&" );
    static private Pattern pLT =  Pattern.compile( "<" );
    static private Pattern pGT =  Pattern.compile( ">" );

   /**
    * Return input string with ampersands (&),
    * less-than signs (<), and greater-than signs (>)
    * replaced with character entity references.
    */

   static public String escapeXML( String inString ) {
      Matcher matcher = pAmp.matcher( inString );
      String modified = matcher.replaceAll( "&amp;" );
      matcher = pLT.matcher( modified );
      modified = matcher.replaceAll( "&lt;" );
      matcher = pGT.matcher( modified );
      modified = matcher.replaceAll( "&gt;" );
      return modified;
   }
}
```

Structures, techniques et normes du Web (CSI 3540) : Laboratoire 2

http://www.site.uottawa.ca/~turcotte/teaching/csi-3540/lectures/lab-02/index.html

Google

uOttawa Informatics Bioinformatics Biotech Sun Academic Web Folio Timeout Google Networks References Maison

- Étant donné le DTD suivant:

```
<!ELEMENT tableau (entete?, (ligne+), propr?)>
<!ELEMENT entete (#PCDATA)>
<!ELEMENT ligne (#PCDATA)>
<!ELEMENT propr EMPTY>
<!ATTLIST propr num ID #REQUIRED>
```

Pour chacun des exemples qui sui
syntaxiquement correct, et donn
l'erreur s'il y lieu, en vous référ

```
<tableau>
  <ligne>Un</ligne>
  <ligne>Deux</ligne>
  <ligne>Trois</ligne>
</tableau>
```

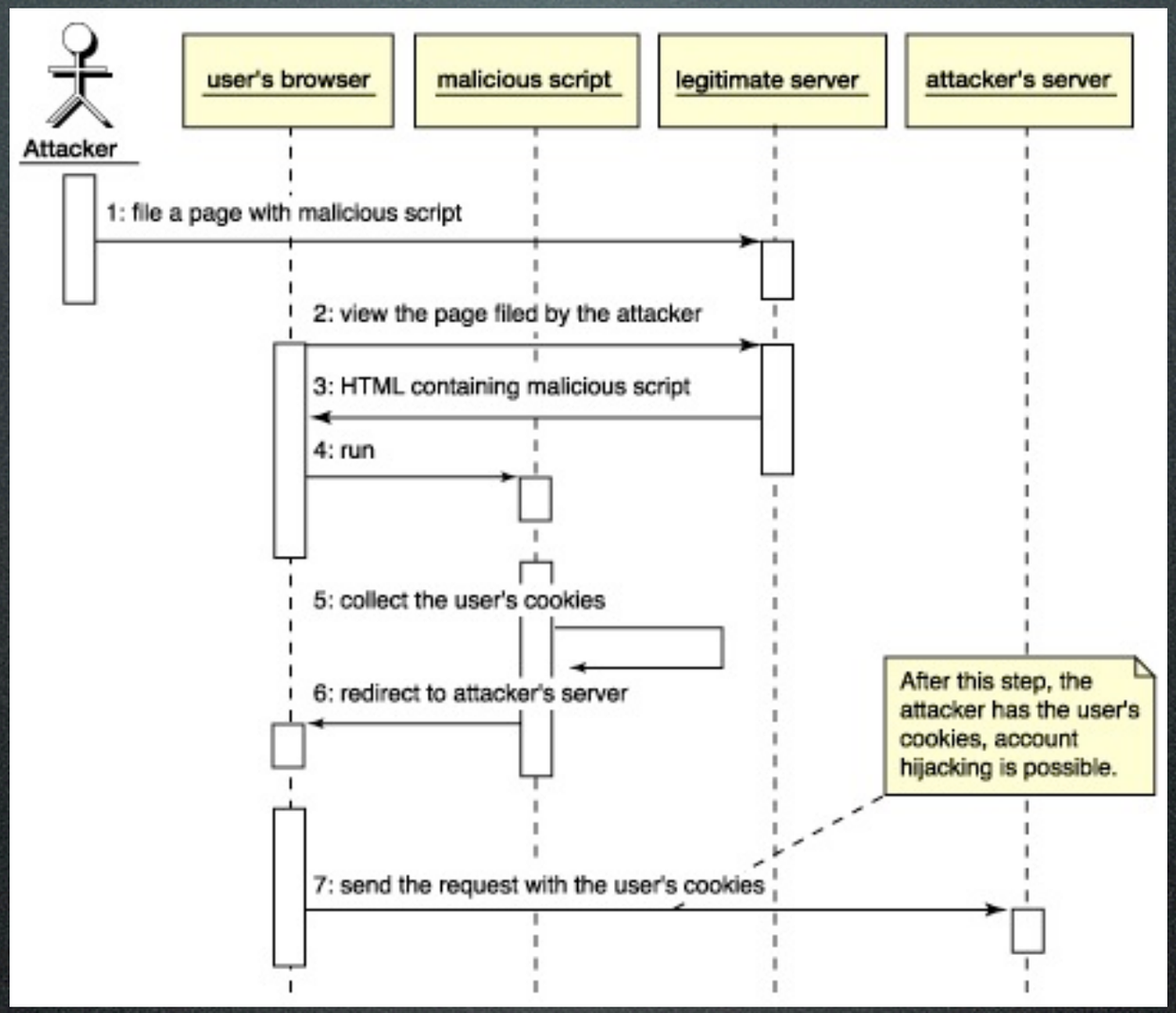
Source of http://www.site.uottawa.ca/~turcotte/teaching/csi-3540/lectures/lab-02/index.html

```
- &Eacute;tant donn&eacute; le DTD suivant:

  &lt;!ELEMENT tableau (entete?, (ligne+), propr?)&gt;
  &lt;!ELEMENT entete (#PCDATA)&gt;
  &lt;!ELEMENT ligne (#PCDATA)&gt;
  &lt;!ELEMENT propr EMPTY&gt;
  &lt;!ATTLIST propr num ID #REQUIRED&gt;

  Pour chacun des exemples qui suivent, dites si le code XML est
  syntaxiquement correct, et donnez une br&egrave;ve description de
  l'erreur s'il y lieu, en vous r&eacute;f&eacute;rant au DTD.

  &lt;tableau&gt;
    &lt;ligne&gt;Un&lt;/ligne&gt;
    &lt;ligne&gt;Deux&lt;/ligne&gt;
    &lt;ligne&gt;Trois&lt;/ligne&gt;
  &lt;/tableau&gt;

  &lt;tableau&gt;
    &lt;ligne&gt;Un&lt;/ligne&gt;
    &lt;entete&gt;Nombres&lt;/entete&gt;
  &lt;/tableau&gt;

  &lt;tableau&gt;
    &lt;entete&gt;Nombres&lt;/entete&gt;
  &lt;/tableau&gt;

  &lt;tableau&gt;
    &lt;entete&gt;Nombres&lt;/entete&gt;
    &lt;ligne&gt;Un&lt;/ligne&gt;
    &lt;propr num="1"&gt;&lt;/propr&gt;
  &lt;/tableau&gt;
```

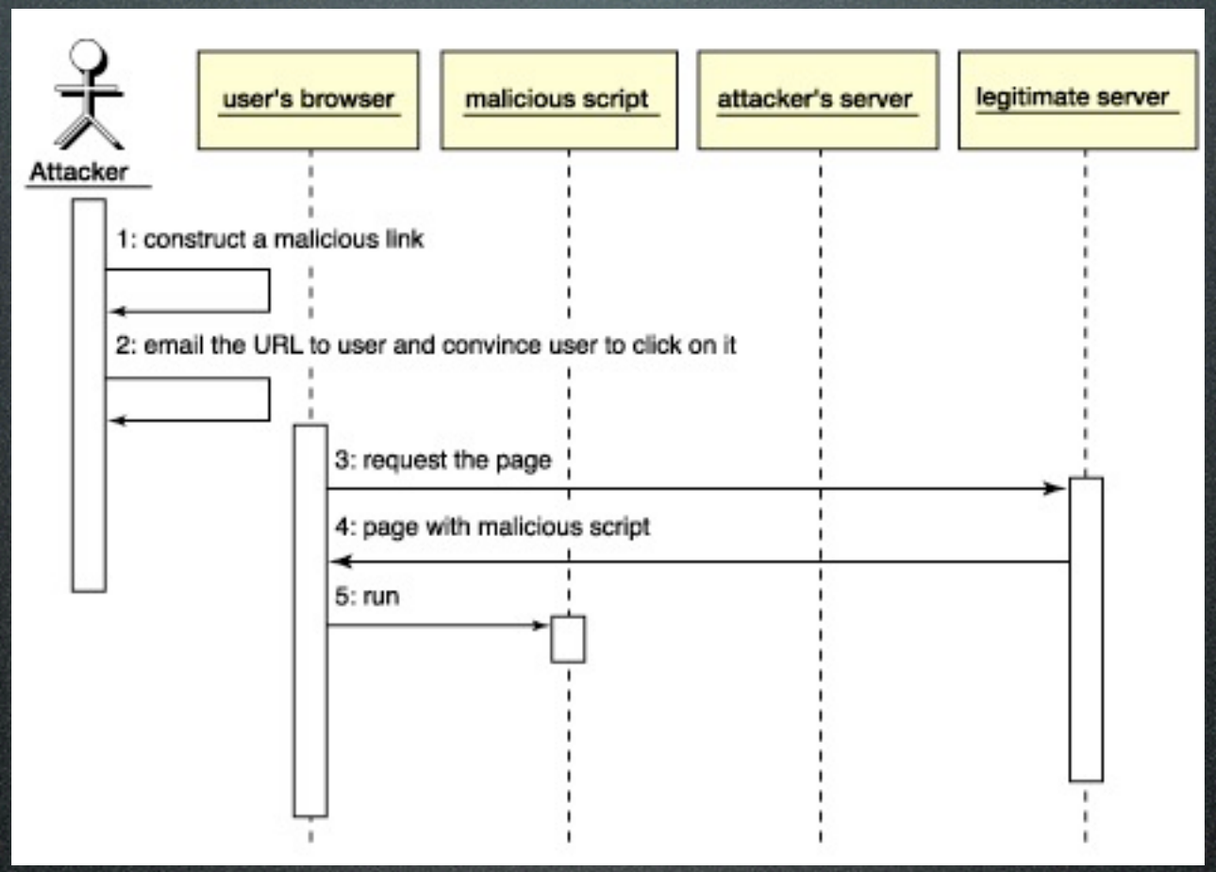
XSS : Comment?

- Je développe une application Web de type blogue
- Les usagers du blogue soumettent (publient) leurs commentaires
- Un usager malicieux inclut **<script>...</script>** dans son commentaire
- Les usagers subséquents téléchargent la nouvelle page et exécutent le script

- Afin de camoufler ses traces
  - <a href="http://legitimateSite.com/registration.cgi?clientprofile=<script>malicious code</script>">Click here
- L'usager malicieux encodera les données à l'aide d'un encodage hexadécimal
  - Click here



http://www.ibm.com/developerworks/tivoli/library/s-csscript/



http://www.ibm.com/developerworks/tivoli/library/s-csscript/

XSS : Pourquoi?

- Exécution de code JavaScript
- Prendre le contrôle de la **session** de l'usager
- **Détourner** le client vers le serveur de l'attaquant
- Vol de **témoins de session**...

CSI3530
Bases de données II

- Conception avancée de bases de données physiques. Droits d accès, protection et **sécurité**. Traitement et optimisation des requêtes. Traitement des transactions, contrôle du parallélisme et reprise. Bases de données orientées objets, base de données distribuées et multi-bases de données. Entrepôts de données. Intégration des données. Conception et implantation d un composant de base de données (projet de groupe).

CSI4539

Conception de systèmes informatiques sécuritaires

- Politiques de **sécurité**. Mécanismes de **sécurité**. **Sécurité** physique. Conscience de la **sécurité**. Authentification d'utilisateur. Application des mécanismes de **sécurité**. Codage. "Firewalls" internes et externes. **Sécurité** des systèmes d'opération et des logiciels. **Sécurité** des applications de commerce électronique. Design de systèmes et composantes de **sécurité**. Dispositifs pour l'analyse de la **sécurité**, renifleurs, détecteurs d'attaque. Guerre de l'information. Aspects éthiques de la **sécurité** informatique.

Épilogue

Résumé

- Redéfinir **doGet**() et/ou **doPost**()
- Accès aux paramètres : **getQueryString**(), **getParamerterValue**(name)
- **getSession**(), **getCookies**()

Servlet (suite)

- **HttpServletRequest** : getRemoteAddr(), getRemoteHost(), getProtocol(), isSecure(), getRequestURL(), getHeaderNames(), getHeader(champ)

- **HttpServletResponse** : setHeader(nom, valeur), setDateHeader(nom, valeur), setContentLength(taille), etc.

- **HttpServlet** : doDelete(), doOptions(), doPut(), doTrace(), doHead()

Limites

- Que ce soit les Servlets ou les applications CGI
- **La logique des programmes et la structure des pages générées sont entremêlées**
- Ces deux aspects d'une application Web sont, en général, réalisés par différents spécialistes
- **Solution** : Les pages JSPX

XML (suite)

- **XML** sera derrière toutes les applications à venir :
  - Les pages JSPX
  - Services : WDSL, SOAP, ...
  - Transformations : XSL, XSLT
  - Ajax, ...

Ressources

- SR-000154 JavaTM Servlet 2.4 Specification [http://jcp.org/aboutJava/communityprocess/final/jsr154/index.html] 2007
- Java Servlet Technology [http://java.sun.com/products/servlet] 2008-02-26