Bevezetés

Ebben a dokumentumban összeszedem, hogy milyen szabványokat, ajánlásokat, technológiákat, keretrendszereket és segédalkalmazásokat használunk a Java-s fejlesztéseinknél. Megtalálható lesz benne a hivatalos weboldal, ahonnan letöltöttük a third party alkalmazásokat, összegyűjtöm, hogy a működéséhez milyen függvénykönyvtárakra van szükség, mi kell a konfigurációhoz és egy rövid leírást is adok, hogy könnyebben megérthessük, elkezdhessük a munkát. Nem célom mindenhez teljes leírást adni, hiszen ez nem is lenne lehetséges, ezért vannak a megfelelő oldalakon a teljes Reference-k. „Csak" egy irányt és segítséget próbálok nyújtani az elinduláshoz.

Ezek az információk függetlenek a fejlesztőkörnyezettől. Két IDE terjedt el, amit érdemes tanulmányozni, az Eclipse és a Netbeans, melyek mind képesek az alábbiak kezelésére. Ahol különbség van, ott azt igyekszem külön jelezni.

A dokumentum írása közben használt szoftverek verziói a következők voltak:

- Operációs rendszer
    - Windows XP version 5.1 running on x86; Cp1252; en_GB (nb)
- JVM
    - jdk1.6.0_14
- Szerverek
    - apache-tomcat-5.5.27
    - Sun Java System Application Server 9.1_02 (build b04-fcs)
- IDE-k
    - NetBeans IDE 6.1 (Build 200810140114)
    - Eclipse Platform Version: 3.3.2 Europa (Build id: M20080221-1800)

1.1 Rövidítések

IDE	Integrated development environment (Integrált fejlesztőkörnyezet)
JVM	Java Virtuális Gép (Java Virtual Machine)
JDBC	Java DataBase Connectivity
JNDI	Java Naming and Directory Interface
$WSDIR	A workspace könyvtár elérési útvonala és neve
$PROJECT	A projekt neve
$PROJECTDIR	A projekt könyvtár elérési útvonala és neve (azonos $WSDIR/$PROJECT)

1.2 Tartalomjegyzék

1.3 A projekt könyvtár szerkezete

Egy alkalmazás fejlesztését egy projekt keretében végezzük, ami egy könyvtárba dolgozik, ahol jellemzően négy csoportba tartozó fájlok vannak. Ezek a csoportok a következők:

- külső függvények, library-k (`libs`)
- forráskódok, templatek, konfigurációs állományok (`src`, `web` vagy `webapp`, `conf` vagy `resource`, `java`)
- teszteléshez kódok és adatok (`test`)
- lefordított, futtatható/telepíthető állományok, bináris kódok (`bin`, `build`, `dist`, `target`)

Ezek automatikusan létrejönnek a fejlesztőkörnyezetekben, de konfigurálhatóak, így átnevezhetőek illetve mozgathatóak, ahogy logikus. Például test lehet a projekt könyvtárában közvetlenül vagy az src alatt is. Ugyanígy a web vagy webapp (ami ugyanaz, csak más néven a két IDE-ben) is előfordulhat mindkét helyen. Ha több projekt is használja ugyanazokat a külső függvényosztályokat, akkor azokat egy magasabb szintre szoktuk helyezni, a projekten kívül. Jellemzően a `...\.m2\repository` könyvtárat adjuk meg konténernek. Innen tudja az összes alkalmazás használni a közös library-ket. Sőt a különböző IDE-k is nyúlhatnak ide. Létezik eszköz ezek frissítésére, ami automatikusan letölti az Internetről a megfelelő osztályokat.

Egy jó, nálunk Eclipse-ben már használt könyvtárszerkezet a következő:

```
Projektnév
├──build
│   ├──classes
│   │   └──...
│   └──test-classes
│       └──...
├──dist
├──libs
└──src
    ├──main
    │   ├──java
    │   ├──resource
    │   └──webapp
    └──test
        ├──java
        ├──resource
        └──webapp
```

Ugyanez a könyvtárszerkezet Netbeans-ben is használható.

IDE-k konfigurálása

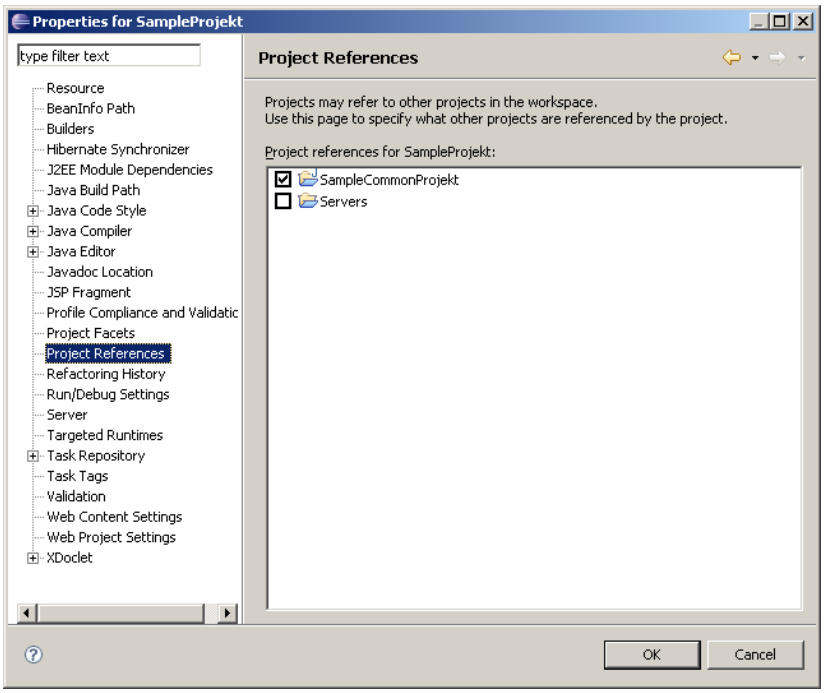
1.3.1 Eclipse

Az Eclipse felépítését három szinten kell elképzelni, így konfigurálni is három szinten lehet.

A legfelső szint az IDE alkalmazás szintje, ezt nem módosítjuk gyakran, csak a komponensek telepítése, frissítése van itt nyilvántartva.

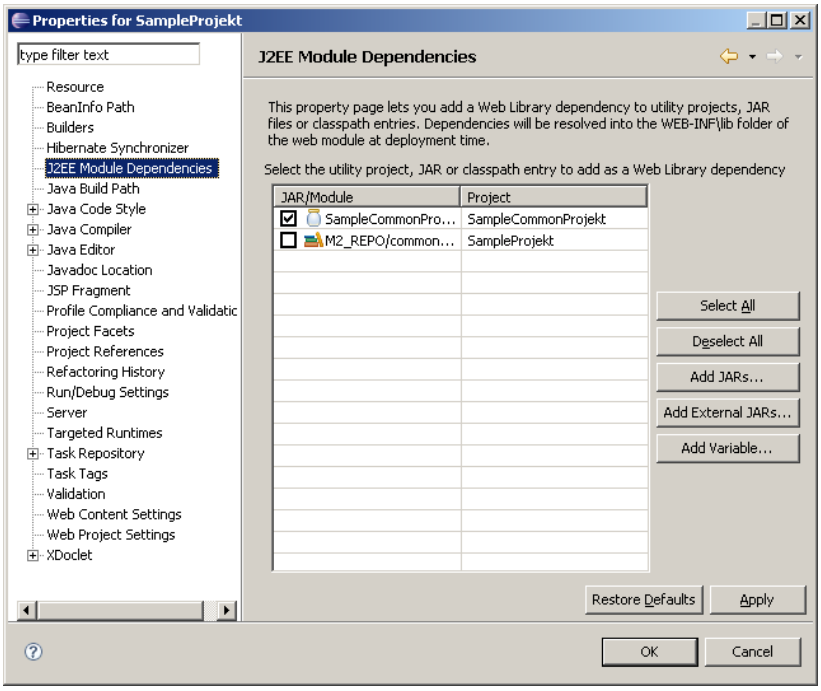
A legalsó szint a projekt szintje, ami teljesen a saját fejlesztésünk, így ezt gyakran konfigurálhatjuk a fejlesztés során. (A projekt nevén kell az egér jobb gombjával kattintani és a lebegő menüben a Properties-t választani vagy Alt+Enter billentyűkombinációt használni.)

- `$PROJECTDIR/.classpath`
    - A fájlok helyéről (bemenet: src, kimenet: build), külső függvénykönyvtárak használatáról tartalmaz információkat. Ez a fejlesztéskor az ellenőrzéshez és a fordításhoz kell. Például: `<classpathentry kind="src" path="/SampleCommonProjekt"/>`



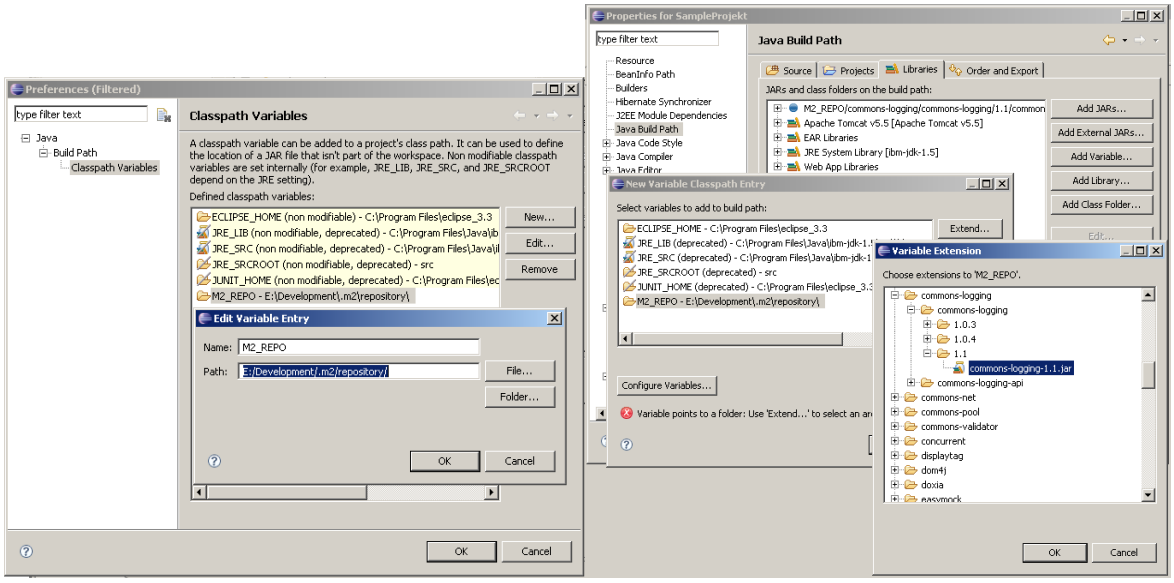
- `$PROJECTDIR/.project`
    - A fordításhoz szükséges kiegészítő információkat tartalmazza
- `$PROJECTDIR/.settings/...`
    - Ebben a könyvtárban speciálisabb információkat tárolunk a projektről, nem is mindig létezik ilyen (pl. Common, ami csak osztályokat tartalmaz). De például a Dinamikus Web Projekt-nél (Dinamic Web project) van ilyen, ahol le van írva, hogy milyen szerveren fusson teszteléskor, milyen könyvtárakat használjon a szerver, mi a platform verziója, ...
    - Itt van beállítva, hogy milyen függőségek vannak a futáshoz, mit kell a deploynál használni: `$PROJECTDIR/.settings/org.eclipse.wst.common.component` fájlban `<dependent-module deploy-path="/WEB-INF/lib"`

```
handle="module:/resource/SampleCommonProjekt/SampleCommon
Projekt">
```



A középső szint a munkaterületünk (workspace) szintje, ahol több projekt, segéd projekt lehet, így a konfigurálásával vigyázni kell, mert több helyen lehet hatása.

- `$WSDIR/.metadata/...`
    - Itt vannak beállítva a workspace adatai. Például az M2_REPO változó értéke az `$WSDIR\.metadata\.plugins\org.eclipse.core.runtime\.settings\org.eclipse.jdt.core.prefs` fájlban.



A módosított adatok néha (még nem találtam benne rendszert) csak az IDE újraindítása után kerülnek felhasználásra!

Az eddig elkészült példa projektet ki is lehet próbálni. Mellékelve van az eclipse_ws_ELTE_1.zip.

1.3.2 Netbeans

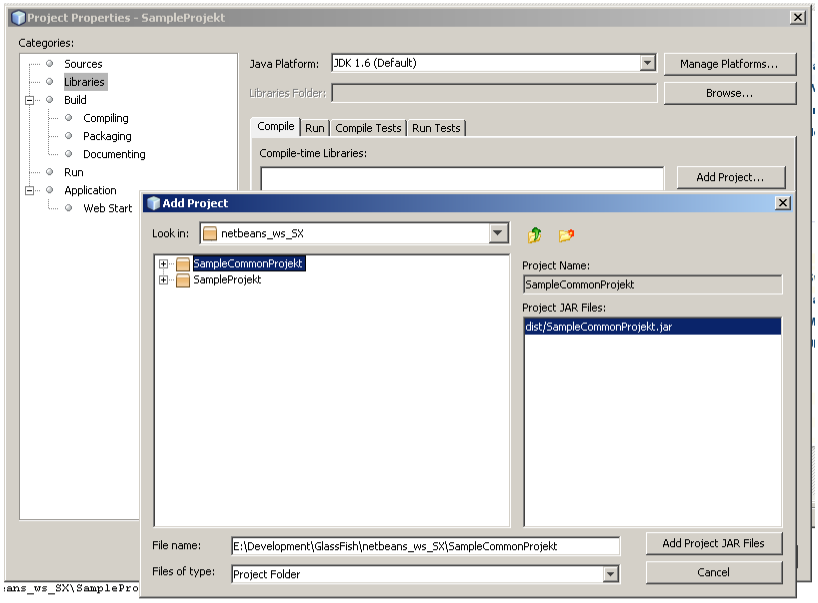
A Netbeans-ben is létezik a környezet szintje, mint konfigurálási, frissítési szint.

A munkaterület nem olyan éles, külön konfigurálható szint, mint az előbb. Csak csoportokba lehet fogni a projekteket (Project Group), hogy ne kelljen az összes munkát betölteni, csak az éppen használtakat. Nincs külön lehetőség konfigurálásra.

A projekt szintje itt is külön konfigurálható. (A projekt nevén kell az egér jobb gombjával kattintani és a lebegő menüben a Properties-t választani vagy Alt+F majd T billentyűkombinációt használni.)

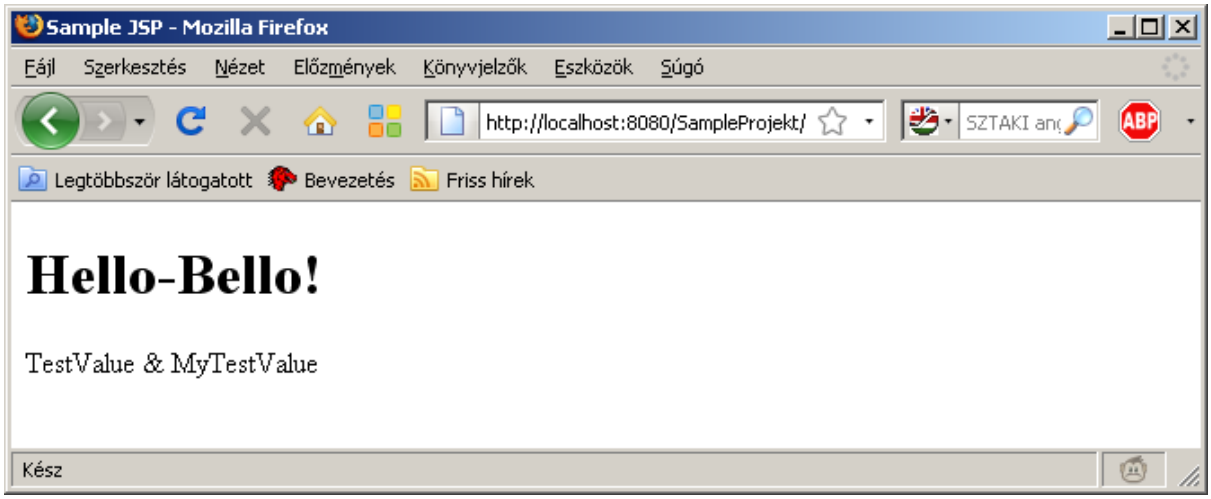
A konfigurációs fájlok az `nbproject` nevű könyvtárba kerülnek.

Például, ha hozzáadjuk a SampleCommonProjekt-et a függvénykönyvtárak közé, hogy leforduljon az alkalmazásunk, akkor módosításra kerül a build-impl.xml, project.properties és project.xml fájlok is.



Az eddig elkészült példa projektet ki is lehet próbálni. Mellékelve van a netbeans_ws_ELTE_1.zip.

Látszik, hogy mindkét környezetben ugyanazt a forráskódot használtuk. Az Eclipse-hez a Tomcat 5.5-t konfiguráltam be, míg a Netbeans az integrált Sun Application Serverét használta a Web-es alkalmazás futtatásához. Minkét esetben a böngészőben ezt láthatjuk:



Építőelemek

Ebben a fejezetben végigveszem a fejlesztésekhez szükséges építőelemeket olyan sorrendben, ahogy azok egymásra épülnek. De előre veszem a projekt kezelését, a tesztelést, majd a naplózást, mert ezeket kiemelten fontosnak tartom. Illetve az adatbázis elérés bekonfigurálását, mert ez a szerveren történő beállítás és nem tartozik szorosan a Java-s fejlesztéshez. A Verzió mindig a dokumentum készülésekor elérhető utolsó stabil verziót jelenti, mely az olvasáskor már lehet, hogy nem a legfrissebb, így ekkor javasolt az utolsó stabil verzió használata, hiszen a továbbfejlesztés általában jobb, hatékonyabb kódot tartalmaz és abban már javították az előzőben észrevett hibákat is. A verziók közötti eltérések miatt viszont egy projekt elején kiválasztott verziót már a projekt során ne nagyon váltogassuk, csak teljes teszt után!

1.4 Maven

Ez a segédalkalmazás projektek menedzselésére és kezelésére. Alapja a projekt(objektum)-modell (POM, Project Object Model). Képes projektek alapját elkészíteni az egyes IDE-khez, melyekhez integrálható. Tudja a szükséges library-k automatikus beszerzésére, Internetről letölti őket. A projektet lefordítja, előtte automatikus unit-teszteket tud futtatni az alkalmazáson, deployolni. …

1.4.1 Verzió

2.1.0

1.4.2 Website

http://maven.apache.org/

1.4.3 Letöltés

http://maven.apache.org/download.html

1.4.4 Library-k

%M2_HOME%/lib/maven-2.1.0-uber.jar

1.4.5 Konfiguráció

1.4.5.1 Maven konfigurálása

A telepítéskor kell konfigurálni. Be kell állítani környezeti változókhoz az M2_HOME, M2 és MAVEN_OPTS értékeit és módosítani kell a PATH-t.

```
M2_HOME=C:\Program Files\Apache Group\apache-maven-2.1.0
```

(ahova telepítettük)

```
M2=%M2_HOME%\bin
```

(futtató binárisok helye)

```
MAVEN_OPTS=-Xms256m -Xmx512m
```

(opcionális, memória használati adatok)

```
PATH=%PATH%;%M2%
```

(bárhonnan lehessen futtatni az mvn parancsot)

`%M2_HOME%\conf\settings.xml`-ben be tudjuk állítani a következő node felvételével, hogy mi legyen a letöltött függvénykönyvtárak tárolási mappája:
`<localRepository>...\.m2\repository</localRepository>`

1.4.5.2 Konfiguráció a használathoz

Az alkalmazást a projekt könyvtárából kell futtatni, ahol az egyes futásokhoz a projektre jellemző kiegészítő információkra van szükség, ezért itt létre kell hozni a `$PROJECTDIR\pom.xml` fájlt, mely tartalmazza ezeket a szükséges adatokat. Erről bővebben is olvashatsz:
http://maven.apache.org/pom.html

1.4.6 Rövid leírás

A munkát itt kezd: http://maven.apache.org/guides/getting-started/index.html Próbáld ki a parancsokat! Az eredményt a `MvnTest.zip` tartalmazza. Új projekt létrehozása a következő paranccsal:

```
mvn archetype:create -DgroupId=hu.elte.app -DartifactId=MvnTest
```

Ekkor elkészül egy új projekt a fenti könyvtárszerkezettel.

Hogy ne az alapértelmezett `target` könyvtárba, hanem a `build` könyvtárba készítse a binárisokat, módosítani kell a `pom.xml`-t a következő hozzáadásával:

```
<build>
  <directory>build</directory>
  <outputDirectory>build/classes</outputDirectory>
  <finalName>${project.artifactId}-${project.version}</finalName>
  <testOutputDirectory>build/test-classes</testOutputDirectory>
  <sourceDirectory>src/main/java</sourceDirectory>
  <testSourceDirectory>src/test/java</testSourceDirectory>
</build>
```

Majd le kell fordítani a projektet a következő paranccsal:

```
mvn compile
```

Az alkalmazást le tudjuk futtatni, ha belépünk a `build/classes` könyvtárba a `cd build\classes` paranccsal majd a `java hu.elte.app.App` utasítást adjuk ki.

Az alkalmazásból deploy-olható JAR is készíthető így:

```
mvn package
```

Ezt tesztelni a `java -cp build\MvnTest-1.0-SNAPSHOT.jar hu.elte.app.App` paranccsal lehet.

Tesztelhetjük az `mvn test` paranccsal a projektünket.

Használjuk még azt, hogy az előző fordítást töröljük, és újra fordítunk, amit külön-külön is kiadhatunk, de egy parancsban is:

```
mvn clean package
```

Ilyenkor a build könyvtár törlésre, majd újra létrehozásra kerül a friss fájlokkal.

Az `mvn install` a helyi repository-nkba telepíti az projektünket, hogy azt más projektek használhassák. (Ezt most rosszul használjuk!)

A `pom.xml` fájlban kell megadni azt is, hogy milyen külső függvénykönyvtárakra van szükség.

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.12</version>
    <scope>compile</scope>
  </dependency>
</dependencies>
```

A verziószám megadása kötelező és mindig azt fogja használni, ezt tölti le a távoli repo-ból, az Internetről (http://repo1.maven.org/maven2/), amit központilag tárolnak és frissítenek. A `scope` node-ban adhatjuk meg, hogy mikor (tesztelés, fordítás, ...) kell az adott library.

Ugyanitt adhatjuk meg, hogy a projekt build-elésekor milyen Java verziójú a forrás. Az annotációk kezeléséhez szükséges beállítani az 1.5-ös verziót!

```
<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <configuration>
      <source>1.5</source>
      <target>1.5</target>
    </configuration>
```

```
  </plugin>
  <!-- ... any other configurations -->
</plugins>
```

1.5 jUnit

A jUnit egy tesztelő keretrendszer, melyet fejlesztőknek és tesztelőknek ajánlanak a gyorsabb és automatizált teszteléshez. Nagyon hasznos eszköze a Test Driven Development-nek.

1.5.1 Verzió

4.6

1.5.2 Website

http://www.junit.org/

1.5.3 Letöltés

http://sourceforge.net/project/showfiles.php?group_id=15278&package_id=12472

Vagy a Maven repositorában is elérhető: http://repo1.maven.org/maven2/junit/junit/4.6/

1.5.4 Library-k

junit-4.6.jar

1.5.5 Konfiguráció

Külön konfigurálást nem igényel, a CLASSPATH beállítására lehet szükség, ha önálló alkalmazásként futtatjuk, de az IDE-knél külön kell a Library-t hozzáadni, a Maven-hez pedig a dependency-ként kell felsorolni a pom.xml-ben.

1.5.6 Rövid leírás

Nulladik megközelítésben az alkalmazásainkhoz írt osztályokat úgy teszteljük, hogy írunk hozzájuk egy kis alkalmazást, amit lefuttatunk és az a konzolra kiírja, hogy az osztályunk jól működik-e vagy sem (Unit Test). Esetleg több tesztesetet használva, több bemenetre várjuk a megfelelő választ, amiket a main eljárás hívogathat meg egymás után.

Ezt a módszert szabványosítja és segíti a jUnit **első megközelítés**ben. A `netbeans_ws_ELTE_2/MySampleTest.java` erre mutat példát. A tesztelő osztályunknak két külső package-re kell hivatkoznia:

```
import org.junit.*;
import static org.junit.Assert.*;
```

A kis alkalmazásunk main eljárása csak annyit kell hogy tartalmazzon, hogy futtatja magát, a tesztelő osztályt:

```
    public static void main(String args[]) {
       org.junit.runner.JUnitCore.main("hu.elte.rendszer.nev.MySampleTest");
    }
```

Az egyes teszteseteket sorban lefuttatja. Onnan tudja, hogy mely eljárások a tesztestek, hogy eléjük oda kell írni a következő annotációt:

```
@Test
```

Ezeken kívül még tartalmazhat más függvényt is az osztály, és még négy speciális eljárást is, melyek a tesztesetek futása előtt, illetve után futnak le. Az osztályunk inicializálása előtt egy static eljárás futhat le csak egyszer:

```
@BeforeClass
public static void ...
```

Ugyanígy a tesztelés után is lefuthat egyszer egy eljárás a teszt legvégén:

```
@AfterClass
public static void ...
```

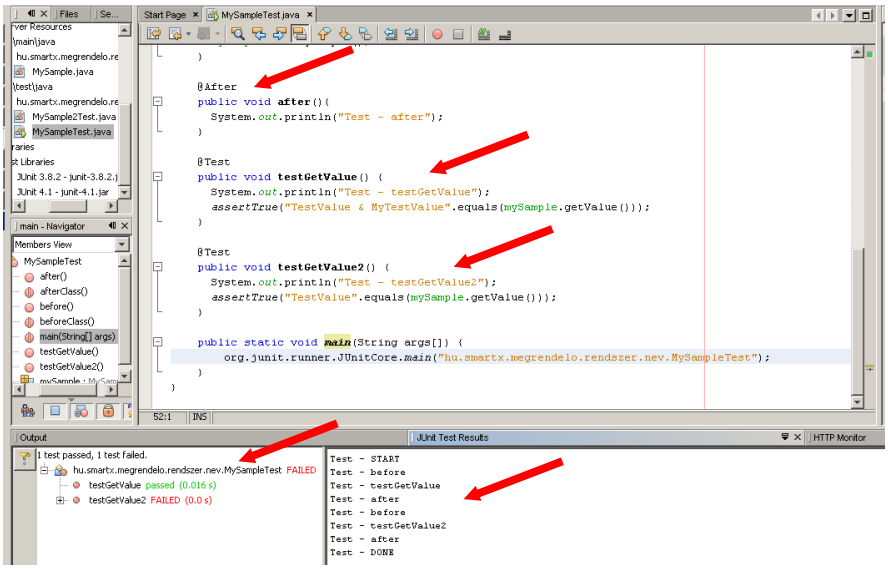
Minden teszt előtt lefut a következő:

```
@Before
public void ...
```

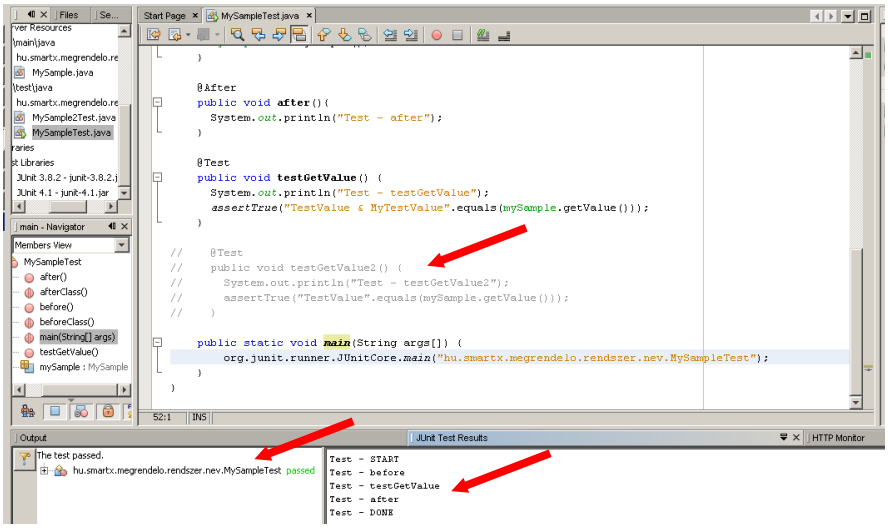
Az előzőhöz hasonlóan minden teszt után lefut ez:

```
@After
public void ...
```

Az osztályt Netbeans-ben a `Shift+F6` billentyűkombinációval futtathatjuk, az eredmény a következő:



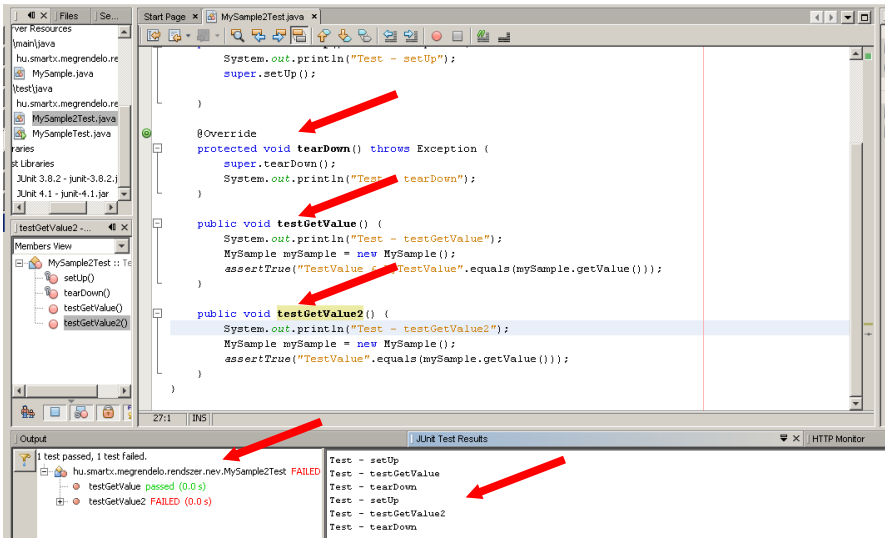
Látszik, hogy az osztály tesztelése sikertelen, mert a második teszteset hibás (FAILED), és a befor illetve after kétszer lefutott, mindkét teszteset előtt illetve után. A konzolon a START, before, testGetValue, after, before, testGetValue2, after, DONE kiírás jelenik meg.



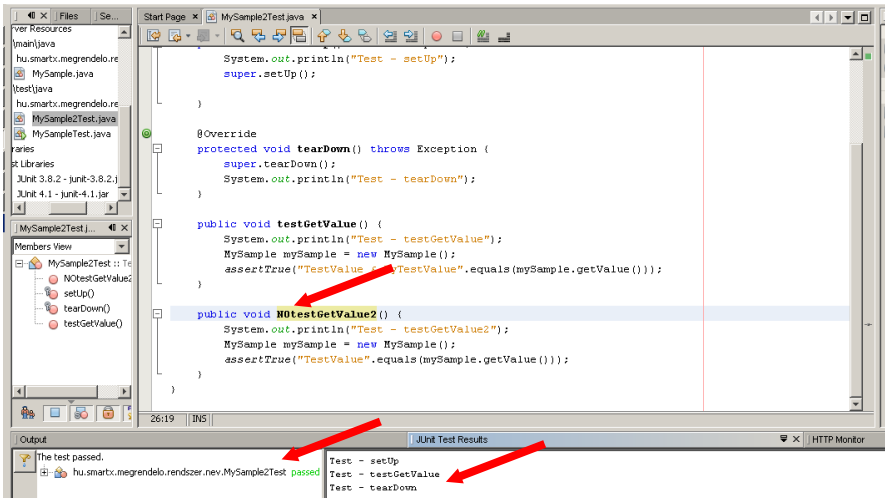
A második tesztesetet kikommentezve láthatjuk, hogy az osztály tesztelése sikerült (passed) és csak a START, before, testGetValue, after, DONE kiírások vannak.

A **második megközelítés** már nem saját tesztprogramot feltételez, hanem csak a tesztesetek elkészítését. Nincs lehetőség az osztály előtt és után a statikus eljárások definiálására, a tesztesetek előtt és után pedig a szabványos nevű `setUp` illetve `tearDown` nevű eljárásokat kell felüldefiniálnunk. Felüldefiniálnunk, hiszen a saját tesztelő osztályunkat a TestCase osztályból származtatjuk le, ami a `junit.framework` package-ben érhető el. A teszteseteket tartalmazó eljárások neveinél pedig `test` a prefix (`testGetValue`).

Ezt is `Shift+F6` billentyű kombinációval futtathatjuk Netbeans-ben.



Így ha a második eljárás nevének elejét módosítjuk, akkor már jó lesz a teszt.



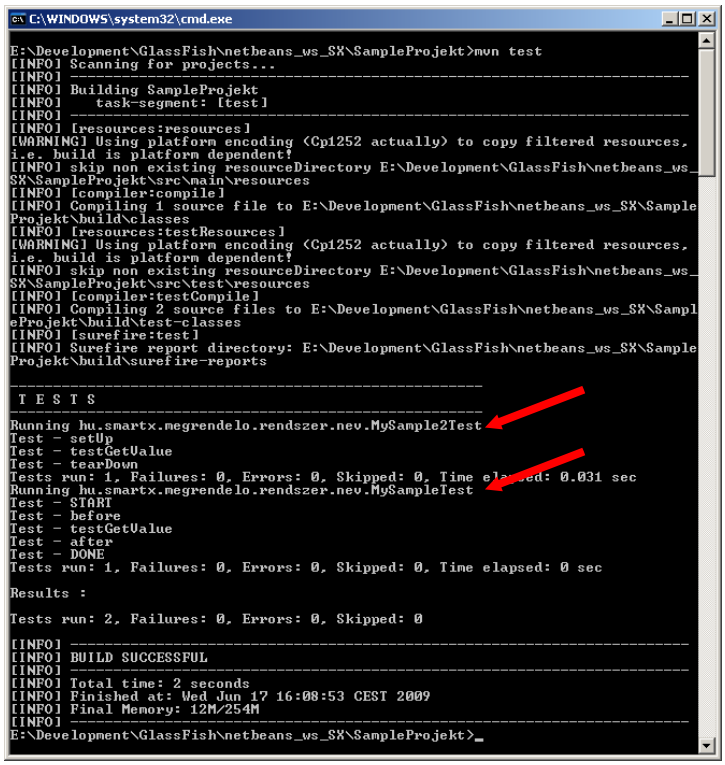
Bővebb leírást itt találhatunk:

- http://junit.sourceforge.net/
- http://junit.sourceforge.net/doc/cookbook/cookbook.htm
- http://junit.sourceforge.net/doc/faq/faq.htm

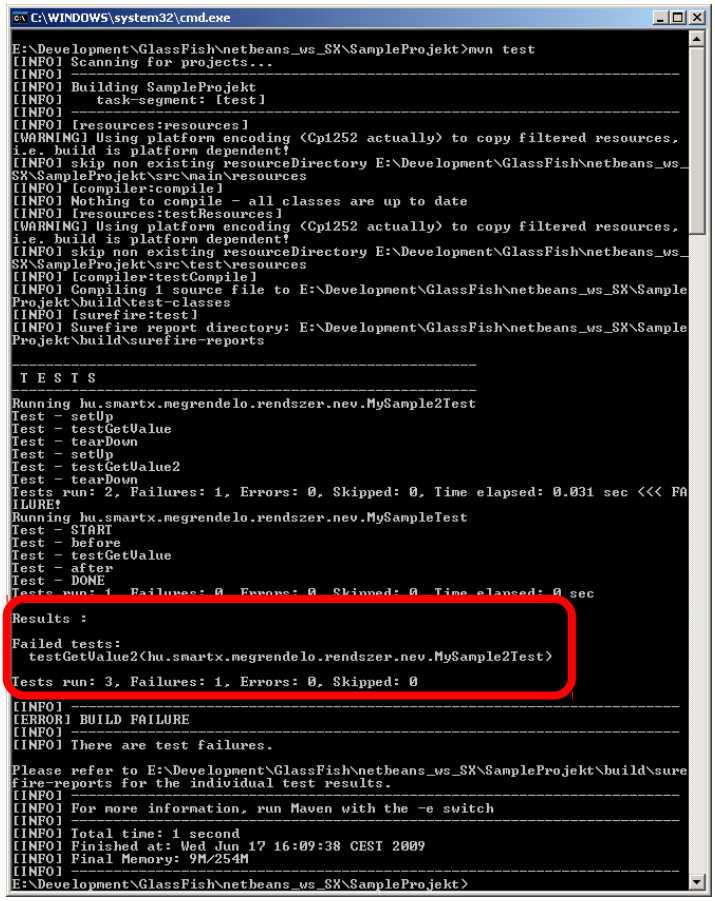
1.5.7 Használat Maven-nel

Az `mvn test` paranccsal tudjuk futtatni a teszteseteket, az összes `test` könyvtárban lévő `Test` postfix-ű osztályt. Fontos tudni, hogy az `mvn package` parancs is futtatja ezeket, és csak akkor fordul le a projekt, ha minden teszten átment előtte.

A Maven az `org.junit`-os és a `junit.framework`-ös megoldást is kezeli, és a `test` könyvtárában elérhető összes osztályt felhasználja a tesztek futtatásához.



Ha most az eljárás nevét visszajavítom, akkor a Maven is hibát jelez a teszt futtatásánál:



Látható az eredmények között, hogy a sikertelen tesztek listájában a MySample2Test osztály testGetValue2 eljárása volt a hibás.

1.6 log4j

A log4j egy jó megvalósítási kerete a Java alkalmazások sokrétű és hatékony naplózására, mely általános célú és open source. Most a 2.0 még fejlesztés alatt áll. Az 1.2 az utolsó stabil verziójú branch. Segítségével egységesíthetjük a visszajelzéseket a fejlesztőknek, tesztelőknek és a felhasználóknak (adminoknak).

1.6.1 Verzió

1.2.15

1.6.2 Website

http://logging.apache.org/log4j/

http://logging.apache.org/log4j/1.2/apidocs/index.html

1.6.3 Letöltés

http://logging.apache.org/log4j/1.2/download.html

Vagy a Maven repositorában is elérhető: http://repo1.maven.org/maven2/log4j/log4j/1.2.15/

1.6.4 Library-k

log4j-1.2.15.jar

1.6.5 Konfiguráció

1.6.5.1 Általános konfigurálás

Külön konfigurálást nem igényel, a CLASSPATH beállítására lehet szükség, ha önálló alkalmazásként futtatjuk, de az IDE-knél külön kell a Library-t hozzáadni, a Maven-hez pedig a dependency-ként kell felsorolni a pom.xml-ben.

1.6.5.2 Konfigurálás alkalmazásonként

Az alkalmazást egy szöveges fájlban egyszerűen lehet konfigurálni. Példákat a leírásban olvashatsz.

1.6.6 Rövid leírás

A logolás megvalósítása három fő részből áll. A forrásban használjuk a **logger**-eket. A tényleges logolást az **appender**-ek végzik. A **layout**-ok pedig a logok formázásáért felelősek.

Úgy működik, hogy az osztályainkhoz statikus logger osztályokat rendelünk, és azokat használva az osztályunkban az osztályra vonatkozó logbejegyzések íródnak. Többszintű logolást tesz lehetővé, úgy hogy egy logolási szint beállítható, majd ezután csak az ennek megfelelő, illetve e szint fölötti logok kerülnek kiírásra. A következő szintek definiáltak: `ALL < DEBUG < INFO < WARN < ERROR < FATAL < OFF`, a `WARN` beállításakor tehát a `WARN`, `ERROR` és `FATAL` szint is íródik.

A tényleges kiírást appenderek végzik (append, hozzáfűz), melyek több kimeneti csatornát is tudnka kezelni. lehet írni, fájlba, a syslog-ba, queue-ba(JMS), adatbázisba, de a leggyakrabban a fájlba írást használják, mert ez a leggyorsabb. Az adatbázisba íráshoz pl létezik a JDBCAppender, de ezt a log4j-ben is lecserélik, nálunk is lassú volt, ezért inkább ne használjuk (helyette megoldható, hogy a fájl betöltésre kerül egy adatbázisba, ha szükséges; egy csúnya, de ötletes megoldás, ha rögtön INSERT INTO SQL formában írjuk fájlba az adatokat).
(http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/AppenderSkeleton.html)

A fájlba történő íráskor beállítható, hogy egy bejegyzés, milyen formában (layout) kerüljön kiírásra, mit tartalmazzon egy adatcsomag (általában egy sor, egy bejegyzést jelent, vagyis a minta leggyakrabban `%n`-vel végződik.). Lásd még:
http://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/PatternLayout.html

Külön loggereket lehet definiálni a különböző ***package-hez, osztályokhoz egy fa struktúra szerint***. A loggerekhez megadható a logolási szint és az appender. Az appenderhez pedig beállítható a hozzá szükséges paraméterek és/vagy a mintája. A konfigurációs fájl közvetlenül is megadható a forrásban (`PropertyConfigurator.configure(logFilePath);`), de a CLASSPATH-ban keresi a `log4j.properties` nevű fájlt alapértelmezettként (web-es alkalmazásban a classes könyvtárban, pl. a resources könyvtár forráskönyvtárként való beállítása után, ez automatikusan oda kerül package készítésekor.). A konfigurációs fájl egy property fájl (`név=érték` párokkal), de lehet XML formátumú is, ekkor `log4j.xml` néven kell elhelyezni.

1.7 JDBC Resource (Oracle) és JNDI

Fejlesztéseinknél általában Oracle adatbázist használunk az adatok tárolására. A Java-s alkalmazásainknak tehát el kell érniük az adatbázist. Az Oracle a Java-hoz külön függvénykönyvtárat

ad (ojdbc), hogy ne a tnsnames.ora-s magas szintű eléréssel kelljen az adatbázist megszólítani, hanem közvetlenül natív módon.

A konzolos alkalmazásunkban meg kell valósítani az adatbázishoz való kapcsolódást, tranzakció-kezelést, adatkezelést illetve a lekapcsolódást. Az alkalmazás- illetve webszerverek viszont támogatják az erőforrások kezelését, ami azért jó, mert nem nekünk kell a kapcsolódást, az egyik legidőigényesebb feladatot elvégeznünk. Ezt a műveletet szerverek a háttérben megoldják. Egy ilyen JDBC erőforrásra a Java egy szabványos, úgynevezett JNDI szolgáltatásával tudunk hivatkozni.

Az újabb, robosztusabb alkalmazásszerverek (pl a Sun AppServer is) támogatja az úgynevezett ConnectionPool használatát, ami azért jó, mert több szálon egyszerre tudjuk használni az adatbázis kapcsolatokat. Csak egyet ki kell választanunk a több nyitott kapcsolati halmazból és már megy is.

1.7.1 Verzió
ojdbc14-10.2.0.1.0

1.7.2 Website
http://www.oracle.com/technology/tech/java/java_db/index.html

1.7.3 Letöltés
http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/index.html

http://www.oracle.com/technology/software/tech/java/sqlj_jdbc/htdocs/jdbc_10201.html

1.7.4 Library-k
ojdbc14-10.2.0.1.0.jar

1.7.5 Konfiguráció

1.7.5.1 Konzolalkalmazás
Nincs szükség konfigurálásra.

1.7.5.2 Apache Tomcat
A függvénykönyvtárat a server függvénykönyvtárai közé kell másolni, nekem a `"C:\Program Files\Apache Group\apache-tomcat-5.5.27\common\lib\"` könyvtárba kellett.

A konfiguráció három részből áll, ezért három különböző konfigurációs fájlt kell módosítani. Először be kell állítani a szervernek az erőforrást a következő node hozzáadásával:

`...\apache-tomcat-5.5.27\conf\server.xml`

```
<Resource accessToUnderlyingConnectionAllowed="true"
auth="Container" defaultAutoCommit="true"
driverClassName="oracle.jdbc.driver.OracleDriver"
factory="org.apache.tomcat.dbcp.dbcp.BasicDataSourceFactory"
initialSize="0" logAbandoned="true" maxActive="2"
name="jdbc/oracleSample/server" password="sample"
removeAbandoned="true" removeAbandonedTimeout="10"
type="javax.sql.DataSource" url="jdbc:oracle:thin:@IP:port:SID"
username="sample"/>
```

Természetesen a megfelelő adatbázis-kapcsolati adatokat kell megadni. Ez a ***példa adatbázis*** először létrehozandó.
Második lépésként az alkalmazásunk `WEB-INF\web.xml`-jében be kell állítani azt a JNDI nevet, amit a forráskódban is használunk:

```
<resource-ref>
 <res-ref-name>jdbc/oracleSample</res-ref-name>
 <res-type>javax.sql.DataSource</res-type>
 <res-auth>Container</res-auth>
 <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

Végül a szerveren definiált és az alkalmazásban használt JNDI neveket össze kell kapcsolnunk a `META-INF\contex.xml`-ben:

```
<ResourceLink global="jdbc/oracleSample/server"
name="jdbc/oracleSample" type="javax.sql.DataSource" />
```

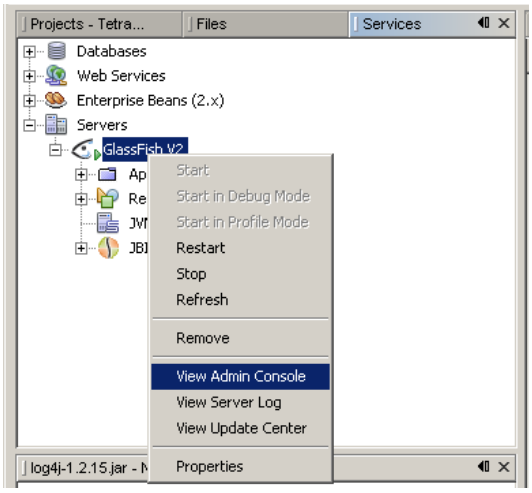
Ezt megtehettük volna úgy is, hogy mindenhol ugyanaz a név (jdbc/oracleSample), de elképzelhető, hogy a szerveren több adatbázishoz definiálunk erőforrás-leírást (pl. tesztadatbázishoz), és így gyorsabban tudunk váltani az adatbázisok között.

1.7.5.3 Sun Application Server (Netbeans)

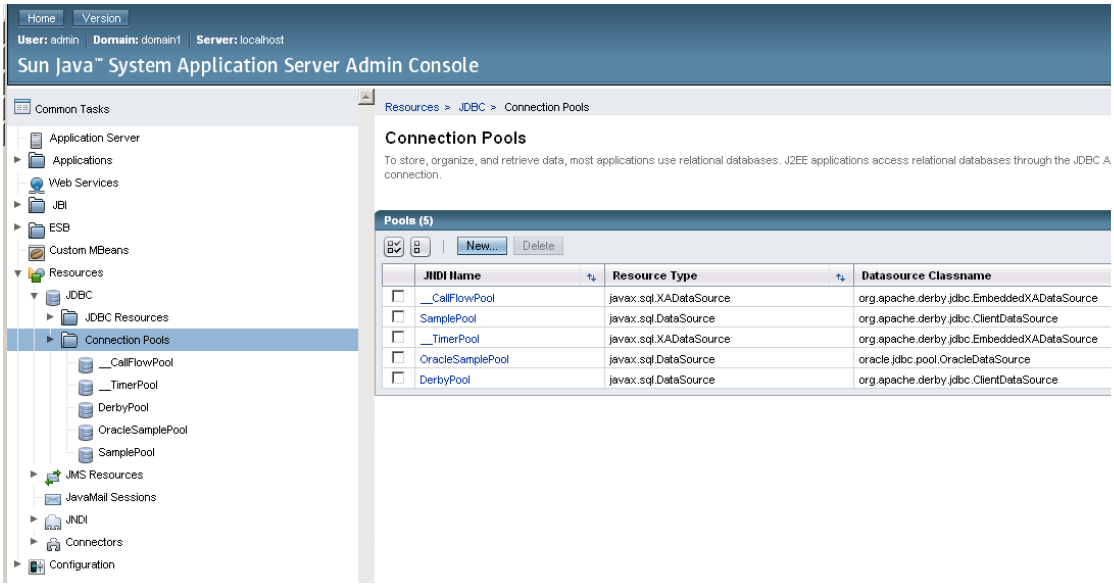
A függvénykönyvtárat a server függvénykönyvtárai közé kell másolni, nekem a `"C:\GlassFishESB\glassfish\lib\"` könyvtárba kellett.

A konfiguráció itt is három részből áll, ezért három különböző konfigurációs fájlt kell módosítani. Először be kell állítani a szervert, amit nem kézzel végzünk el, hanem az adminisztrátori felületen keresztül.

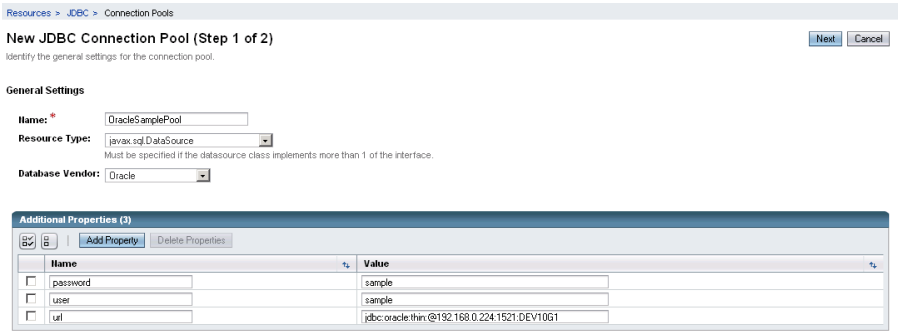
Az adminisztrátori felület böngészőben jelenik meg, miután a szerver nevén az egér jobb gombjával kattintottunk és kiválasztottuk a „View Admin Console” menüpontot.



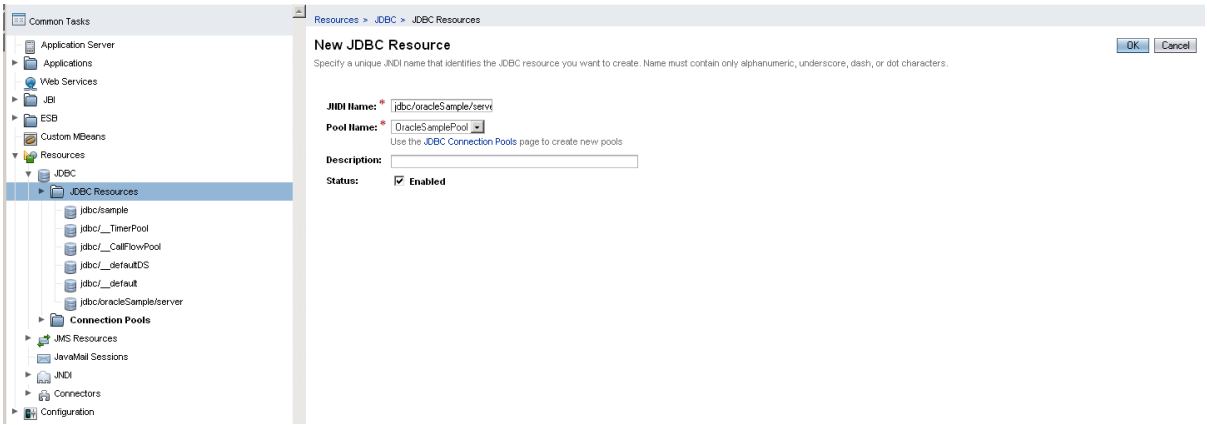
Itt először fel kell venni új Connection Pool-t (Resources/JDBC/Connection Pools oldalon a New... gomb):



Meg kell adni egy egyedi nevet (**OracleSamplePool**), és a második lapon három paramétert (URL= ***jdbc:oracle:thin:@IP:port:SID***, user=***sample***, password=***sample***) az adatbázis eléréséhez (a többit törölhetjük).



Ezután fel kell venni új JDBC Resource-t (Resources/JDBC/JDBC Resources oldalon a New... gomb):

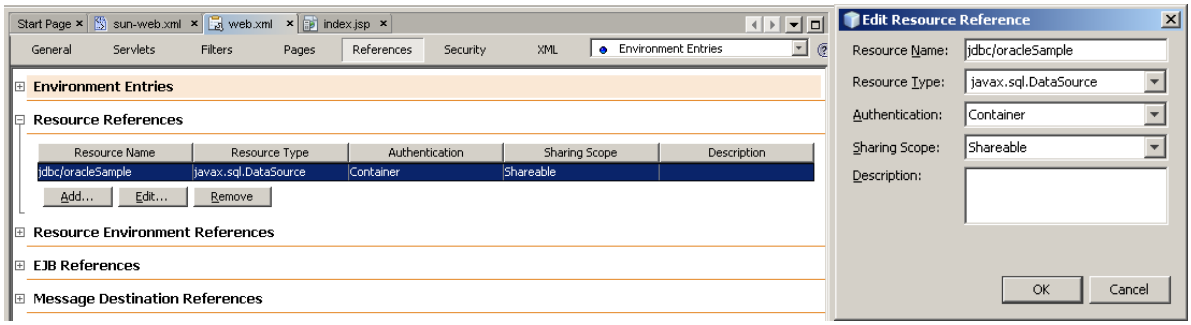


Meg kell adni egy egyedi nevet (**jdbc/oracleSample/server**) és ki kell választani a listából az előbb létrehozott Connection Poolt (**OracleSamplePool**).

A második lépés itt is ugyanaz, mint előbb, az alkalmazásunk `WEB-INF\web.xml`-jében be kell állítani azt a JNDI nevet, amit a forráskódban is használunk. Ezt beírhatjuk kézzel a fájl forrásába:

```
<resource-ref>
 <res-ref-name>jdbc/oracleSample</res-ref-name>
 <res-type>javax.sql.DataSource</res-type>
 <res-auth>Container</res-auth>
 <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

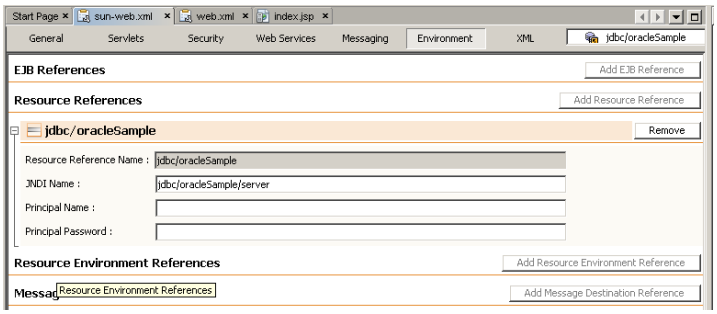
Vagy az IDE felületén állítjuk be, ami a Netbeans-ben így néz ki:



Végül a szerveren definiált és az alkalmazásban használt JNDI neveket össze kell kapcsolnunk a `WEB-INF\sun-web.xml`-ben kézzel:

```
<resource-ref>
 <res-ref-name>jdbc/oracleSample</res-ref-name>
 <jndi-name>jdbc/oracleSample/server</jndi-name>
</resource-ref>
```

Vagy felületen:



1.7.6 Rövid leírás

Az itt leírtakat majd a JSTL-ről szóló fejezetben nyílik alkalmunk kipróbálni. (tomcat_jdbc_jstl.zip; netbeans_ws_ELTE_4)

1.8 HTML és XHTML

A HTML (HyperText Markup Language=hiperszöveges jelölőnyelv) a weboldalak készítésére fejlesztettek, mára szabvánnyá vált. Az XHTML a HTML megjelenése XML-ben, mely csak abban különbözik, hogy szigorúbbak a formai követelmények.

1.8.1 Verzió

XHTML 1.0 Strict

1.8.2 Website

http://hu.wikipedia.org/wiki/HTML

http://www.w3.org/TR/xhtml11/

1.8.3 Letöltés

Nem kell letölteni semmit.

1.8.4 Library-k

Nincs szükséges függvénykönyvtár.

1.8.5 Konfiguráció

Nincs szükség konfigurálásra.

1.8.6 Rövid leírás

Az összes web-es fejlesztésünkhöz a HTML nyelv ismerete feltétel. Ebbe a dokumentumba azért került bele, hogy a teljes képet kaphassunk. Az alábbi képeken látható, hogy mely böngészők mennyire támogatják az egyes HTML dokumentum típusokat, azok mennyire platformfüggetlenek, ezért javasolt az XHTML 1.0 Strict használata.

Doctype	NS6	Old Moz	Moz & Safari	Opera 9	Opera 7.5	IE 7 & Opera 7.10	IE 6 & Opera 7.0	Mac IE 5	Konq 3.2
None	Q	Q	Q	Q	Q	Q	Q	Q	Q
HTML 3.2 doctype ie. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">	Q	Q	Q	Q	Q	Q	Q	Q	Q
HTML 4.0 Strict doctype without a URL ie. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">	S	S	S	S	S	A	A	A	A
HTML **4.01** Strict doctype **without a URL** ie. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">	S	S	S	S	S	A	A	Q	A
HTML 4.0 Strict doctype with a URL ie. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/html4/strict.dtd">	S	S	S	S	S	A	A	A	A
HTML 4.01 Strict doctype with a URL ie. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">	S	S	S	S	S	A	A	A	A
HTML 4.0 Transitional doctype **without a URL** ie. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">	Q	Q	Q	Q	Q	Q	Q	Q	Q
HTML 4.01 Transitional doctype **without a URL** ie. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">	Q	Q	Q	Q	Q	Q	Q	Q	Q
HTML **4.01** Transitional doctype **with the URL http://www.w3.org/TR/html4/loose.dtd** ie. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">	S	S	A	A	A	A	A	A	Q
HTML **4.01** Transitional doctype **with the URL http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd** ie. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd">	Q	S	A	A	A	A	A	A	Q
HTML **4.0** Transitional doctype **with a URL** eg. <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">	Q	Q	Q	Q	A	A	A	A	Q

Doctype	NS6	Old Moz	Moz & Safari	Opera 9	Opera 7.5	IE 7 & Opera 7.10	IE 6 & Opera 7.0	Mac IE 5	Konq 3.2
XHTML 1.0 Strict doctype **without** an XML declaration ie. `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">`	S	S	S	S	S	A	A	A	A
XHTML 1.0 Transitional doctype **without** an XML declaration ie. `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`	S	S	A	A	A	A	A	A	Q
XHTML 1.0 Strict doctype **with** an XML declaration eg. `<?xml version="1.0" encoding="UTF-8"?>` `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">`	S	S	S	S	S	A	Q	A	Q
XHTML 1.0 Transitional doctype with an XML declaration eg. `<?xml version="1.0" encoding="UTF-8"?>` `<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">`	S	S	A	A	A	A	Q	A	Q
ISO HTML **2000 version** doctype, **short form** ie. `<!DOCTYPE HTML PUBLIC "ISO/IEC 15445:2000//DTD HTML//EN">`	Q	S	S	Q	Q	Q	Q	Q	Q
ISO HTML **2000 version** doctype, **long form** ie. `<!DOCTYPE HTML PUBLIC "ISO/IEC 15445:2000//DTD HyperText Markup Language//EN">`	Q	S	S	S	S	A	A	A	Q
ISO HTML **1999 version** doctype, **short form** ie. `<!DOCTYPE HTML PUBLIC "ISO/IEC 15445:1999//DTD HTML//EN">`	S	S	S	Q	Q	Q	Q	Q	Q
ISO HTML **1999 version** doctype, **long form** ie. `<!DOCTYPE HTML PUBLIC "ISO/IEC 15445:1999//DTD HyperText Markup Language//EN">`	S	S	S	S	S	A	A	A	Q
HTML5, ie. `<!DOCTYPE html>`	Q	S	S	S	S	A	A	A	

Forrás: http://hsivonen.iki.fi/doctype/ (Q: Quirks; A: Almost Standard; S: Standard)

A validálás, érvényesítés után biztosak lehetünk benne, hogy a böngészők fel tudják dolgozni a fájlunkat. (http://validator.w3.org/)

http://www.w3schools.com/html/default.asp

http://www.w3schools.com/xhtml/default.asp

1.9 CSS

A CSS (Cascading Style Sheets = sorba kapcsolt stíluslapok) a HTML-vel együtt jelent meg. A HTML kódban szereplő tag-ek (elemek) stílusát tudja definiálni, de strukturális módosításra nincs lehetőség. Tehát az adatokat és azok strukturáját (pl lista vagy táblázat) a HTML tartalmazza, a színét, margót, ... pedig a CSS-ben írjuk le. Közös CSS fájl használatával a site-unk HTML oldalain egységes stílusban tudunk megjelenni és könnyű benne (egy helyen) javítani.

1.9.1 Verzió

CSS2 (=2.1)

1.9.2 Website

http://hu.wikipedia.org/wiki/CSS

http://www.w3.org/TR/CSS2/

1.9.3 Letöltés

Nincs szükség letöltésre.

1.9.4 Library-k

Nincs szükség kiegészítő függvénykönyvtárakra.

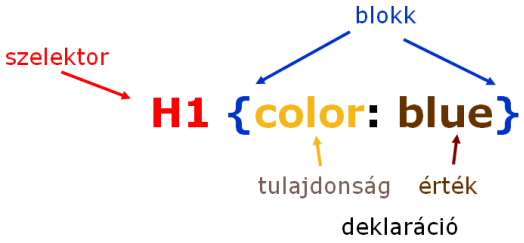
1.9.5 Konfiguráció

Nincs szükség konfigurálásra.

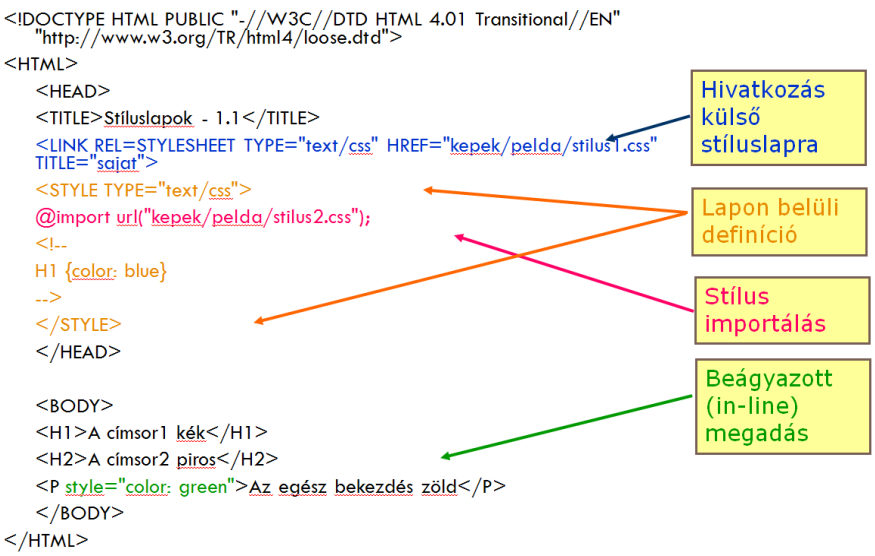
1.9.6 Rövid leírás

A CSS utasítás két részből áll:

- a szelektor tartalmazza a formázandó HTML tag megnevezését (pl. H1);
- a deklaráció végzi el a szelektorban meghatározott tag formázását.



A HTML oldalhoz többféleképpen is tudjuk kapcsolni:



Optimalizálni azért érdemes, mert gyorsabb lesz a feldolgozása a kliens oldalon. (http://www.cssoptimiser.com/)

A validálás, érvényesítés után pedig biztosak lehetünk benne, hogy a böngészők fel tudják dolgozni a fájlunkat. (http://jigsaw.w3.org/css-validator/)

http://www.w3schools.com/css/default.asp

1.10 JavaScript

A JavaScript segítségével **kliens oldalon** lehet **programozni a web-oldalakat**. Eredeti neve ECMAScript, melyet 1996 óta fejlesztenek (kezdetben a Netscape Navigator 2.0; 1996). ISO szabvány 1998 óta.

1.10.1 Verzió
JavaScript 1.8 [ECMAScript (ECMA-262) Edition 3]

1.10.2 Website
http://hu.wikipedia.org/wiki/JavaScript

1.10.3 Letöltés
Nincs szükség letöltésre.

1.10.4 Library-k
Nincs szükség külső függvénykönyvtárakra.

1.10.5 Konfiguráció
Nincs szükség konfigurálásra.

1.10.6 Rövid leírás
Java szintakszisú szkript nyelv, mely forrása a HTML oldalba ágyazható/betölthető és az oldalon található elemekkel tud műveleteket végezni, de sok segéd függvény tölthető le hozzá, mellyel nagyon szép és hasznos eljárást tudunk írni.

A JavaScript nagyon elterjedt, ehhez is sok keretrendszer érhető el, melyekkel platform és böngésző független kliens oldali programokat lehet írni. Érdemes lehet a JavaScript keretrendszereket tovább tanulmányozni. Ugyanígy a nagyon gyorsan terjedő AJAX-szal is érdemes foglalkozni.

http://www.w3schools.com/js/default.asp

1.11 JSP
A JSP (Java Server Pages) egy szöveges fájl, mely statikus, szöveges adatokat – jellemzően HTML kód – és a dinamikusan generált részekhez Java forrást tartalmazó elemeket tartalmazhat. Ez a forrásfájl fájlnév.jsp néven kerül fel a szerverre, de ott először java forrás, majd abból bináris kód generálódik, így gyorsabban fut, mintha futásidőben történne a feldolgozás (ez így van PHP-ben alapértelmezetten).

1.11.1 Verzió
2.1

1.11.2 Website
http://java.sun.com/products/jsp/?intcmp=2817

http://java.sun.com/javaee/5/docs/tutorial/doc/bnagx.html

1.11.3 Letöltés
Nincs szükség külön letöltésre.

1.11.4 Library-k
JSP: Nincs szükség külön függvénykönyvtárra.

1.11.5 Konfiguráció
Nincs szükség külön konfigurálásra.

1.11.6 Rövid leírás

A forrás kinézetre hasonlít a PHP-hez, itt is a HTML kódok közé írjuk be a Java forrást a <% és %> jelek közé. A fájl elején a lapra jellemző adatok (tartalom adatai; milyen külső osztályokat használ az oldal) beállításával kezdeni a munkát:

```
<%@page language="java" contentType="text/html; charset=ISO-8859-2"
pageEncoding="ISO-8859-2"%>
<%@page import="hu.elte.rendszer.nev.*" %>
```

Ahhoz csak néhány igen szükséges programozási feladatot (elágazás, iteráció) hajtsunk végre a megjelenítéshez az oldalunkon, jobban egységesítsük a JSP oldalainkat és még véletlenül se érezzünk kényszert arra, hogy ide helyezzünk el fontos, üzleti kódokat bevezették az úgynevezett tag library-ket. Lásd a következő fejezetet. A példakód még a következő fejezetet is tartalmazza. (netbeans_ws_ELTE_4)

1.12 JSTL

A JSTL (JavaServer Pages Standard Tag Library) segítségével a HTML-ben megszokott tag-ekhez hasonló formában íródnak a Java parancsok, és mögötte Java kód fut. Így elkerülhető, hogy csúnya Java kódrészletek kerüljenek a JSP fájlokba és nehezebbé váljon a tesztelés, hibakeresés.

1.12.1 Verzió

1.2

1.12.2 Website

http://java.sun.com/products/jsp/jstl/

http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/index.html

http://java.sun.com/javaee/5/docs/tutorial/doc/bnake.html

1.12.3 Letöltés

Nincs szükség letöltésre. Része a Java EE 5 platform-nak.

1.12.4 Library-k

jstl.jar, standard.jar

1.12.5 Konfiguráció

Nincs szükség külön konfigurálásra.

1.12.6 Rövid leírás

A JSP fájlokban fejlécében meg kell adni, hogy tag library-t fogunk használni névterekkel.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
```

Ezután az adott névtérben elérhetővé válnak a Java-ra hivatkozó tag-ek. Netbeans-ben a kódkiegészítés (Code completion) nagyban segíti a megfelelő parancsok kiválasztását, és a linkek között ott van a referencia is.

```
<sql:query var="users" sql="SELECT fullname from USERS"
dataSource="jdbc/oracleSample"/>
<table border="1">
 <c:forEach var="user" begin="0" items="${users.rows}">
  <tr><td><c:out value="${user.fullname}"/></td></tr>
```

```
 </c:forEach>
</table>
```

A fenti kód a **jdbc/oracleSample** (dataSource attributum) JNDI-vel hivatkozott adatbázisban futtatja le a select-et (sql attributum) (sql:query tag) és egy táblázat soraiban jeleníti meg őket egyesével végighaladva (c:forEach tag) rajtuk. Ezt tartalmazza a netbeans_ws_ELTE_4.

1.13 Template-ek, saját tag library írása

A fenti tag library-k nem feltétlenül elegek az alkalmazásunk megírásához. Így nagyon indokolt esetekben saját tag-eket vezethetünk be és használhatunk a fejlesztéseink során egy saját tag library elkészítéséve. Javasolt lenne ezeket céges szintre kivezetni és más alkalmazásokban is felhasználni.

A különböző keretrendszerek is elkészítették a saját tag library-jeiket, melyek segítségével nagyon könnyen, gyorsan szép web-es alkalmazások készíthetők.

Templatek, paraméteres templatek, taglibrary, saját fgv,

Folyt...

1.14 Display tag library

Ez a tag library csak táblázatok megjelenítésére használható.

1.14.1 Verzió

1.2

1.14.2 Website

http://displaytag.sourceforge.net/1.2/index.html

1.14.3 Letöltés

http://sourceforge.net/project/showfiles.php?group_id=73068

1.14.4 Library-k

displaytag-1.2.jar és ez még feltételezi a következőket is: commons-logging-1.1.jar, commons-lang-2.3.jar, commons-collections-3.2.jar, commons-beanutils-1.7.0.jar

1.14.5 Konfiguráció

A tag library leíró fájlját (displaytag.tld) a szerverre kell másolni, amit a `web.xml`-ben be is kell állítani, hogy használni tudjuk. Ennek a helye logikasan a `WEB-INF` könyvtárban esetleg annak egy alkönyvtárában van.

```
<jsp-config>
...
 <taglib>
  <taglib-uri>http://displaytag.sf.net</taglib-uri>
  <taglib-location>/WEB-INF/tld/displaytag.tld</taglib-location>
 </taglib>
...
</jsp-config>
```

1.14.6 Rövid leírás

A JSP fájlokban úgy tudunk külső tag library-t használni, hogy a fejlcében megadjuk a névteret (,amit az előbb a web.inf-ben beállítottunk).

```
<%@taglib uri="http://displaytag.sf.net" prefix="display"%>
```

Ebben a névtérben, már szerkesztéskor feljön, hogy milyen tag-eket használhatunk, vagy a referenciában olvasható.

Példaként a USERS tábla adatiból hármat kilistázunk:

```
<sql:query var="users" sql="SELECT u_id, username, fullname from USERS"
dataSource="jdbc/oracleSample"/>
<display:table name="${users.rows}" class="myTable">
 <display:column property="u_id" sortable="true" />
 <display:column property="username" />
 <display:column property="fullname" sortable="true" />
 <display:caption>Ez egy Display tag library-s táblázat</display:caption>
</display:table>
```

A display:table name attributuma egy listára mutat, ami egy JavaBean is lehetne, a class egy CSS osztály neve. A display:column tagben a sortable attributum hatására a fejlécre kattintva lehet rendezni az adott oszlop szerint.

Az egyes mezőket úgynevezett dekorátorok segítségével is ki lehet írni, amik megformázhatják az adatot. Lehetőség van a táblázathoz export funkciót is kötni. Lásd a részletes leírásban.

Lásd: `netbeans_ws_ELTE_5`.

1.15 JSF

A JSF (Java Server Faces) egy szabvány, melyet a Sun a szerver oldali felhasználói felületek fejlesztéséhez dolgozott ki, lényegében a JSP továbbfejlesztése oly módon, hogy az oldal hívása előtt egy konfigurációs fájlban definiálhatjuk (egyes fejlesztőkörnyezetekben akár vizuálisan) a lapok közötti átmeneteket (sitemap). Itt kell a szükséges JavaBean-eket is definiálni. Több implementáció is létezik. A Sun Application server például alapértelmezetten tartalmazza a Sun megvalósítását, így majd látható, hogy sehol nem hivatkozunk rá, de a szerverre való feltelepítés után rögtön használható.

1.15.1 Verzió

1.2_12

1.15.2 Website

http://java.sun.com/javaee/javaserverfaces/

https://javaserverfaces.dev.java.net/

1.15.3 Letöltés

https://javaserverfaces.dev.java.net/servlets/ProjectDocumentList?folderID=10411

1.15.4 Library-k

jsf-api.jar, jsf-impl.jar

1.15.5 Konfiguráció

WEB-INF/web.xml, WEB-INF/faces-config.xml

1.15.6 Rövid leírás

A JSF használatára fel kell készíteni a kiszolgálónkat. A `web.xml`-ben be kell állítani, hogy a `jsf` kiterjesztésű fájlokat hogyan dolgozza fel:

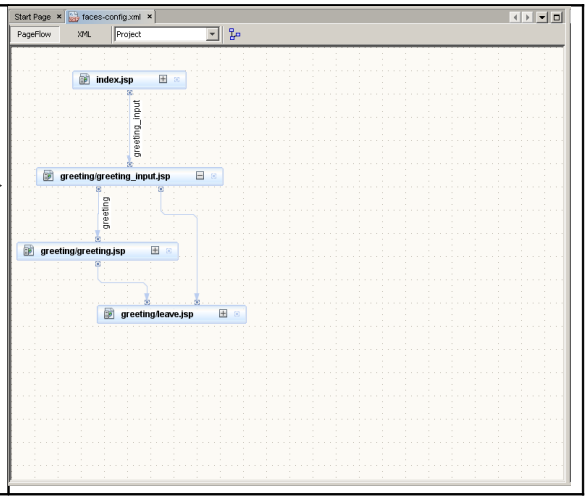
```
<servlet>
 <servlet-name>Faces Servlet</servlet-name>
 <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
 <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
 <servlet-name>Faces Servlet</servlet-name>
 <url-pattern>*.jsf</url-pattern>
</servlet-mapping>
```

Hol van az ehhez szükséges konfigurációs fájl:

```
<context-param>
 <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
 <param-value>server</param-value>
</context-param>
<context-param>
 <param-name>javax.faces.CONFIG_FILES</param-name>
 <param-value>/WEB-INF/faces-config.xml</param-value>
</context-param>
<listener>
 <listener-class>com.sun.faces.config.ConfigureListener</listener-class>
</listener>
```

A `faces-config.xml` fájl fogja tárolni az oldalak közötti átmeneteket, a navigációs szabályokat:

```
<navigation-rule>
 <from-view-id>index.jsp</from-view-id>
 <navigation-case>
  <from-outcome>greeting_input</from-outcome>
  <to-view-id>greeting/greeting_input.jsp</to-view-id>
 </navigation-case>
</navigation-rule>
<navigation-rule>
 <from-view-id>/greeting/greeting_input.jsp</from-view-id>
 <navigation-case>
  <from-outcome>greeting</from-outcome>
  <to-view-id>/greeting/greeting.jsp</to-view-id>
 </navigation-case>
 <navigation-case>
  <to-view-id>/greeting/leave.jsp</to-view-id>
 </navigation-case>
</navigation-rule>
<navigation-rule>
 <from-view-id>/greeting/greeting.jsp</from-view-id>
 <navigation-case>
  <to-view-id>/greeting/leave.jsp</to-view-id>
 </navigation-case>
</navigation-rule>
```



Ahogy látszik, JSP fájlokat kell létrehozni, de a hivatkozások a HTML-oldalakon JSF kiterjesztésekkel szerepel (lásd: `netbeans_ws_ELTE_6/SampleProjekt/src/main/webapp/index.jsp`), így tudja a szerver, hogy a konfigurációs fájl segítségével kell tovább menni a következő JSP lapra. A `from-outcome` node-ban a „*” jel használható, mint helyettesítő karakter.

A többi JSP oldalon már a JSF `core` illetve `html` tag library-jeit is használjuk. Az adatok kezelését JavaBean-ek végzik. A JSP oldalakon a tag-ek attributumában a Bean nevével és azon belül a property nevével tudunk hivatkozni az adatra oldalakon keresztül, hiszen az állapotokat server oldalon mentjük (lásd a web.xml konfigurációjánál fent). A Bean-ek létrehozása automatikusan

történik a faces-config.xml-ben való paraméterezés szerint. De természetesen a MyBean osztálynak léteznie kell, melyben a property mezőnek (mezoNev) létrehozzuk a getter-t (getMezoNev) és a setter-t (setMezoNev) (kézzel, vagy Netbeans-ben a java osztályon, ahol létezik már a mező jobb egérgomb/Refactor/Encapsulate Fields ..., vagy Eclipse-ben jobb egérgomb/Source/Generate Getter and Setter ...).

```
<managed-bean>
 <managed-bean-name>myBean</managed-bean-name>
 <managed-bean-class>hu.elte.rendszer.nev.MyBean</managed-bean-class>
 <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

Figyelni kell, hogy a konfigurációban szereplő néven kell a JSP-kben is az adott osztályra hivatkozni. A kódban így hivatkozunk rá:

```
<h:inputText value="#{myBean.personName}"/>
```

A JSF támogatja az adatok ellenőrzését. A JSP oldalakon attributumok és tag-ek segítségével kell konfigurálni a megszorításokat (`required, f:validateLength`) és az üzenet helyét (`h:messages` tag). További lehetőségeket még a tag library-k dokumentációjában olvashatsz (http://java.sun.com/javaee/javaserverfaces/1.2/docs/tlddocs/index.html).

A **2.0-ás specifikáció** már 2008 közepe óta elérhető, de az implementáció jelenleg még beta verzióban érhető csak el (https://javaserverfaces.dev.java.net/servlets/ProjectDocumentList?folderID=11414).

Léteznek **más implementáció**k is, melyek a Sun szabvánnyal teljesen kompatibilisek, például az Apache Group fejlesztése a MyFaces (http://myfaces.apache.org/). Ezzel később érdemes lehet foglalkozni, mert sok szép és hasznos (AJAX-os, fájlfeltöltés, lapozás, fastruktúra, tabsheet-ek, …) kiegészítő (tag library, framework) komponenst (Projektek: Trinidad, Tobago) fejlesztettek hozzá.

Mivel nem a JSF-et választottuk, mint irány, hanem a Struts-ot (lásd később), illetve az Eclipse nem támogatja, ezért nem foglalkozunk most a Visual JSF-vel.

1.16 Spring

A JSF-ben megismert automatikus bean kezelést általánosítja ez a keretrendszer. Menedzseli az objektumokat, a többi keretrendszert (Velocity, Tiles, Struts, WebWork, Tapestry, Hibernate, TopLink, JDO, iBatis) integrálja, támogatja az AOP-t (Aspect Oriented Programming) és a tesztelést. Működik XML konfigurációs fájlokkal és annotációkkal is.

1.16.1 Verzió

3.0.x, de a 2.5-öt használják elterjedten, ezért mi is ezt használjuk (stabilabb).

2.5.6

1.16.2 Website

http://www.springsource.org/

http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/pdf/spring-framework-reference.pdf

1.16.3 Letöltés

http://www.springsource.com/download?__utma=1.3698475409358766000.1245323593.1246002665.1246002665.11&__utmb=1.5.10.1246002665&__utmc=1&__utmx=-&__utmz=1.1245851731.9.2.utmcsr=struts.apache.org|utmccn=(referral)|utmcmd=referral|utmcct=/2.0.14/docs/struts-2-spring-2-jpa-ajax.html&__utmv=-&__utmk=5690291

Nem lesz rá szükség, mert a következő Struts tartalmazza. ☺

http://www.jarfinder.com/index.php/jars/versionInfo/66289

1.16.4 Library-k

spring-2.5.4.jar (Ez tartalmaz mindent, de lehet modulonként is telepíteni, úgy is elérhetőek a függvénykönyvtárak. Ugyanis ez esetleg tartalmaz fölöslegeseket is (pl: org.springframework.orm.*))

1.16.5 Konfiguráció

A CLASSPATH-ban elérhetővé kell tenni a függvénykönyvtárakat a használatához. A működését lényegében konfigurációs állományok írják le, így erről részletesebben olvashatsz a rövid leírás részben. Az annotációk használatáról a referenciában olvashatsz részletesebben.

1.16.6 Rövid leírás

A Spring keretrendszer nagyon sok mindent tud, de mi csak a modulok (Java bean-ek, osztályok, erőforrások) betöltésére használjuk, ezért erről lesz szó bővebben. Maga a keretrendsezr is moduláris felépítésű, csak ennek a használatával meg lehetne írni egy alkalmazást, de támogatja más keretrendszerek beágyazását, így inkább a jobb, robosztusabb, más keretrendszereket használjuk. Pl: MVC modulja helyett a Struts-ot, ORM modulja helyett a Hibernate plugin-t, validálása helyett az XWork-öt, … (ezeket lásd később.)

A használathoz először be kell állítani a `web.xml`-ben, hogy legyen egy listener és egy filter, ami az adott típusú hívásokat lekezeli. Illetve be kell állítani, hogy hol található a konfigurációs fájl:

```
<context-param>
 <param-name>contextConfigLocation</param-name>
 <param-value>classpath*:applicationContext.xml</param-value>
</context-param>

<filter>
 <filter-name>requestContextFilter</filter-name>
 <filter-class>org.springframework.web.filter.RequestContextFilter</filter-class>
</filter>
<filter-mapping>
 <filter-name>requestContextFilter</filter-name>
 <url-pattern>/*</url-pattern>
</filter-mapping>

<listener>
 <listener-class>org.springframework.web.context.ContextLoaderListener</listener-
class>
</listener>
```

Ezután már a deployoláskor hiányzik az `applicationContext.xml` fájl, ha nem hoztuk még létre. Ez kell, hogy tartalmazza a használandó bean-ek nevét, osztályát és egyéb beállításokat.
Az első bean definíciója a legegyszerűbb:

```
<bean id="mss" class="hu.elte.rendszer.nev.MySecondSample"  />
```

Ezt a JSP fájlunkban úgy tudjuk használni, hogy a web-alkalmazásunkhoz tartozó környezetből elkérjük az adott bean-t. Először a környezetet kell elérni (ezt csak egyszer, a többi bean-nél nyilván már nem), majd a bean-t az id segítségével érjük el, és már használhatjuk is. Nem kell és nem is szabad `new`-val létrehozni.

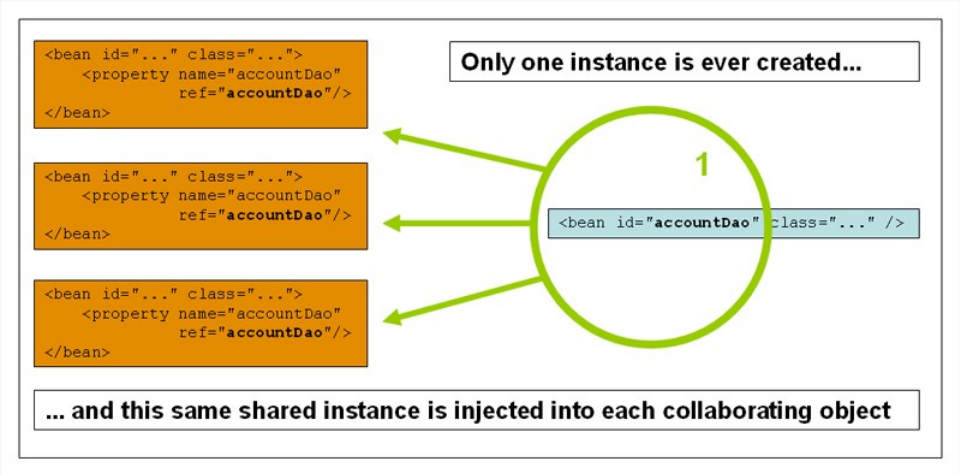
```
ServletContext context = this.getServletContext();
WebApplicationContext applicationContext =
WebApplicationContextUtils.getWebApplicationContext(context);

MySecondSample mss = (MySecondSample) applicationContext.getBean("mss");
out.print(mss.getValue());
```

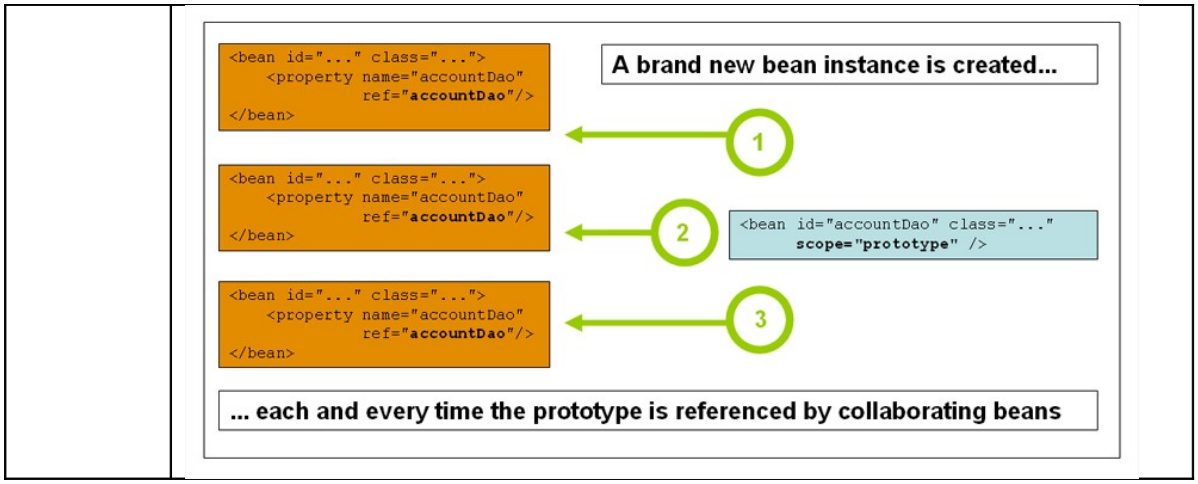
Ez a bean automatikusan létrejön és az alkalmazásunkban használhatjuk. A létrehozáskor a konstruktor fut le, aminek a paramétereit a `<constructor-arg>` node-ban adhatjuk meg, amiből több is lehet egymás után, és a sorrend számít. Ezen kívül még beállíthatjuk az osztály property-jeit a `<property name="myName" value="myValue" />` node-okkal (lehet több), ahol a `name` definiálja, hogy melyik setter hívódjon meg az adott értékkel (`setMyName(myValue);`)

```
<bean id="mcserver" class="hu.elte.rendszer.nev.MyCounter" scope="singleton">
 <constructor-arg><value>1</value></constructor-arg>
 <property name="step" value="1" />
</bean>
<bean id="mcsession" class="hu.elte.rendszer.nev.MyCounter" scope="session">
 <constructor-arg><value>1</value></constructor-arg>
 <property name="step" value="2" />
</bean>
<bean id="mcfelhasznalas" class="hu.elte.rendszer.nev.MyCounter" scope="prototype">
 <constructor-arg><value>1</value></constructor-arg>
 <property name="step" value="3" />
</bean>
```

Tehát itt beállítottunk ugyanabból az osztályból három számlálót, ahol mindegyik 1-vel kezdi a számolást (konstruktor) és a lépésköz rendre 1, 2 illetve 3.

A lépés nagyságán kívül, mindháromnál van még egy különbség, a `scope`. Ezek jelentése a következő:

singleton	Alapértelmezett érték. Egy létezik belőle a szerveren. Indításkor/újraindításkor jön létre. 
session	Egy munkafolyamathoz jön létre belőle egy. Böngészőnként, kliensenként egyedi adott munka alatt.
prototype	Amikor hivatkozunk rá, akkor jön létre, és ha már nem használjuk, felszabadításra kerül.



1.17 Struts2 (XWork/WebWork, FreeMaker, OGNL)

A Struts2 egy ingyenes, nyílt forráskódú keretrendszer, mely Java-s webes alkalmazások készítésére fejlesztenek. A Struts2 a Struts előző verziójának és a WebWork összeházasításával jött létre, de több egyébb függvénykönyvtárra is szüksége van a működéshez. Ilyenek a FreeMaker template engine, ami jó alternatívája a JSP-nek, vagy az OGNL, ami egy olyan kifejezés és kötésdefiníciós nyelv, mely a Java objektumok tulajdonságait éri el a getter/setter-ek segítségével.

1.17.1 Verzió

2.1.6

1.17.2 Website

http://struts.apache.org/index.html

http://freemarker.sourceforge.net/

http://www.opensymphony.com/ognl/html/LanguageGuide/introduction.html

http://struts.apache.org/2.x/docs/interceptors.html

1.17.3 Letöltés

http://struts.apache.org/download.cgi#struts216

1.17.4 Library-k

struts2-codebehind-plugin-2.1.6.jar, struts2-config-browser-plugin-2.1.6.jar, struts2-convention-plugin-2.1.6.jar, struts2-core-2.1.6.jar, struts2-dojo-plugin-2.1.6.jar, struts2-dwr-plugin-2.1.6.jar, struts2-jasperreports-plugin-2.1.6.jar, struts2-javatemplates-plugin-2.1.6.jar, struts2-jfreechart-plugin-2.1.6.jar, struts2-jsf-plugin-2.1.6.jar, struts2-junit-plugin-2.1.6.jar, struts2-pell-multipart-plugin-2.1.6.jar, struts2-plexus-plugin-2.1.6.jar, struts2-portlet-plugin-2.1.6.jar, struts2-rest-plugin-2.1.6.jar, struts2-sitegraph-plugin-2.1.6.jar, struts2-sitemesh-plugin-2.1.6.jar, struts2-spring-plugin-2.1.6.jar, struts2-struts1-plugin-2.1.6.jar, struts2-testng-plugin-2.1.6.jar, struts2-tiles-plugin-2.1.6.jar

1.17.5 Konfiguráció

A legkevesebb függvénykönyvtár, amit fel kell használni, a következők:

commons-fileupload-1.2.1.jar, commons-logging-1.0.4.jar, freemaker-2.3.13.jar, ognl-2.6.11.jar, struts2-core-2.1.6.jar, xwork-2.1.2.jar

Ezeket a CLASSPATH-ba (lib) kell elhelyezni.

A WEB-INF könyvtárba kell másolni a struts-tags.tld fájlt, ami a tag library hivatkozásokat tartalmazza.

A web.xml-be el kell helyezni az előzőre való hivatkozást:

```
<jsp-config>
 <taglib>
  <taglib-uri>/s</taglib-uri>
  <taglib-location>/WEB-INF/struts-tags.tld</taglib-location>
 </taglib>
</jsp-config>
```

Ezen kívül itt kell egy filtert is definiálni:

```
<filter>
 <filter-name>struts2</filter-name>
 <filter-
class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter
-class>
</filter>
<filter-mapping>
 <filter-name>struts2</filter-name>
 <url-pattern>/*</url-pattern>
</filter-mapping>
```

A többi beállítás az alkalmazáshoz tartozik és nem a szerverhez, ezért a következő alfejezetben található.

1.17.6 Rövid leírás

A classes könyvtárba kerülő resources-ek közé létre kell hozni a struts.properties fájlt. Ez tartalmazza az alkalmazásban szereplő saját kiterjesztést (myext):

```
struts.action.extension=myext
```

Az alapértelmezett helyi beállításokat (hely, karakterkódolás):

```
struts.locale=hu_HU
struts.i18n.encoding=UTF-8
```

Ezeken kívül még több beállítási lehetőség is van, ezeket most nem részletezzük.

A következő fontos fájl a struts.xml ugyanott, ami körülbelül a JSF-ben lévő faces-config.xml-nek felel meg. Itt írjuk le, hogy honnan hova menjen a hívás. A nagy előnye a Strutsnak, hogy nem azt kell megmondani, hogy melyik JSP után melyik másik JSP jön (JSP helyett lehet FreeMaker templatet is hívni, de ekkor a result node type="freemaker" attributumát meg kell adni, sőt a Velocity template engine-t is támogatja a Struts.). Hanem egy absztrakt kérést, úgynevezett action-t definiálunk, aminek az érkezésekor előbb egy Java osztály fut le, és az eredménytől függően több JSP is lehet a válasz. A Java osztály megadása történhet közvetlenül, vagy a Spring plugin-nak köszönhetően egy egyedi névvel is lehet rá hivatkozni, amivel a Spring konfigurációjában definiáltuk a bean-t. Az előre definiált konstansok segítségével történik az elágazás, de mi is vehetünk fel új konstansokat. Ezek jól be is mutatják a lényeget, vagyis azt, hogy elképzelhető egy kérés érkezésekor, hogy még hibásan van kitöltve a kérdőív (input), ezért vissza

kell dobnunk az előző oldalt, javításra, és nem mehetünk tovább a feldolgozásra (`success`). A Java osztályunkban lévő property-ket getter/setter-ek segítségével éri el a Struts a JSP-k feldolgozásakor. A konstansokat az ActionSupport (xwork2) osztályból történő leszármaztatással tudjuk legegyszerűbben elérni.

A `struts` root-on belül `package`-ket hozunk létre, melyek egymást terjeszthetik ki. Van nekik nevük és használhatnak névtereket (`namespace`), ami az URL-ben jelenik meg, mint egy könyvtár, így tudjuk az összetartozó oldalakat átláthatóan strukturálni.

Tehát egy `action` leírásakor meg kell adnunk a nevét (lehet helyettesítő karaktereket is használni, pl.: *), hogy melyik osztály fusson le, és megadhatjuk, hogy melyik eljárást hívja meg (alapértelmezetten az `execute`). Az `action`-on belül a `result` node-ok definiálják (`name` attribútum nélkül a `success` az alapértelmezett), hogy milyen válaszra, melyik JSP-t adja vissza.

A JSP fájlokat tehát a struts hívja, az index.jsp kivételével. Ebben azt érdemes megfigyelni, hogy rögtön továbbadja a kérést az első oldalra egy URL segítségével (lásd namespace/"könyvtár" és kiterjesztés):

```
<META HTTP-EQUIV="Refresh" CONTENT="0;URL=greeting/startGreeting.myext">
```

A többi JSP-ben a Struts saját tag library-jét használhatjuk `s` perfix-vel, ezt másoltuk be a `WEB-INF`-be.

```
<%@taglib prefix="s" uri="/struts-tags" %>
```

Ennek segítségével az oldalakon (`greeting_input.jsp, greeting.jsp, leave.jsp`) automatikusan létrejönnek a form-ok, egyéb mezők és kontrol objektumok. A `s:submit`-ban az `action` attribútum azonosítja a `struts.xml`-ben lévő `action`-t. A Java osztályban lévő mezők megjelenítéséhez többféle tag is ad lehetőséget. Ezeket lásd a részletes leírásban, a honlapon a referenciában. Azért megjegyezném, hogy figyelni kell, mert egyikben a `name`, másikban a `value` nevű attribútumban kell megadni a mező nevét!

A kvázi HTML oldalba szöveget is írhatunk, de ha igazán szép, többnyelvű, nemzetközi alkalmazást szeretnénk írni, akkor a key attribútumok kitöltésével hivatkozunk a szövegekre, melyeket Java osztály neve, plusz a locale értéke, plusz a „.properties" kiterjesztéssel (`MyGreeting_hu_HU.properties` és `MyGreeting_en_US.properties`) rendelkező szótár fájlba kell helyeznünk.

Nálunk alapértelmezetten a böngészők a `hu_HU` értéket küldik, de ezt felülírhatjuk, ha küldjük a `request_locale` paramétert a kérésben. Ekkor a megfelelő szótárból kerülnek kiírásra a szövegek.

Ezt a váltást egy template (`nyelv.ftl`) segítségével is feltehetjük az összes oldalra, hogy ne kelljen mindenhol lekódolnunk, hogy legyen egy link, ahol csak a másik nyelvet lehessen választani.

Az `s:component` segítségével külső fájlokból tudunk behívni templateket, melyekhez a FreeMaker alapból adott, tehát az ftl (FreeMaker Template Language) fájlokkal tudjuk megírni, melynek paramétereket is lehet átadni. A FreeMaker nyelvi elemeit a weboldalán található referenciában találsz.

A Struts bevezette a téma (theme) tulajdonságot, mely alapértelmezetten xhtml értékű, de van simple, és ajax is. Ezzel is lehet strukturálni (különböző könyvtárszerkezetekben) az alkalmazásunkat.

A Java osztályunkban történő elágazások lefejlesztésén kívül lehetőség van az XWork segítségével a validálás, érvényesítés elvégzésére és így az input-ra történő visszadobás konfigurálására. Ezt a classes-ban a Java osztállyal egy helyre kell tenni „-validation” prefix-vel és xml kiterjesztéssel (`MyGreeting-validation.xml`). A fejlesztéskor a resources-ben lévő ugyanolyan Java package struktúrába kell helyezni a fájl, ami így a megfelelő classes alkönyvtárba kerül.

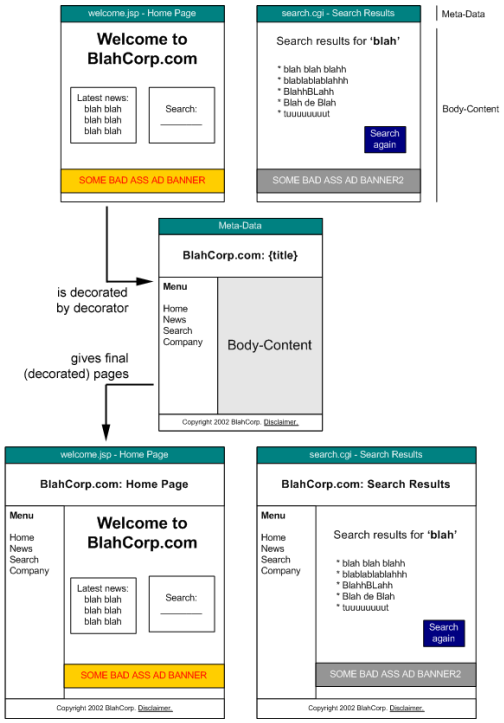
A `validators` root-ban minden egyes `field`-hez `field-validator`-okban kell leírni az érvényesítés típusát, paramétereit és a hiba esetén kiírandó hibaüzenetet.

A Struts lehetőséget ad még úgynevezett interceptorok írására, melyeket több alapinterceptorból lehet leszármaztatni és a `struts.xml`-ben kell definiálni. Ezek az osztályok (logolás, előfeldolgozás, érvényesítés, szabálykezelés, ...) (`MyGreetingInterceptor.java`) az ősosztálytól függően jól definiált időben futnak le az action osztály előtt és/vagy után, sőt az action osztály-t is el tudja érni. Csak azoknál az action osztályoknál fut le abban a sorrendben, ahol és ahogy a `struts.xml`-ben konfiguráljuk (`<interceptor-ref name="defaultMyStack"/>`). Az interceptor osztályokat egy memóriacsomagba kell helyezni, hogy együtt legyen kezelve az osztállyal és ne írja felül az adatokat az esetleges különböző futásoknál.

Ezt kipróbálhatod a netbeans_ws_ELTE_8-ben.

1.18 Sitemesh

A Sitemesh egy weblap megjelenítő és dekorációs keretrendszer, melynek segítségével könnyen lehet webes alkalmazásokat integrálni nagy site-okba, megtartva a könnyű navigációt és konzisztens kinézetet.



1.18.1 Verzió
2.4.1

1.18.2 Website
http://www.opensymphony.com/sitemesh/

1.18.3 Letöltés
http://www.opensymphony.com/sitemesh/download.action

1.18.4 Library-k
sitemesh-2.4.1.jar

1.18.5 Konfiguráció
A függvénykönyvtárat a WEB-INF/lib könyvtárba kell másolni és be kell írni a CLASSPATH-ba.

Módosítani kell a web.xml fájlt egy filter hozzáadásával:

```
    <filter>
        <filter-name>sitemesh</filter-name>
        <filter-
class>com.opensymphony.sitemesh.webapp.SiteMeshFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>sitemesh</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

1.18.6 Rövid leírás
Az alkalmazás konfigurálásához létre kell hozni egy sitemesh.xml fájlt a WEB-INF könyvtárba, a következő tartalommal:

```
<sitemesh>
 <property name="decorators-file" value="/WEB-INF/decorators.xml" />
 <excludes file="${decorators-file}" />
 <page-parsers>
  <parser content-type="text/html"
class="com.opensymphony.module.sitemesh.parser.HTMLPageParser" />
  <parser content-type="text/html;charset=ISO-8859-1"
class="com.opensymphony.module.sitemesh.parser.HTMLPageParser" />
 </page-parsers>
 <decorator-mappers>
  <mapper
class="com.opensymphony.module.sitemesh.mapper.ConfigDecoratorMapper">
   <param name="config" value="${decorators-file}" />
  </mapper>
 </decorator-mappers>
</sitemesh>
```

Ugyanide kell létrehozni a decorators.xml konfigurációs fájlt is, amire hivatkozik az előző:

```
<decorators defaultdir="/decorators">
    <decorator name="main" page="main.jsp">
          <pattern>/*</pattern>
    </decorator>
</decorators>
```

Ebben írjuk le, hogy mely hívásokhoz kell alkalmazni a dekorátort és az melyik könyvtárban található melyik fájlt használja fel.

A dekorátor lesz tehát a kérés alapja, kerete, melybe az adott hívás eredménye a megfelelő helyre töltődik be. Ezeket a helyeket a Sitemesh tag library-jében található azonosítókkal jelöljük meg. Például `<decorator:body/>`, ahova a válasz törzse töltődik be. Természetesen a JSP include-ot is használhatjuk a még jobb modularizáláshoz.
Támogatja a JSP helyett a Freemaker és Velocity template engine-ket is dekorátornak.
A Struts2 tartalmaz Sitemesh plugint, így Struts tag-eket is lehet a dekorátorban használni.
Ezt kipróbálhatod a netbeans_ws_ELTE_9-ben.

1.19 Tiles

Ez egy ugyanolyan template keretrendszer, mint a Sitemesh.

1.19.1 Verzió

2.1.2

1.19.2 Website

http://tiles.apache.org/index.html

1.19.3 Letöltés

http://tiles.apache.org/download.html

1.19.4 Library-k

tiles-api-2.1.2.jar, tiles-core-2.1.2.jar, tiles-jsp-2.1.2.jar, tiles-servlet-2.1.2.jar

1.19.5 Konfiguráció

web.xml

tiles-defs.xml

1.19.6 Rövid leírás

Jellemzően csak Spring-vel és Struts-val együtt használják. Mi a Sitemesh-t választottuk helyette, mert egyszerűbb a tagek definiálása.

1.20 Spring Security (Acegi)

Spring Security néven fut tovább a korábbi Acegi. Ennek a keretrendszernek a segítségével könnyen biztonságossá tehetjük az egész alkalmazásunkat. Támogatja a bejelentkezést, a web URL-ek alapján tudja kezelni az alkalmazás részeit és kezeli az egyes eljárásokhoz való hozzáférést is. Feltételezi a Spring keretrendszer használatát.

1.20.1 Verzió

2.0.4

1.20.2 Website

http://static.springsource.org/spring-security/site/index.html

1.20.3 Letöltés

http://static.springsource.org/spring-security/site/downloads.html

1.20.4 Library-k

spring-security-acl-2.0.4.jar, spring-security-cas-client-2.0.4.jar, spring-security-catalina-2.0.4.jar, spring-security-core-2.0.4.jar, spring-security-core-tiger-2.0.4.jar, spring-security-jboss-2.0.4.jar, spring-security-jetty-2.0.4.jar, spring-security-ntlm-2.0.4.jar, spring-security-openid-2.0.4.jar, spring-security-portlet-2.0.4.jar, spring-security-resin-2.0.4.jar, spring-security-taglibs-2.0.4.jar

1.20.5 Konfiguráció

A konfigurálás a web.xml és a Spring-nél megismert applicationContext.xml segítségével történik. Be kell állítani az CLASSPATH-ba a szükséges függvénykönyvtárakat és a web.xml-be fel kell venni egy új filtert:

```
<filter>
 <filter-name>springSecurityFilterChain</filter-name>
 <filter-
class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
 <filter-name>springSecurityFilterChain</filter-name>
 <url-pattern>/*</url-pattern>
</filter-mapping>
```

Szükség lehet egy listenerre is:

```
<listener>
 <listener-
class>org.springframework.security.ui.session.HttpSessionEventPublisher</li
stener-class>
</listener>
```

A rövid leírásban több megoldást is végignézünk, amikhez más és más konfiguráció tartozik, ezért bővebben ott lesz leírva.

1.20.6 Rövid leírás

Először egy gyors áttekintést nézünk meg, majd a következő fejezet után egy komolyabb biztonsági beállítást is megvizsgálunk.

A példánkhoz a következő függvénykönyvtárakra van szükség: `aspectjrt.jar, commons-logging-1.0.4.jar, spring.jar, spring-security-core-2.0.4.jar, spring-security-core-tiger-2.0.4.jar, spring-security-taglibs-2.0.4.jar`.

A `web.xml`-be még a Spring előző fejezetében megismert filterét és listener-ét is fel kell vennünk.

Ehhez a `web.xml`-ben be kell állítani a tag library-t és a fájlt oda is kell másolni:

```
<jsp-config>
 <taglib>
  <taglib-uri>http://www.springframework.org/security/tags</taglib-uri>
  <taglib-location>/WEB-INF/security.tld</taglib-location>
 </taglib>
</jsp-config>
```

Az applicationContext.xml-ben kell definiálni a felhasználókat és az ő jogosultságaikat:

```
<security:authentication-provider>
 <!--security:password-encoder hash="md5"/-->
 <security:user-service>
  <security:user name="admin" password="adminadmin"
authorities="ROLE_ADMIN, ROLE_USER, ROLE_MANAGER" />
```

```
  <security:user name="manager" password="manager"
authorities="ROLE_USER,ROLE_MANAGER" />
  <security:user name="user" password="user" authorities="ROLE_USER"
/>
 </security:user-service>
</security:authentication-provider>
```

Ugyanitt definiáljuk az URL-eket és a jogosultságokat is:

```
<security:http auto-config="true">
 <security:intercept-url pattern="/admin/**" access="ROLE_ADMIN"/>
 <security:intercept-url pattern="/manager/**" access="ROLE_MANAGER"/>
 <security:intercept-url pattern="/user/**"
access="IS_AUTHENTICATED_REMEMBERED" />
 <security:intercept-url pattern="/**"
access="IS_AUTHENTICATED_ANONYMOUSLY" />
</security:http>
```

Ez az alapértelmezett URL szűrést be is konfiguráltuk. Az alkönyvtárak fájljait, csak azok a felhasználók tudják megtekinteni, akiknek megfelelő jogosultságaik vannak beállítva. A bejelentkeztetés JSP oldalai is alapértelmezetten, a Spring segítségével megjelennek.

Az alkönyvtárakhoz történő jogosultságok kezelésén kívül, a tag library segítségével lehetőség van a saját JSP oldalainkon belül a tartalmat is szűrni. Használni kell a `<%@ taglib prefix='security' uri='http://www.springframework.org/security/tags' %>` deklarációt, majd a `<security:authorize> ... </security:authorize>` közötti tartalom, csak akkor jelenik meg, ha az attributumban definiált feltétel teljesül. Használhatjuk a következő attributumokat, melyek értékének a jogosultságokat kell vesszővel felsorolni: `ifAllGranted, ifAnyGranted, ifNotGranted.`

A felhasználó nevét pedig a `<security:authentication property="principal.username"/>` tag-gel tudjuk kiíratni.

Az eddigieket kipróbálhatod a netbeans_ws_ELTE_10-ben.

Lehetőség van Basic HTTP Authentikációra is. A HTTPS, password erősség, egyéb saját osztály (pl egyszer használatos PIN ellenőrzés) illetve a felhasználók és jogosultságaik adatbázisban történő tárolására egy másik fejezetben bővebben kitérünk. Lásd a Példa projektek című fejezet 4.XX netbeans_ws_ELTE_XX című alfejezetében.

1.21 Hibernate

A Hibernate egy nagy teljesítményű objektum relációs modell/mapping implementáció, mely segítségével objektum-orientált perzisztens osztályok fejleszthetők, amikkel és egy saját hordozható SQL kiterjesztéssel (HQL) könnyen lehet az adatokat kezelni, hiszen támogatja szinte az összes adatbázis-kezelő rendszert. Így adatbázis táblák helyett Java osztályokkal dolgozhatunk. XML-lel és annotációkkal lehet konfigurálni. A könnyű használatát az IDE-k támogatják az adatbázisból történő kód generálásával.

1.21.1 Verzió

3.3.2

1.21.2 Website

https://www.hibernate.org/

https://www.hibernate.org/152.html

http://docs.jboss.org/hibernate/stable/core/reference/en/html/

1.21.3 Letöltés

https://www.hibernate.org/6.html

1.21.4 Library-k

hibernate3.jar

1.21.5 Konfiguráció

A library-ket a classpath-ban kell elhelyezni. A többi konfigurálás az alkalmazáshoz tartozik, így a következő fejezetben ezekről még írunk.

1.21.6 Rövid leírás

A hibernate.cfg.xml konfigurációs fájlban tudjuk beállítani, hogy az alkalmazásunk melyik adatbázishoz kapcsolódjon és hogy mely mapping fájlok tartoznak ehhez a session-höz.

```
<hibernate-configuration>
  <session-factory>
      <property
name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</p
roperty>
      <property name="hibernate.connection.password">sample</property>
            <property
name="hibernate.connection.username">sample</property>
            <property
name="hibernate.connection.url">jdbc:oracle:thin:@IP:port:SID</property>
    <property name="dialect">org.hibernate.dialect.OracleDialect</property>
    <property
name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>
    <property name="show_sql">true</property>
    <mapping resource="Users.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

Natív elérés helyett, JNDI segítségével is megadható az adatforrás (datasource). A mapping fájlok, további konfigurációs állományok, melyek leírják, hogy mely View vagy Table mezői melyik java osztály property-jeinek feleljenek meg.

```
<hibernate-mapping>
  <class dynamic-insert="false" dynamic-update="false" mutable="true"
name="hu.elte.rendszer.nev.User" optimistic-lock="version"
polymorphism="implicit" select-before-update="false" table="USERS">
    <id column="U_ID" name="u_id">
      <generator class="native"/>
    </id>
    <property column="USERNAME" name="username"/>
...
  </class>
</hibernate-mapping>
```

Ezek az osztályok olyan egyszerű osztályok, melyek csak a megfelelő típusú property-ket és azok getter/setter-eit tartalmazzák. (POJO, Plain Old Java Object)

```
package hu.elte.rendszer.nev;
public class User {
    private Long u_id;
    private String username;
    public User() {}
```

```
    public Long getU_id() {
        return u_id;
    }
    public void setU_id(Long u_id) {
        this.u_id = u_id;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
...
}
```

Az előző fájlok elkészítését, legenerálását az IDE-khez beszerezhető pluginok támogatják. Java-ból csak osztályokat látunk és minden SQL utasítás el van rejtve a Java fejlesztő elöl. Lehetőség van az adatok lekérdezésére, beszúrására, módosítására és törlésére. Ehhez Java-ban csak a session-t kell megnyitni és használni a load illetve save parancsokat és az tábláknak megfelelő osztályokat.

```
session.beginTransaction();
User u = (User) session.load(User.class, Long.parseLong("4"));
u.setPasswd(u.getPasswd()+String.valueOf(u.getPasswd().length()+1));
session.save(u);
session.getTransaction().commit();
```

A lekérdezéseknél filterek segítségével tudjuk megadni a kritériumokat, melyek a list utasításkor visszaadják az osztályelemeket tartalmazó sorozatot, melyen egy iterator-ral végig tudunk menni.

```
Criteria c = session.createCriteria(User.class);
c.add((Criterion) Expression.le("u_id", Long.parseLong("5")));
c.add((Criterion) Expression.ge("u_id", Long.parseLong("0")));
Iterator i = c.list().iterator();
while (i.hasNext()) {
  System.out.println("username = " + ((User) i.next()).getUsername());
}
```

Sajnos tárolt függvény vagy eljárás meghívását ez a keretrendszer nem teljesen támogatja. lehetőség van a közvetlen adatbázis kapcsolatot lekérni, és azon keresztül már a szabványos java.sql.* osztályokon keresztül hozzáférünk mindenhez, de ez deprecated.

```
Connection con = session.~~connection~~();
try {
 CallableStatement cs = con.prepareCall("{ call
createnew('a','b','c','d')}");
 ResultSet rs = cs.executeQuery();
} catch (SQLException ex) {
 ex.printStackTrace();
}
```

A keretrendszer pedig csak az olyan tárolt függvényeket támogatja, ahol a visszatérési érték egy SYS_REFCURSOR-on visszaadott SELECT eredménye. Erre az a megoldás lehet (workaround), hogy a tényleges tárol eljárás köré építünk egy olyan tárolt függvényt, mely sys_refcursor-ban adja vissza a tárolt eljárás OUT paramétereit. Vagy közvetlenül olyan tárolt függvényt írunk, amiben nincsenek kimenő paraméterek, csak bemenők és az eredményt egy virtuális select adja vissza. (`SELECT 'SUCCESS' AS result FROM dual;`) Ehhez egy nevesített sql lekérdezést és egy eredményosztályt kell konfigurálnunk a hibernate-ben:

```
<hibernate-mapping>
    <class name="hu.elte.rendszer.nev.Result">
        <id name="result" />
        <property column="RESULT" name="result"/>
    </class>
    <sql-query name="newUser" callable="true">
        <return class="hu.elte.rendszer.nev.Result">
            <return-property name="result" column="RESULT"/>
        </return>
    { ? = call createNewUser(:user,:pass,:full,:notes) }
    </sql-query>
</hibernate-mapping>
```

Így könnyen tudjuk a tárolt függvényt egy tranzakcióban kezelni a többi adatbázis-művelettel együtt.

```
session.beginTransaction();

 User u = (User) session.load(User.class, Long.parseLong("4"));
 u.setPasswd(u.getPasswd() + String.valueOf(u.getPasswd().length() + 1));
 session.save(u);

 Iterator i = session.getNamedQuery("newUser").setString("user",
"testuser").setString("pass", "testuserpass").setString("full", "Test
User").setString("notes", "Test Notes").list().iterator();
 if (i.hasNext()) {
  System.out.println("EREDMENY = " + ((Result) i.next()).getResult());
 }

//session.getTransaction().rollback();
session.getTransaction().commit();
```

Ezeket az adatelérő objektumok generálását (Data Access Object, DAO) több segéd alkalmazás, plugin támogatja, melyeket esetleg többször kell alkalmazni, ezért az a javasolt, hogy a generált osztályokat (Base<tábla>.java) leszármaztatva hozzuk létre a saját esetleg testreszabott osztályainkat (<tábla>.java), így az újragenerálás nem fogja felülírni a módosításainkat.

Példa projektek

Az IDE-k konfigurálásától kezdve példa projekteken mutattam be a megvalósítási lehetőséget. Most ismét összeszedem, hogy az egyes verziók mit tartalmaznak.

1.22 <IDE>_ws_ELTE_1

Ebben a tömörített mappában két projekt található egy SampleCommonProjekt nevű és egy SampleProjekt. A SampleCommonProjekt csak egy Sample osztályt tartalmaz, amiben egy getValue függvény visszaadja a „TestValue” értéket.

Ebből az osztályból származik le a SampleProjekt-nek a MySample osztálya, melyben felüldefiniáljuk a getValue-t úgy, hogy az ős értékéhez még hozzáfűzzük a „ & MyTestValue” értéket is. Ez a Web-es alkalmazás egy JSP fájlból áll, mely meghívja a MySample osztály getValue függvényét, melynek eredményét kiírja az oldalra.

Be kellett konfigurálni a webszervert, az alkalmazásban pedig a külső függvényosztályt, a főprojekt használja a common projektet.

1) Tanulmányozd át a csomagot!

2) Futtasd az alkalmazást!

1.23 netbeans_ws_ELTE_2

A jUnit használatára találhatsz példakódokat az src/test/java/hu/elte/rendszer/nev mappában. A MySampleTest.java az org.junit package-t használja, míg a mySampletest2.java a junit.framework package-t. A kommentezet kódrészleteket visszaírva láthatsz hibás teszteseteket is, amik azért vannak most kikommentezve, hogy a Maven ne fusson hibára a fordításkor.

1) Teszteld le a csomagot Netbeansben!

2) Teszteld le a csomagot Maven segítségével!

1.24 netbeans_ws_ELTE_3

A log4j kerül itt bevezetésre. Felvettünk a log4j függvénykönyvtárat, mint szükséges library (dependency) a Maven miatt a pom.xml-be, a Netbeans-ben pedig a fejlesztés/fordítás/futtatás miatt pedig a SampleProjekt/Libraries-nél „Add JAR/Folder …”-val az nbproject/build-impl.xml, nbproject/project.propoerties és nbproject/project.xml fájlokba (ide soha nem írunk kézzel!).

A $PROJECTDIR/src/main/resources könyvtárat létrehoztuk; beállítottuk, mint forrás könyvtár; és ide létrehoztuk a log4j konfigurációs fájljait.

1) Próbáld ki! Hibát kapunk:

```
log4j:WARN No appenders could be found for logger
(hu.elte.rendszer.nev.MySample).
log4j:WARN Please initialize the log4j system properly.
```

Nem logol, mert hiányzik a konfigurációs fájl.

2) Nevezd át az `X_log4j.properties` fájlt az `X_` prefix kitörlésével `log4j.properties`–re és deployold újra. Így készül log a konzolra.
3) Tanulmányozd át a konfigurációs fájlt!
4) Most ezt nevezd vissza, és az `Y_` prefixet töröld és deployold újra. Most is logol, de más formában.
5) Vizsgáld meg konfigurációs fájlok közötti különbségeket!
6) Máshol is készült log? (Létezzen a `C:\temp` könyvtár, vagy írd át a `log4j.xml` fájlban a logfájl helyét!)
7) Milyen különbséget látsz még?
8) Miért került csak kevesebb sor az egyik fájlba?

1.25 netbeans_ws_ELTE_4

A JSP és JSTL megismerésére van ebben a csomagban lehetőség.

1.26 netbeans_ws_ELTE_5

Ebben a csomagban a Display tag library használata látható.

1.27 netbeans_ws_ELTE_6

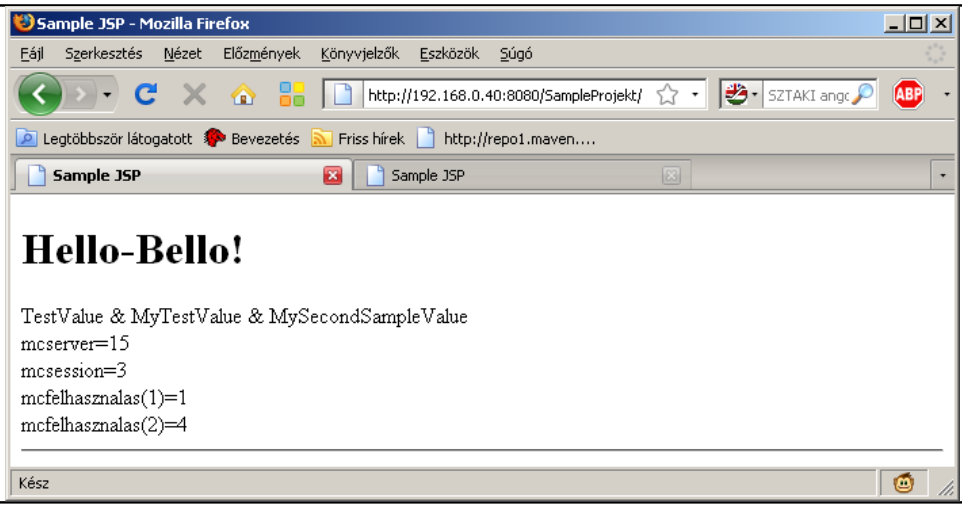
Újdonságként a JSF használata kerül bele a projektbe. Elég részletesen leírásra került a JSF-es fejezetben. Ott nem került megemlítésre, de még újdonság a `messages.properties` fájl használata, mely a felületen megjelenő szövegeket tartalmazza `név=érték` formában. Ennek a **szótár fájl**nak később a többnyelvű, nemzetközi projekteknél lesz jelentősége.

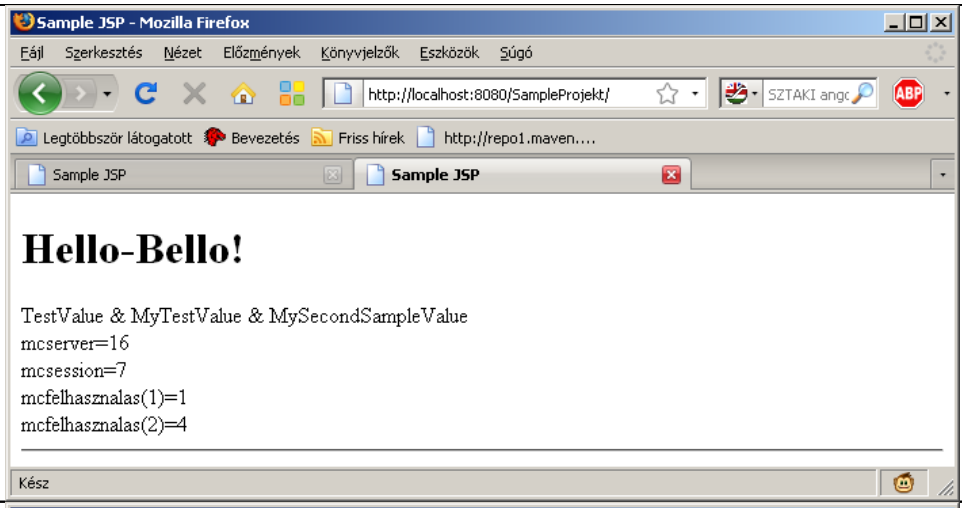
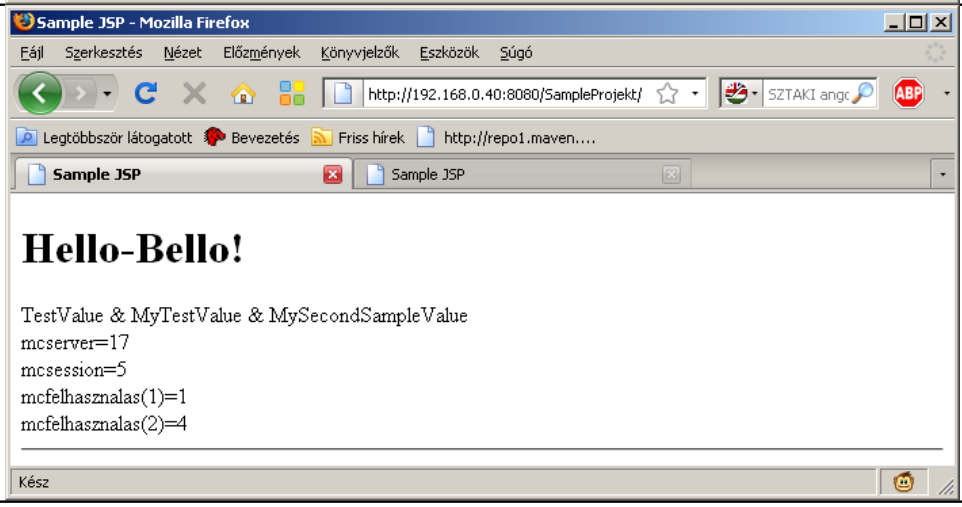
1) Hasonlítsd össze az előző csomaggal a fájlokat!
2) Mely fájlok változtak? Hogyan?
3) Milyen új fájlok jöttek létre?
4) Hogyan kerülnek kijelzésre a hibás adatok?

1.28 netbeans_ws_ELTE_7

A Spring bemutatása.

1. Változott-e a web.xml?
2. Hol található az applicationContext.xml?
3. Milyen különbségek vannak az applicationContext.xml-ben található négy bean definíciójában?
4. Hogyan hivatkozunk a bean-ekre a forrásból?
5. Teszteld le két, külön címmel hívva a kis alkalmazásban a session scope-ot is! (http://localhost:8080/SampleProjekt/ és http://192.168.0.40:8080/SampleProjekt/ (értelemszerűen a saját IP-ddel ☺))

1. hívás	

2. hívás	
3. hívás	

1.29 netbeans_ws_ELTE_8

A Struts2 alap lehetőségeinek bemutatása.

1.30 netbeans_ws_ELTE_9

A Sitemesh dekorátor megismerése.

1.31 netbeans_ws_ELTE_10

A Spring Security alapértelmezett beállítása.

1.32 netbeans_ws_ELTE_11

A Hibernate kipróbálása.

Egyéb hasznos linkek

- Java EE 5 Tutorial - http://java.sun.com/javaee/5/docs/tutorial/doc/index.html
- jarFinder - http://www.jarfinder.com/
- Spring - http://opensource.atlassian.com/confluence/spring/dashboard.action

-

TODO

- jBoss
- MyFaces
- AJAX
- JavaScript keretrendszerek
- JDO
- TopLink
- JasperReports http://jasperforge.org/projects/jasperreports
- TestNG http://testng.org/doc/documentation-main.html
- Graphviz http://www.graphviz.org/
- DWR http://directwebremoting.org/dwr/index.html
- Portlet és Struts portlet plugin http://struts.apache.org/2.x/docs/portlet-plugin.html
- OVal http://oval.sourceforge.net/
- JFreeChart http://www.jfree.org/jfreechart/
- Meld http://meld.sourceforge.net/
- http://agile.csc.ncsu.edu/SEMaterials/tutorials/fit/
- http://fit.c2.com/wiki.cgi?FitTools