Глава 7, URL, наставак презентације

7.2 Класе URLEncoder и URLDecoder

Карактери у URL-овима могу бити:

- велика и мала слова енглеске абецеде A-Z, a-z
- цифре 0-9
- знаци интерпункције - _ . ! ~ * ' ( )

Карактери : / & ? @ # ; $ + = % се такође могу користити, али само у специјалне сврхе. Ако је неки од њих део имена фајла, мора се кодирати.

Кодирање је веома једноставно:

- Карактер се кодира бајтом, и сваки бајт се пише као % за којим следе две хексадекадне цифре.
- Белина је специјални случај јер је веома честа њена употреба. Осим што се кодира са %20, може се кодирати и као +.
- Сам + се кодира са %2B.
- Карактере / # = & ? треба кодирати када су део имена, а не када су сепаратори између делова URL-а.

Класа URL не врши аутоматски кодирање и декодирање. Могуће је конструисати URL објекте који користе недопуштене ASCII и не- ASCII карактере и %-секвенце. Такви карактери и секвенце се не кодирају и декодирају аутоматски приликом исписа помоћу метода getPath() и toExternalForm(). Програмер је одговоран да сви карактери буду исправно кодирани у стринговима који се користе за креирање URL-а.

На срећу, у Јави постоји класа URLEncoder која кодира стрингове у овај формат. Постоји и класа URLDecoder, која декодира стрингове из овог формата. Није могуће креирати објекте ових класа.

Класа **URLEncoder** садржи само један статички метод:

```
public static String encode(String s, String encoding)  throws
                                                    UnsupportedEncodingException
```

Метод конвертује све не-алфанумеричке карактере у %-секвенце (осим бланка, подвлаке, цртице, тачке и звездице). Кодира и све не- ASCII карактере. Бланко се конвертује у +. Метод је мало преагресиван: конвертује и тилду, једноструки наводник, знак узвика и заграде у %-секвенце, иако то апсолутно није потребно. Међутим, то није забрањено URL спецификацијом, тако да се browser-и успешно носе са тим.
Метод враћа кодирани стринг.
За кодирање увек и искључиво треба задавати UTF-8. UTF-8 је компатибилан са много више software-а него иједно друго кодирање (са IRI спецификацијом, класом URI, web browser-има, ...).

Овај метод кодира /, &, = и : не покушавајући да утврди у ком својству су ти карактери употребљени у URL-у. Због тога је неопходно кодирати URL део по део уместо читав само једним позивом метода. Ово је битно јер је најчешћа употреба класе URLEncoder припрема упита за комуникацију са server-side програмима који користе GET.

Пример 5: класа QueryString која користи URLEncoder како би кодирала узастопне парове име-вредност у Јава објекат који ће се користити приликом слања података server-side програмима. Приликом креирања објекта класе QueryString, први пар име-вредност може се задати конструктору, као два одвојена стринга. За додавање осталих парова, позива се метод add(), који такође прима два стринга као аргументе, и кодира их. Метод getQuery() враћа резултујући стринг у коме су парови име-вредност кодирани.

Користећи ову класу, упит:
```
pg=q&kl=XX&stype=stext&q=+"Java I/O"&search.x=38&search.y=3
```
се може кодирати са:
```
QueryString qs = new QueryString("pg", "q");
qs.add("kl", "XX");
qs.add("stype", "stext");
qs.add("q", "+\"Java I/O\"");
qs.add("search.x", "38");
qs.add("search.y", "3");
String url = "http://www.altavista.com/cgi-bin/query?" + qs;
System.out.println(url);
```

Класа **URLDecoder** поседује два статичка метода који декодирају стрингове кодиране у x-www-form-url-encoded формату:
```
public static String decode(String s) throws Exception
public static String decode(String s, String encoding) throws
                                                    UnsupportedEncodingException
```

За кодирање задавати UTF-8 јер је највероватније да ће бити коректно више него било које друго.

Може бити избачен изузетак типа IllegalArgumentException у случају да постоји % који није праћен двема хексадекадним цифрама или се декодира у недопуштену секвенцу.

Овом методу може се проследити читав URL, не мора се прво раставити на делове.
Нпр.
```
String input = "http://www.altavista.com/cgi-bin/" +
"query?pg=q&kl=XX&stype=stext&q=%2B%22Java+I%2FO%22&search.x=38&search.y=3";
 try {
  String output = URLDecoder.decode(input, "UTF-8");
  System.out.println(output);
 }
```

7.4 Прокси сервери

Прокси сервер прима од локалног клијента захтев за удаљени сервер. Прокси поставља захтев удаљеном серверу и прослеђује одговор локалном клијенту. Понекад се то чини из безбедносних разлога, како би се спречило да удаљени хостови сазнају детаље конфигурације локалне мреже. У неким ситуацијама се то ради како би се корисници спречили да приступају забрањеним сајтовима филтерисањем излазних захтева и ограничавањем који сајтови могу бити посећени. А некад се ради само због перформанси, како би се омогућило да већи број корисника добија исте популарне документе из локалног кеша уместо да понављају преузимања са удаљеног сервера.

Јава програми базирани на класи URL могу радити са већином уобичајених проксија и протокола.
System Properties
Потребно је поставити само неколико њих како би се указало на адресе наших локалних прокси сервера.
Ако се користи само HTTP прокси, постави се http.proxyHost на име или IP адресу прокси сервера и http.proxyPort на порт прокси сервера (подразумевани је 80). Постоји неколико начина да се то уради, укључујући позиве System.setProperty() у Јава коду или задавање -D опција приликом покретања програма.

Уколико желимо да се за неки хост не користи прокси и да конекција буде директна, постави се system propery http.nonProxyHosts на његов hostname или IP адресу. Да би се то урадило за већи број хостова, њихова имена се раздвоје усправним цртама. Нпр.
следећи фрагмент кода поставља прокси за све осим за java.oreilly.com и xml.oreilly.com:

```
System.setProperty("http.proxyHost", "192.168.254.254");
System.setProperty("http.proxyPort", "9000");
System.setProperty("http.nonProxyHosts", "java.oreilly.com|xml.oreilly.com");
```

Може се користити и * као wildcard како би се реферисали сви хостови из одређеног домена или поддомена.

...

7.5 Communicating with Server-Side Programs Through GET

Класа URL омогућује једноставну комуникацију Јава аплета и апликација са server-side програмима попут CGI-ova, сервлета, PHP страна, и осталог што користи метод GET. Све што је потребно да знамо је коју комбинацију имена и вредности програм очекује да добије и да саставимо URL са упитом који обезбеђује потребна имена и вредности. Сва имена и вредности морају бити x-www-form-url-encoded.

Постоји већи број начина за утврђивање тачне синтаксе упита за одређени програм. Ако смо сами написали server-side програм, онда већ знамо које парове име-вредност он очекује. Ако смо на свој сервер инсталирали неки програм, у његовој документацији би требало да пише шта он очекује.

Ако комуницирамо са програмом на серверу који није наш, ствари су мало проблематичније. Увек је могуће питати људе који на њему раде да нам дају спецификацију за обраћање њиховом сајту. Међутим, питање је да ли ће то уродити плодом, јер то не спада у опис њиховог посла. Зато је вероватно да је потребно мало reverse engineering-а.

Многи програми су дизајнирани тако да процесирају уносе форми. У таквом случају, праволинијски се одређује који је очекивани улаз програма. Метод који форма користи, требало би да буде вредност атрибута METHOD елемента FORM. Ова вредност би требало да буде GET или POST. Део URL-а који претходи упиту задат је вредношћу атрибута ACTION елемента FORM. То може бити релативни URL и у том случају је неопходно одредити и одговарајући апсолутни. Коначно, парови име-вредност су једноставно атрибути NAME елемената INPUT, осим за INPUT елементе чији атрибут TYPE има вредност submit.

Нпр. размотримо следећу HTML форму:

```
<form name="search" action="http://www.google.com/search" method="get">

  <input name="q" />

  <input type="hidden" value="cafeconleche.org" name="domains" />

  <input type="hidden" name="sitesearch" value="cafeconleche.org" />

  <input type="hidden" name="sitesearch2" value="cafeconleche.org" />

   <br />

   <input type="image" height="22" width="55"

      src="images/search_blue.gif" alt="search" border="0"

      name="search-image" />

</form>
```

Може се видети да она користи метод GET. Програму који процесира форму приступа се преко URL-а http://www.google.com/search. Постоје 4 пара име-вредност, од којих 3 имају подразумеване вредности.

Тип поља INPUT није од значаја. Није битно да ли се ради о скупу чек-боксова, падајућој листи или текстуалном пољу, битни су само име поља INPUT и одговарајућа вредност. Једини изузетак је submit, који каже browser-у када да пошаље податке али не даје серверу никакву информацију. У неким случајевима јављају се скривена INPUT поља која морају имати одређену подразумевану вредност. Форма из примера има 3 таква поља.

Програм који не обрађује форму је много тежи за reverse engineering. Нпр. на http://www.ibiblio.org/nywc/bios.phtml постоји мноштво линкова на PHP стране које контактирају базу како би добиле листу дела одређеног композитора. Међутим, не постоји никаква форма која одговара овом програму. Све је урађено помоћу URL-ова. У овом случају, најбоље је прегледати што је могуће више тих URL-ова и видети да ли је могуће погодити шта сервер очекује. Уколико дизајнер није превише околишао, то не би требало да буде претешко. Нпр.

```
http://www.ibiblio.org/nywc/compositionsbycomposer.phtml?last=Anderson
    &first=Beth&middle=
http://www.ibiblio.org/nywc/compositionsbycomposer.phtml?last=Austin
    &first=Dorothea&middle=
http://www.ibiblio.org/nywc/compositionsbycomposer.phtml?last=Bliss
    &first=Marilyn&middle=
http://www.ibiblio.org/nywc/compositionsbycomposer.phtml?last=Hart
    &first=Jane&middle=Smith
```

из ових URL-ова може се претпоставити да програм очекује 3 улаза: first, middle и last, са вредностима имена, средњег слова и презимена композитора, редом.

Понекад улази можда немају тако очигледна имена. Тада је неопходно мало експериментисати, покушавајући најпре са неким постојећим вредностима, а затим њиховим подешавањем видети које вредности јесу, а које нису прихваћене. То није потребно радити у Јава програму. Може се просто едитовати URL у адресној линији web browser-а.

Вероватноћа да ће други хакери експериментисати са Вашим server-side програмима на исти начин је добар разлог да их учините екстремно робусним за неочекивани улаз.

Без обзира на то како сте одредили скуп парова име-вредност који сервер очекује, једном када су они познати, комуникација је једноставна. Све што је потребно урадити јесте да се оформи упит који укључује неопходне парове име-вредност, а затим и URL који га укључује. Упит се шаље серверу, а затим чита одговор од њега коришћењем истих метода који се користе за конектовање на сервер и добијање статичке HTML стране. Не постоји неки специјални протокол који треба пратити након што је конструисан URL.

Пример 6 (DMoz):
Једноставан програм за претраживање тема у Netscape Open Directory (http://dmoz.org/). Постоји једноставна форма са једним пољем за унос чије је име search. Унос из овог поља шаље се CGI програму на http://search.dmoz.org/cgi-bin/search, који врши стварну претрагу. HTML форма изгледа овако:

```
<form accept-charset="UTF-8"
      action="http://search.dmoz.org/cgi-bin/search" method="GET">
          <input size=30 name=search>
<input type=submit value="Search">
<a href="http://search.dmoz.org/cgi-bin/search?a.x=0">
<small><i>advanced</i></small></a>
</form>
```

Да би се захтев за претрагом послао Open Directory, треба само узети стринг који је унео корисник, кодирати га у упит и послати на `http://search.dmoz.org/cgi-bin/search`. Да би се тражила „јава“, потребно је отворити конекцију са URL-ом `http://search.dmoz.org/cgi-bin/search?search=java` и прочитати резултујући улазни ток.

Код за комуницирање са сервером је веома једноставан. Комуникација са server-side програмом који користи GET није ништа компликованија од добијања произвољне HTML стране.

7.6 Accessing Password-Protected Sites

Многи популарни сајтови захтевају корисничко име и шифру да би допустили приступ. Неки то имплементирају коректно, кроз HTTP аутентификацију. Други, пак, раде то некоректно, користећи cookies и HTML форме.

Јавина класа URL може приступати сајтовима који користе HTTP аутентификацију, иако јој, наравно, треба рећи које корисничко име и шифру да користи. Јава не нуди подршку за сајтове који користе нестандардну cookie-based аутентификацију.

Класа Authenticator

```
public abstract class Authenticator extends Object
```

Класа служи да се корисничко име и шифра обезбеде сајтовима који се штите коришћењем HTTP аутентификације.

Како је класа апстрактна, из ње се мора извести потклаcа. Различите потклаcе могу добијати информације на различите начине. Нпр. корисник може укуцати корисничко име и шифру на стандардном улазу или их робот може читати из криптованог фајла.

Да би класа URL користила ту потклаcу, потребно ју је инсталирати као подразумевани аутентификатор, тако што се проследи статичком методу `Authenticator.setDefault()`:

```
public static void setDefault(Authenticator a)
```

Нпр. ако смо написали класу `DialogAuthenticator` изведену из Authenticator, инсталирамо је са:

```
Authenticator.setDefault(new DialogAuthenticator( ));
```

То се ради само једном. Од тог места па надаље, када су класи URL потребни корисничко име и шифра, она их тражи статичким методом `Authenticator.requestPasswordAuthentication()`:

```
public static PasswordAuthentication requestPasswordAuthentication(
  InetAddress address, int port, String protocol, String prompt, String
scheme) throws SecurityException
```

Аргумент `address` је хост за који се тражи аутентификација, `port` је порт на том хосту, `protocol` је протокол апликацијског слоја којим се приступа сајту. HTTP сервер обезбеђује prompt. То је типично име домена за који се тражи аутентификација. (Велики web сервери, попут www.ibiblio.org имају више домена, од којих сваки захтева другачија корисничка имена и шифре.) `scheme` је аутентификациона схема која се користи (овде то није синоним за протокол).

Аплетима није дозвољено да траже корисничко име и шифру. Да би могли, неопходно је да имају `requestPasswordAuthentication NetPermission`. Иначе, метод избацује изузетак типа `SecurityException`.

Потклаcа класе `Authenticator` мора предефинисати метод `getPasswordAuthentication( )`. Унутар овог метода од корисника или неког другог извора се добијају корисничко име и шифра и враћају се као инстанца класе `java.net.PasswordAuthentication`. Прототип метода је:

```
protected PasswordAuthentication getPasswordAuthentication( )
```

Ако не желите да извршите аутентификацију захтева, вратите `null`, и Јава ће рећи серверу да не зна како да изврши аутентификацију конекције. Уколико се пошаљу неисправно име и шифра, Јава ће позвати поново метод `getPasswordAuthentication( )` како би Вам дала нову шансу да

унесете исправне податке. Уобичајено, имате 5 покушаја, а након тога `openStream( )` избацује `ProtocolException`.

Корисничка имена и шифре се кеширају на нивоу сесије виртуалне машине. Једном када сте за неки домен унели исправно корисничко име и шифру, неће Вам бити тражени поново, осим ако сте експлицитно обрисали шифру анулирајући низ карактера који је садржи.

Више детаља о захтеву може се добити позивима неких од следећих метода, наслеђених од суперкласе `Authenticator`:

```
protected final InetAddress getRequestingSite( )
protected final int         getRequestingPort( )
protected final String      getRequestingProtocol( )
protected final String      getRequestingPrompt( )
protected final String      getRequestingScheme( )
protected final String      getRequestingHost( )
```

Ови методи или враћају информацију прослеђену последњем позиву метода `requestPasswordAuthentication( )` или `null`, уколико информација није доступна. У случају да није доступна информација о порту, одговарајући метод враћа -1.

Постоје још два метода:

```
protected final String getRequestingURL( )
protected Authenticator.RequestorType getRequestorType( )
```

Први враћа потпуни URL за који је тражена аутентификација - што је значајно ако сајт користи различита корисничка имена и шифре за различите фајлове.
Други враћа једну од две именоване константе: `Authenticator.RequestorType.PROXY` или `Authenticator.RequestorType.SERVER` како би указао да ли аутентификацију тражи сервер или прокси сервер.

Класа `PasswordAuthentication`

У питању је веома једноставна `final` класа која има два read-only атрибута: корисничко име и шифру. Корисничко име је `String`. Шифра је `char[]`, тако да ју је могуће обрисати када више није потребна. `String` мора да чека да га покупи garbage collector пре него што буде обрисан, а чак и тада може постојати још негде у меморији локалног система, можда чак на диску, ако је блок меморије који га садржи отишао у виртуалну меморију у неком тренутку. И корисничко име и шифра се задају као аргументи конструктора:

```
public PasswordAuthentication(String userName, char[] password)
```

Може им се приступити позивом одговарајућих `get*()` метода:

```
public String getUserName( )
public char[] getPassword( )
```

Класа `JPasswordField`

Swing компонента `JPasswordField` представља користан алат за тражење корисничког имена и шифре од корисника на мање-више безбедан начин.

```
public class JPasswordField extends JTextField
```

Компонента се понаша скоро потпуно исто као текстуално поље. Међутим све што корисник укуца види се као звездица. На овај начин шифра је безбедна у случају да неко „преко рамена“ корисника гледа на екрану шта је корисник куцао.

Ова класа такође чува шифру као `char[]` тако да можемо да препишемо шифру нулама када нам више није потребна.

Класа поседује метод:
```
public char[] getPassword( )
```

Осим њега, углавном се користе методи наслеђени од суперкласе `JTextField`.

Пример 7:
Swing-based потклаcа `DialogAuthenticator` класе `Authenticator` која отвара дијалог како би од корисника добила корисничко име и шифру. Већина кода рукује графичким корисничким интерфејсом. Шифра се уноси у `JPasswordField`, а корисничко име у `JTextField`.

Класа `SecureSourceViewer` представља измењени програм `SourceViewer` који тражи корисничко име и шифру од корисника користећи класу `DialogAuthenticator`.