



# WebStore #maven

JAX-WS SOAP WS, Stateful Session Bean, PMD, JavaDoc

Óbudai Egyetem, Java Enterprise Edition

Műszaki Informatika szak

Labor 8

Bedők Dávid  
2018-03-03  
v1.0

# SOAP webszolgáltatások

- ▷ 1998, 2000 (v1.1), 2003 (v1.2 W3C ajánlással)
- ▷ **Simple Object Access Protocol (SOAP)**, de 1.2 után "nem használjuk kibontva" mert nem túl találó a név
- ▷ XML SOAP Request üzenetre XML SOAP Response üzenet érkezik (XSD írja le a típus információkat!)
- ▷ **Web Services Description Language (WSDL)** írja le a kommunikáció lehetőségeit és szabályait
- ▷ Általában HTTP(S)-n keresztül küldjük (hasonlóan a *RESTful* webszolgáltatásokhoz), ám ez csupán egy gyakori opció jelen esetben
- ▷ Rendkívül jól bővíthető, nagyszerű és kiforrott (és jelenleg is bővülő) szabványok vezérlik
- ▷ Titkosítás és digitális aláírás(ok) támogatása - **Web Services Security (WS-Security, WSS)**
- ▷ **Universal Description Discovery and Integration (UDDI)**, webszolgáltatások regisztrálása szolgáltatások keresőbe

# Top-Down vs. Bottom-Up megközelítés

- ▷ **Top-down** megközelítés: WSDL-t készítünk, majd ebből gépi úton legeneráljuk a szükséges Java osztályokat (*stub*-okat az XSD alapján, és *service*-t a WSDL többi része alapján)
- ▷ **Bottom-up** megközelítés: Java kódot készítünk (*stub*-ok, *service*, számos annotáció), és ebből generáltatjuk le a WSDL-t

## Kliens vonatkozásában

Kliens készítéshez csak és kizárólag a WSDL-re van szükség (ettől lesz a történet programozási nyelvek között szabadon átjárható, mégis közel olyan típusos (és egyértelmű, gép és ember számára is), mint pl. a Java).

# Konfiguráció

SOAP üzenet formátuma vonatkozásában (négy kombináció)

- ▷ **Encoding Style/Communication Style** (megnevezett tulajdonság a WSDL-ben)
  - **Document/Message-Oriented**: szabadon formázható XML tartalom (egyedül az XML formázás szabályainak kell megfelelni)
  - **RPC**: kötöttebb (sokszor bővebb, a művelet nevének és argumentumainak muszáj jelen lennie), ám gépi úton könnyebben automatizálható
- ▷ **Encoding Models** (megnevezett tulajdonság a WSDL-ben)
  - **Literal**: a tartalomnak illeszkednie kell a WSDL-ben leírt, felhasználó által definiált XSD típus információkra (XSD validáció). További előny hogy XSLT segítségével az üzenet válasza könnyen és egyértelműen átalakítható mássá (pl. XHTML dokumentummá egy weboldal/webalkalmazás számára)
  - **Encoded**: csak és kizárólag előre definiált XSD típusok használhatóak (nehézkesebb validáció)

*Document style* esetén lehetőség van a *SOAP Response* validálására, míg *RPC* esetén ez nagyon körülményes, bizonytalan. Az általunk választott konfiguráció a `Document/literal` lesz.

- ▷ **Parameter Style** (a hivatkozások lesznek máshogy szervezve a WSDL-ben)
  - **BARE**: nem csomagol semmit a SOAP üzenetekbe, ha olvasható eredményt szeretnénk, mi magunknak kell csomagoló osztályokat készítenünk (mind paraméterként, mind visszatérési értéként). A WSDL szintjén az `operation`, `message` és `element` részek nevei különbözőek lesznek/lehetnek ez esetben.
  - **WRAPPED**: a kérés és a válasz paramétereit becsomagolja a művelet köré (átláthatóbb, ám néhol hosszabb SOAP üzeneteket eredményezhet). A WSDL szintjén az `operation`, `message` és `element` részek nevei azonosak lesznek ez esetben.

Az általunk választott konfiguráció a **WRAPPED *Parameter Style*** lesz.

# WSDL felépítése

A megvalósítandó feladat példája alapján



```
1 <wsdl:definitions [...]>
2   <wsdl:types>
3     <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
4       xmlns:tns="http://www.qvaevisz.hu/WebStore"
5       targetNamespace="http://www.qvaevisz.hu/WebStore"
6       version="1.0">[...]</xs:schema>
7   </wsdl:types>
8   <wsdl:message name="...">[...]</wsdl:message>
9   <wsdl:portType name="WebStore">
10     <wsdl:operation name="...">
11       <wsdl:input message="..." name="...">
12         <wsdl:output message="..." name="..."></wsdl:output>
13       <wsdl:fault message="..." name="..."></wsdl:fault>
14     </wsdl:operation>
15   </wsdl:portType>
16   <wsdl:binding name="WebStoreServiceSoapBinding" type="tns:WebStore">
17     <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
18     <wsdl:operation name="...">
19       <soap:operation soapAction="..." style="document"/>
20       <wsdl:input name="..."><soap:body use="literal"/></wsdl:input>
21       <wsdl:output name="..."><soap:body use="literal"/></wsdl:output>
22       <wsdl:fault name="..."><soap:fault name="..." use="literal"/></wsdl:fault>
23     </wsdl:operation>
24   </wsdl:binding>
25   <wsdl:service name="WebStoreService">
26     <wsdl:port binding="tns:WebStoreServiceSoapBinding" name="WebStorePort">
27       <soap:address location="http://localhost:8080/webstore/WebStoreService"/>
28     </wsdl:port>
29   </wsdl:service>
30 </wsdl:definitions>
```

A WSDL egyes részei egymás elemeire vagy egészére hivatkoznak. A service hivatkozik a binding-ra, a binding a portType-ra és az azon belül definiált operation-ökre, a portType operation-jei hivatkoznak a message-ekre, a message-ek pedig a WSDL-ben megtalálható XSD schema elemekre (types rész).

# SOAP üzenetek - Request és Response

```
1 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:web="http://www.qvaevisz.hu/WebStore">
2   <soapenv:Header/>
3   <soapenv:Body>
4     <web:ListAllProducts/>
5   </soapenv:Body>
6 </soapenv:Envelope>
```

## SOAP Request

A <http://schemas.xmlsoap.org/soap/envelope/> névtér határozza meg az alap elemeket és attribútumokat (pl. Envelope, Header, Body, stb), míg a <http://www.qvaevisz.hu/WebStore> névtér a saját alkalmazásunk névtére, amit majd definiálni fogunk (pl. ListAllProducts, ListAllProductsResponse).

```
1 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
2   <soap:Body>
3     <ns2:ListAllProductsResponse xmlns:ns2="http://www.qvaevisz.hu/WebStore">
4       <Product brand="SONY" productName="ZD9 4K HDR" price="1499"/>
5       <Product brand="SONY" productName="SD85 4K HDR" price="1299"/>
6       <Product brand="SONY" productName="XD83 4K HDR" price="1299"/>
7       <Product brand="PHILIPS" productName="40PFH5500 Smart Led" price="999"/>
8       <Product brand="PANASONIC" productName="TX-40CS620E LED " price="1350"/>
9       <Product brand="PANASONIC" productName="TX-58DX800E" price="699"/>
10      </ns2:ListAllProductsResponse>
11    </soap:Body>
12 </soap:Envelope>
```

## SOAP Response



**Feladat:** Készítsünk egy webáruház alkalmazást, melyben különféle márkájú TV készülékeket lehessen vásárolni. Az alkalmazás a raktáron lévő termékeket memóriában tárolja, a vásárlók hitelesítésével ne foglalkozzunk.

Készítsünk a **raktárkészlet lekérdezésére** szolgáló, illetve a **webkosár kezelésére** alkalmas **SOAP webszolgáltatást!**

A webkosár az adott *user-session* idejére tartalmazzon termékeket (márka, megnevezés, egységár). Ha ugyanabból a termékből többet vásárol a felhasználó, csak a darabszámot növeljük a kosárban!



```
[gradle|maven]\jboss\webstore
```

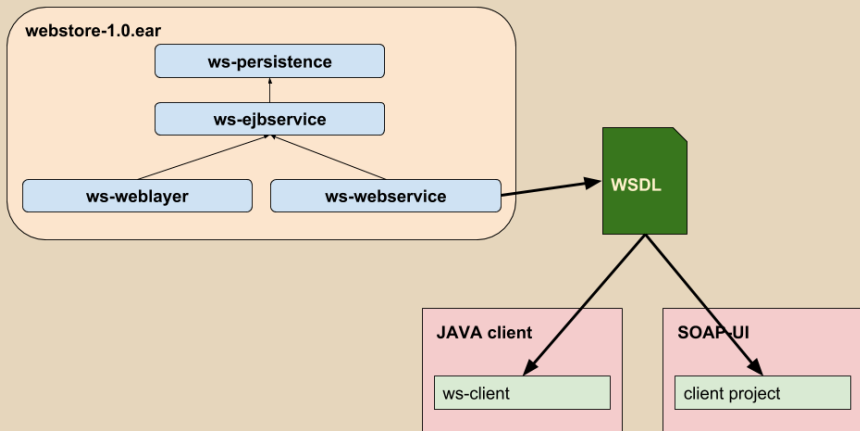




- ▷ A vásárló a kosárnak adhasson egy azonosítót/nevet, melyet beállítás után vissza is kérhet (csupán demonstrációs cél). Ha névtelen kosár nevét kérjük le, a rendszer adjon **SOAP Fault** hibaüzenetet a felhasználónak.
- ▷ A webkosár tartalmát, állapotát **Stateful Session Bean**-ben tároljuk.
- ▷ Kliens alkalmazásként **SOAP-UI** programot illetve saját készítésű **Java kliens** alkalmazást is készítsünk.

# Project felépítése

## Module-ok elhelyezkedése





```
1 @Local
2 public interface StoreHolder {
3     void create(Product product);
4     Product read(String name);
5     List<Product> readAll(String nameTerm);
6     List<Product> readAll(Brand brand);
7     List<Product> readAll();
8     void delete(String name);
9 }
```

### StoreHolder.java

```
1 @Local
2 public interface PersistenceService {
3     void create(Product product) throws PersistenceException;
4     Product read(String name) throws PersistenceException;
5     List<Product> readAll(String nameTerm) throws PersistenceException;
6     List<Product> readAll(Brand brand) throws PersistenceException;
7     List<Product> readAll() throws PersistenceException;
8     void delete(String name) throws PersistenceException;
9 }
```

### PersistenceService.java

A **StoreHolder Singleton Session Bean** tárolja (és inicializálja) a saját *domain* típusokat (Brand és Product), míg a **PersistenceService Stateless Session Bean** ehhez biztosít hozzáférést az *EJB service* réteg szolgáltatásai számára.

A Basket osztály tartalmaz üzleti metódusokat is a webkösár kezeléséhez (`increment()`, `decrement()`, `getSize()`, `getTotal()`).



## ▷ Persistence layer

- **Brand**: PHILIPS, SONY, PANASONIC
- **Product**: **brand** (Brand), **name** (String), **price** (int)

## ▷ EJB Service layer

- **BrandStub**: PHILIPS, SONY, PANASONIC
- **ProductStub**: **brand** (BrandStub), **name** (String), **price** (int)
- **BasketItem**: **product** (ProductStub), **quantity** (int)
- **Basket**: **identifier** (String), **items** (List<BasketItem>)
- **ServiceError**: **code** (int), **message** (String)

## XML annotációk

A *stub*-ok **XML Bind annotációkkal** vannak gazdagítva, hiszen ezek részt vesznek a SOAP kommunikációban, a SOAP XML üzenetek szerver részét fogják képezni!

`@XmlAccessorType(XmlAccessType.FIELD)`: Az annotációk a mezőkön vannak definiálva.

`@XmlAccessorType(XmlAccessType.PROPERTY)`: Az annotációk a *getter* metódusokon vannak definiálva.

# XML Binding

```
1 @XmlAttribute(name = "productName")
2 private String name;
```

A FIELD alapú elérésre példa a termék neve, mely esetén a SOAP-ban szereplő üzleti név eltér a forráskód szabályai szerint megadottól. Az `@XmlAttribute` annotáció értelemszerűen attribútum lesz, itt XML-ben a *camelCase* írásmód az elterjedt, illetve ez az annotáció nem alkalmazható összetett típusokra.

```
1 @XmlAttribute(name = "total")
2 public int getTotalPrice() {..}
```

A kosár teljes értékét egy **számított mezővel** tudjuk megjeleníteni az XML üzenetekben, ez esetben kizárólag a PROPERTY alapú elérés használható.

```
1 @XmlElement(name = "Items")
2 public List<BasketItem> getItems() {..}
```

Listák, tömbök, halmazok esetén a megadott elem név (`@XmlElement` annotáció) a gyermek elemek csoportosító elem neve lesz. *Element*-ek esetén az elterjedt írásmód XML-ben a *CamelCase*, mely eltér a Java elnevezés szabályaitól.



A `ServiceError` tartalmából fogunk későbbiekben **SOAP Fault custom** hibaüzenetet (XML) gyártani. Természetesen most is az EJB réteg kivételt fog dobni hiba esetén (`ServiceException`), melybe elcsomagolunk egy hibakódot (`WebStoreError` enum: `IDENTIFIER`, `PERSISTENCE`, `PRODUCT`, `BASKET`).

## Rétegek közötti függőség

A nyilvános interface-ek felé (SOAP) az EJB szolgáltatások rétege a már megismert minták alapján a `Product` és `Brand` példányokból stub-okat készít a `ProductConverter` *Stateless Session Bean* segítségével. Vegyük észre, hogy attól függetlenül, hogy az adatokat memóriában tároljuk, az EJB service felépítése azonos a korábban megismerttel. Bár jelen esetben *overkill*-nek tűnhet a konverzió és akár egészében a teljes *persistence* réteg, e megoldás segítségével később úgy vezethetjük be pl. a RDBMS-ben való tárolást, hogy a teljes szolgáltatás réteg érintetlen maradhat (**nem függ a tárolási réteg a logikai rétegtől**).



```
1 @Local
2 public interface WebBasketService {
3     void setIdentifier(String identifier) throws ServiceException;
4     String getIdentifier() throws ServiceException;
5     int getBasketSize() throws ServiceException;
6     void addItem(String productName) throws ServiceException;
7     void removeItem(String productName) throws ServiceException;
8     Basket getContent() throws ServiceException;
9 }
```

Stateful Session Bean

WebBasketService.java

```
1 @Local
2 public interface StoreService {
3     void add(ProductStub product) throws ServiceException;
4     ProductStub get(String name) throws ServiceException;
5     List<ProductStub> list(String nameTerm) throws ServiceException;
6     List<ProductStub> list(BrandStub brand) throws ServiceException;
7     List<ProductStub> getAll() throws ServiceException;
8     void remove(String name) throws ServiceException;
9 }
```

Stateless Session Bean

StoreService.java



# Stateful Session Bean

- ▷ Az implementációt tartalmazó osztály rendelkezik a `@Stateful` annotációval.
- ▷ Az osztály tartalmazhat **instance field**-eket, melyek állapota *session*-önként értelmezett. Ha ugyanaz a *session* ismét meghívja a SFSB egy másik metódusát, az *EJB container* olyan állapotú példányt ad vissza kiszolgálásra, melyben megtalálhatóak a *session* állapotai (mező értékei).
- ▷ `@PostConstruct` annotációval ellátott metódus akkor fog lefutni, amikor új *session* hív meg üzleti metódust a bean-en belül.
- ▷ `@Remove` annotációval ellátott metódus akkor fog lefutni, amikor a *session* már *invalid*, nem elérhető (pl. *timeout* miatt).

## Állapotkezelés

A Stateful Session Bean mellett a Singleton Session Bean is tárolhat állapotot, az életciklusa utóbbinak mégis a Stateless Session Bean-hez hasonló, mivel itt nincs szükség ún. **aktiválásra** és **passzíválásra**.



# Session Bean

## Állapotok és életciklusok

- ▷ Stateless és Singleton Session Bean-ek, illetve Message-Driven Bean-ek esetén
  - **Does not exist** állapot
  - **Ready** állapot (üzleti metódus hívásra kész *proxy*-n keresztül)
- ▷ Stateful Session Bean-ek esetén
  - **Does not exist** állapot
  - **Ready** állapot (aktív, üzleti metódus hívásra kész)
  - **Passive** állapot
    - Memóriából a "secondary storage"-re írja a *container* az állapotát, mert még szükség lehet rá, de most épp nem használja a *user*
    - `@PrePassivate` és `@PostActivate` metódusok készíthetők

`@PostConstruct` és `@PreDestroy` metódus mindegyik típusú *Session Bean*-hez készíthető.



WSDL: <http://localhost:8080/webstore/WebStoreService?wsdl>

```
1 package hu.qvaevisz.webstore.webservice;
2 [...]
3 @WebService(name = "WebStore", serviceName = "WebStoreService", targetNamespace =
4   "http://www.qvaevisz.hu/WebStore")
5 @SOAPBinding(style = Style.DOCUMENT, use = Use.LITERAL, parameterStyle =
6   ParameterStyle.WRAPPED)
7 public class WebStoreService {
8   [...]
9   @EJB
10  private StoreService storeService;
11
12  @WebMethod(action = "http://www.qvaevisz.hu/WebStore/getProduct", operationName =
13    "GetProduct")
14  @WebResult(name = "Product")
15  public ProductStub getProduct(@WebParam(name = "Name") String name) throws
16    WebStoreServiceException {
17    try {
18      return this.storeService.get(name);
19    } catch (ServiceException e) {
20      throw new WebStoreServiceException(e.getMessage(), e.getError());
21    }
22  }
23  [...]
24 }
```

Névtér beállítások a `@WebService` annotáció, míg konfigurációs elemek a `@SOAPBinding` annotáció segítségével valósíthatók meg.



A **@WebMethod** annotációval ellátott metódus lesz egy művelet/üzenet pár a web-szolgáltatásban. Az annotáció attribútumaival tudjuk részletesen konfigurálni, hogy milyen néven jelenik meg majd mindez a WSDL-ben (mindezekre kihatnak a konfigurációs beállítások is). Ha az *operation* neve *GetProduct* (és ez *element*-ként megjelenik az *SOAP Request*-ben), akkor erre a válasz a *SOAP Response*-ban *GetProductResponse* lesz.

```
1 @WebMethod(action = "http://www.qvaevisz.hu/WebStore/getProduct", operationName =  
2     "GetProduct")  
3 @WebResult(name = "Product")  
4 public ProductStub getProduct(@WebParam(name = "Name") String name) throws  
5     WebStoreServiceException {  
6     [..]  
7 }
```

A **@WebResult** annotáció a visszatérési érték *wrapper*e lesz. A *ProductStub* példány egy *Product* elembe lesz visszaküldve a *SOAP Response*-ban. Ha nem definiáljuk a **@WebResult**-ot, akkor egy *return* elemben jelenik meg a *ProductStub* XML szerializációja (az **@XmlRootElement** annotáció jelen konfiguráció esetén nem számít). A **@WebParam** annotáció a kérés (*request*) XML elem nevét tudja definiálni (egy *Name* elementben lehet megadni a keresett termék pontos megnevezését).

# SOAP üzenetek - GetProduct

```
1 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
2   xmlns:web="http://www.qwaevisz.hu/WebStore">
3   <soapenv:Header/>
4   <soapenv:Body>
5     <web:GetProduct>
6       <Name>SD85 4K HDR</Name>
7     </web:GetProduct>
8   </soapenv:Body>
</soapenv:Envelope>
```

## SOAP Request

```
1 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
2   <soap:Body>
3     <ns2:GetProductResponse xmlns:ns2="http://www.qwaevisz.hu/WebStore">
4       <Product brand="SONY" productName="SD85 4K HDR" price="1299"/>
5     </ns2:GetProductResponse>
6   </soap:Body>
7 </soap:Envelope>
```

A brand, productName és price attribútum nevek a ProductStub XML Bind annotációin keresztül konfigurálhatóak.

## SOAP Response

### Hibaesetek kezelése

Ha a *WebMethod* kivételt dob, akkor ebből **SOAP Fault** válasz készül. Ellenkező esetben a visszatérési érték példánya XML szerializáláson esik keresztül.



```
1 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
2   xmlns:tns="http://www.qvaevisz.hu/WebStore" >
3   <xs:element name="GetProduct" type="tns:GetProduct"/>
4   <xs:element name="GetProductResponse" type="tns:GetProductResponse"/>
5   <xs:element name="Product" type="tns:productStub"/>
6   <xs:complexType name="GetProduct">
7     <xs:sequence>
8       <xs:element minOccurs="0" name="Name" type="xs:string"/>
9     </xs:sequence>
10  </xs:complexType>
11  <xs:complexType name="GetProductResponse">
12    <xs:sequence>
13      <xs:element minOccurs="0" ref="tns:Product"/>
14    </xs:sequence>
15  </xs:complexType>
16  <xs:complexType name="productStub">
17    <xs:sequence/>
18    <xs:attribute name="brand" type="tns:brandStub"/>
19    <xs:attribute name="productName" type="xs:string"/>
20    <xs:attribute name="price" type="xs:int" use="required"/>
21  </xs:complexType>
22  <xs:simpleType name="brandStub">
23    <xs:restriction base="xs:string">
24      <xs:enumeration value="PHILIPS"/>
25      <xs:enumeration value="SONY"/>
26      <xs:enumeration value="PANASONIC"/>
27    </xs:restriction>
28  </xs:simpleType>
29  [..]
30 </xs:schema>
```

A típus információkhoz egyrészt a <http://www.w3.org/2001/XMLSchema> namespace alatti értékeket (pl. int, string), másrészt az alkalmazás saját névtere (<http://www.qvaevisz.hu/WebStore>) által bevezetett típusok használhatóak.

# WSDL - Termék lekérdezése

## Üzenetek definiálása



```
1 <wsdl:message name="GetProduct" >
2   <wsdl:part element="tns:GetProduct" name="parameters"></wsdl:part>
3 </wsdl:message>
4 <wsdl:message name="GetProductResponse" >
5   <wsdl:part element="tns:GetProductResponse" name="product"></wsdl:part>
6 </wsdl:message>
7 <wsdl:message name="WebStoreServiceException" >
8   <wsdl:part element="tns:WebStoreServiceFault"
9     name="WebStoreServiceException"></wsdl:part>
10 </wsdl:message>
11 <wsdl:portType name="WebStore" >
12   <wsdl:operation name="GetProduct" >
13     <wsdl:input message="tns:GetProduct" name="GetProduct"></wsdl:input>
14     <wsdl:output message="tns:GetProductResponse"
15       name="GetProductResponse"></wsdl:output>
16     <wsdl:fault [..] />
17   </wsdl:operation>
18 </wsdl:portType>
19 <wsdl:binding name="WebStoreServiceSoapBinding" type="tns:WebStore" >
20   <wsdl:operation name="GetProduct" >
21     <soap:operation soapAction="http://www.qvaevisz.hu/WebStore/getProduct"
22       style="document"/>
23     <wsdl:input name="GetProduct" >
24       <soap:body use="literal"/>
25     </wsdl:input>
26     <wsdl:output name="GetProductResponse" >
27       <soap:body use="literal"/>
28     </wsdl:output>
29     [..]
30   </wsdl:operation>
31 </wsdl:binding>
```

Hivatkoznak a típus definíciókra.

A hibakezeléshez kapcsolódó típusokat és üzeneteket nem tartalmazza a kivágott WSDL részlet az áttekinthetőség végett.

# SOAP Webservice - Hibakezelés

## SOAP Fault

Ha a szerver oldalon hiba történik, **SOAP Fault** üzenetet illendő válaszként adni. A *SOAP Fault* jelzi a problémát (faultcode és faultstring), és opcionálisan részleteket is közölhet (detail). Utóbbi egy Java kivétel osztály XML szerializálásának eredménye lehet!

```
1 <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
2   <soap:Body>
3     <soap:Fault>
4       <faultcode>soap:Server</faultcode>
5       <faultstring>Product does not exist in the catalog (name: lorem).</faultstring>
6       <detail>
7         <ns2:WebStoreServiceFault xmlns:ns2="http://www.qvaevisz.hu/WebStore">
8           <code>20</code>
9           <message>Product does not exist in the catalog (name: lorem).</message>
10          </ns2:WebStoreServiceFault>
11        </detail>
12      </soap:Fault>
13    </soap:Body>
14  </soap:Envelope>
```



```
1 package hu.qvaevisz.webstore.ejb-service.domain;
2 [..]
3 @XmlRootElement(name = "ServiceError")
4 public class ServiceError implements Serializable {
5
6     private final int code;
7     private final String message;
8
9     public ServiceError() { this(0, null); }
10
11    public ServiceError(final int code, final String message) { [..] }
12
13    @XmlElement(name = "code")
14    public int getCode() {
15        return this.code;
16    }
17
18    @XmlElement(name = "message")
19    public String getMessage() {
20        return this.message;
21    }
22
23    @Override
24    public String toString() {
25        return "ServiceError [code=" + this.code + ", message=" + this.message + "];"
26    }
27 }
```

JAX-B: Alapértelmezés szerint az XML szerializálás `@XmlAccessorType(XmlAccessType.PROPERTY)` konfiguráció szerint történik, vagyis a *getter* metódusok határozzák meg a kiírandó mezőket.



# Egyedi SOAP Fault üzenet

ws-webservice project



```
1 package hu.qvaevisz.webstore.webservice.exception;
2 [...]
3 @WebFault(name = "WebStoreServiceFault", targetNamespace =
4     "http://www.qvaevisz.hu/WebStore")
5 public class WebStoreServiceException extends Exception {
6     private final ServiceError faultInfo;
7     public WebStoreServiceException(String message, ServiceError
8         faultInfo) {
9         super(message);
10        this.faultInfo = faultInfo;
11    }
12    public WebStoreServiceException(String message, ServiceError
13        faultInfo, Throwable cause) {
14        super(message, cause);
15        this.faultInfo = faultInfo;
16    }
17    public ServiceError getFaultInfo()
18        return this.faultInfo;
19    }
20 }
```

A kivétel dobása a `@WebMethod`-ból fogja indikálni a *SOAP Fault* elkészítését, amennyiben a kivétel `@WebFault` annotációval rendelkezik.

A kivétel `message` attribútuma kerül a *SOAP Fault* `faultstring` részébe. De ettől természetesen el lehet térni.

A `getFaultInfo()` *getter* metódus (a metódust így kell elnevezni) által visszaadott DTO példány *XML serializálás* során bekerül a *SOAP Fault* detail részébe.

- ▷ <https://www.soapui.org/>
- ▷ Verzió: **v5.4.0**
- ▷ Telepítés: next-next-finish
- ▷ Ingyenes, open-source, létezik fizetős Pro változat is
- ▷ Function/End-to-End tesztelésre alkalmas eszköz
  - SOAP webszolgáltatások esetén
  - RESTful webszolgáltatások esetén
  - JMS támogatás (**HermesJMS** van integrálva)
- ▷ magas szinten támogatja a *security* szabvány megoldásokat (titkosítás, digitális aláírás, stb.)

File | New SOAP project

- ▷ Project name: **WebStore**
- ▷ Initial WSDL:  
<http://localhost:8080/webstore/WebStoreService?wsdl>
- ▷ Create sample requests for all operations
- ▷ Creates a TestSuite for the imported WSDL

### Ennyi?

Ennyi. A WSDL minden szükséges információt tartalmaz ahhoz, hogy egy teljes körű kliens alkalmazást tudjon a SOAP UI készíteni. Nincs ebben semmi "különleges", egyetlen CLI command végrehajtásával ugyanezt mi is megteesszük majd. A WSDL-t nem szükséges elérhető URL-ként megadni, egy offline XML állomány is megteszi.



```
1 @WebMethod(action = "http://www.qwaevisz.hu/WebStore/addProduct",
2   operationName = "AddProduct")
3 public void addProduct(@WebParam(name = "Product") ProductStub
4   product) throws WebStoreServiceException {
5   [...]
6 }
```

```
1 <web:AddProduct>
2   <Product brand="PHILIPS" productName="55PFT6510/12 3D SMART
3     Ambilight" price="2499"/>
4 </web:AddProduct>
```

## SOAP Request

```
1 <ns2:AddProductResponse
2   xmlns:ns2="http://www.qwaevisz.hu/WebStore"/>
```

## SOAP Response

# Webes vásárlás folyamata - I

## SOAP Request

```
1 <SetBasketIdentifier>
2   <Identifier>42</Identifier>
3 </SetBasketIdentifier>
```

```
1 <GetBasketIdentifier/>
```

```
1 <AddItemToBasket>
2   <ProductName>SD85 4K
   HDR</ProductName>
3 </AddItemToBasket>
```

## SOAP Response

```
1 <SetBasketIdentifierResponse />
```

```
1 <GetBasketIdentifierResponse>
2   <Identifier>42</Identifier>
3 </GetBasketIdentifierResponse>
```

```
1 <AddItemToBasketResponse />
```

Az egyik TV-ből kettőt vásárolunk, ezért ezt az üzenetet 2x küldjük el, miközben egy másik TV-t is beteszünk a kosárba.

A háttérben **Stateful service** → Pl. az Identifier megadása az egyes hívások között szükségtelen (ezt a viselkedést próbálja ez a kérés szimulálni), a szerver oldalon a session szinten tárolunk állapotot, és ezáltal fognak a hívások azonos webkosárra vonatkozni. *Stateless* esetén az adatokat pl. egy in-memory adatbázisban kellene eltárolnunk, és minden kérésben szerepeltetni a kosár azonosítóját.

# Webes vásárlás folyamata - II

## SOAP Request

```
1 <GetBasketSize/>
```

```
1 <GetBasketContent/>
```

## SOAP Response

```
1 <GetBasketSizeResponse>  
2   <BasketSize>2</BasketSize>  
3 </GetBasketSizeResponse>
```

```
1 <GetBasketContentResponse>  
2   <Basket identifier="42" total="3448">  
3     <Item quantity="2" total="2598">  
4       <Product brand="SONY"  
5         productName="SD85 4K HDR"  
6         price="1299"/>  
7     </Item>  
8     <Item quantity="1" total="850">  
9       <Product brand="PHILIPS"  
10        productName="55PUH6400 UHD Smart  
        Led" price="850"/>  
11    </Item>  
12  </Basket>  
13 </GetBasketContentResponse>
```

# Java WebService kliens alkalmazás

Maga a WebService nyelvfüggetlen technológia, a WSDL birtokában bármilyen nyelven készíthetünk klienst a Java nyelven írt szerverhez (a *SoapUI*-t is Java-ban írták, de a WSDL-en kívül semmit nem adtunk meg számára). Ettől függetlenül természetesen írhatunk Java klienst mi magunk is, hiszen "csak" XML dokumentumokat kell egy adott *endpoint*-ra elküldenünk, tipikusan HTTP-n keresztül.

De még ennél is egyszerűbb dolgunk van, ha megismerjük a **wsimport** nevű CLI alkalmazást, mely a Java JDK része.

```
1 > cd ws-client
2 > wsimport -s src/main/java -d bin -keep -p
    hu.qwaevisz.webstore.client.generated
    http://localhost:8080/webstore/WebStoreService?wsdl
```

A **ws-client** *project* lesz a kliens alkalmazás. Ennek `src/main/java` *source* könyvtárába, `hu.qwaevisz.webstore.client.generated` csomagnévvel (és könyvtárstruktúrával) szeretnénk legenerálni a megadott URL-en elérhető WSDL segítségével a kliens alkalmazásunkat.

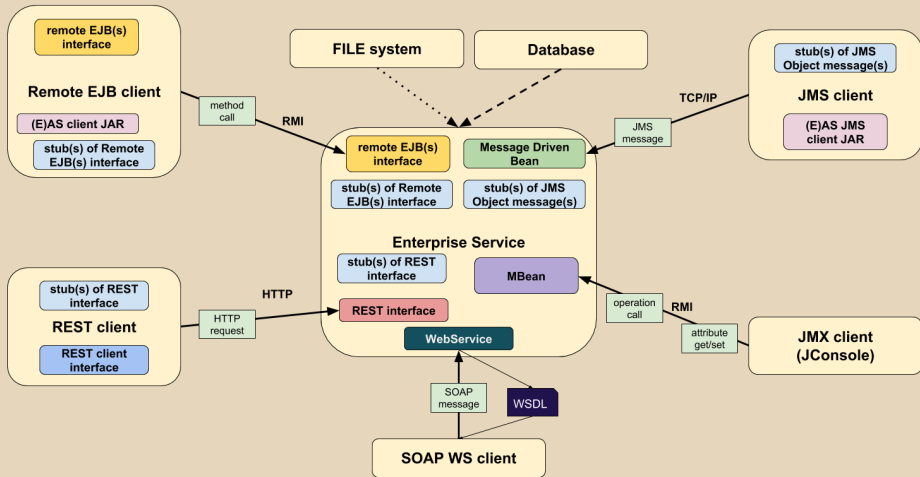


```
1 URL endpoint = new
  URL("http://localhost:8080/webstore/WebStoreService?wsdl");
2
3 WebStoreService service = new WebStoreService(endpoint);
4 WebStore webStore = service.getWebStorePort();
5
6 try {
7     List<ProductStub> products = webStore.listAllProducts();
8
9     for (ProductStub product : products) {
10        System.out.println("Product: " + product.getBrand() + ":" +
11            product.getProductName() + " ( price: " +
12            product.getPrice() + " $ )");
13    }
14
15 } catch (WebStoreServiceException e) {
16     e.printStackTrace();
17 }
```

A megadott URL jelen esetben nem lenne szükséges, hiszen az eredeti WSDL-ben is ez szerepel. Sokszor azonban a WSDL-ből kitörlik a valós URL-t (*security*), ezért a kliensnek kell ezt megadnia. Ennek egyik legegyszerűbb módja a bemutatott.



# Kommunikáció megismert formái



Az alkalmazás gyakorlatban módosítás nélkül működik *WebLogic* alatt is. Az alábbi elemeket azonban figyelembe kell venni:

- ▶ **log4j** helyett **JDK logging** használata
- ▶ a *weblogic* elvárja hogy a *Stateful Session Bean*-ben lévő `@Remote` életciklus metódus *business* metódus is legyen, ezért fel kell venni az interface-be

```
1 [...]
2 @Local
3 public interface WebBasketService {
4     [...]
5
6     void remove();
7 }
```

WebBasketService.java

- ▶ <https://pmd.github.io/>
- ▶ Verzió: **v6.1.0**
- ▶ Támogatott nyelvek: Java, JavaScript, PLSQL, Apache Velocity, XML, XSL
- ▶ Dokumentáció: <https://pmd.github.io/pmd-6.1.0/>
- ▶ Minden **Rule** egy **RuleSet** része.
- ▶ Java-hoz léteznek előre definiált *RuleSet*-ek, de sajátot is készíthetünk.
- ▶ *Rule*: név, leírás, prioritás, példa, implementáló osztály, illetve opcionális tulajdonságok halmaza (pl. *CyclomaticComplexity rule* esetén a `reportLevel` egy *Integer property*).

1. A változtatás mindenképpen szükséges. A jelenlegi viselkedés hibás, kritikus hibát jelez.
2. A kód megváltoztatása erősen javallot. A jelenlegi megoldás nagy valószínűséggel hibás.
3. A változtatás javasolt. A megvalósítás zavaró, talán hibás, avagy ellentmond az *standard*-eknek ill. *best practice*-eknek.
4. A változtatás opcionális. A jelenlegi kód valószínűleg nem hibás, csak nem követi a *standard*-eket, stílust, jó ízlést.
5. A változtatás nem szükséges. Nem árt, de olyan apróságot javít csupán mint pl. a csomag/osztály nevek konzisztenciája.

# PMD és Maven integráció



root project

```
1 <project [...]>
2   <properties>
3     <version.plugin.pmd>3.8</version.plugin.pmd>
4   </properties>
5   <build>
6     <pluginManagement>
7       <plugins>
8         <plugin>
9           <groupId>org.apache.maven.plugins</groupId>
10          <artifactId>maven-pmd-plugin</artifactId>
11          <version>${version.plugin.pmd}</version>
12          <configuration>
13            <rulesets>
14              <ruleset>/rulesets/java/basic.xml</ruleset>
15              <ruleset>/rulesets/java/braces.xml</ruleset>
16            </rulesets>
17          </configuration>
18        </plugin>
19      </plugins>
20    </pluginManagement>
21  </build>
22</project>
```

Az itt megadott *ruleset*-eknek minden projektre (minden modulra) érvényesnek kell lennie! A *pmd* ellenőrzést a `mvn clean package pmd:pmd` parancs futtatásával végezhetjük el.

# PMD Szabály halmazok leírói

pmd-bin-[version].zip | pmd-bin-[version].zip | lib | pmd-java-6.1.0.jar |  
rulesets | java | \*.xml

```
1 <ruleset name="Basic"  
2     xmlns="http://pmd.sourceforge.net/ruleset/2.0.0">  
3     <description>The Basic ruleset..</description>  
4     [..]  
5     <rule name="JumbledIncrementer"  
6         language="java"  
7         since="1.0"  
8         message="Avoid modifying .."  
9         class="net.sourceforge.pmd.lang.rule.XPathRule"  
10        externalInfoUrl="https://pmd.github.io/..">  
11     <description>Avoid jumbled..</description>  
12     <priority>3</priority>  
13     <properties>  
14         <property name="xpath">  
15             <value><![CDATA[..]]</value>  
16         </property>  
17     </properties>  
18     <example><![CDATA[..]]</example>  
19 </rule>  
</ruleset>
```

Ezeket az XML állományokat másoljuk be pl. a *root* Gradle *project* rulesets könyvtárába (pl. rulesets/basic.xml)

# PMD finomhangolása

ws-ejb-service project

Saját *ruleset* állományt legegyszerűbben úgy készíthetünk, hogy meglévő szabály halmazokat válogatunk össze, és ezeket - igény esetén - finomhangoljuk, konfiguráljuk (megváltoztatjuk eredeti beállításait).

```
1 <?xml version="1.0"?>
2 <ruleset name="Custom ruleset"
3     xmlns="http://pmd.sf.net/ruleset/1.0.0"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="http://pmd.sf.net/ruleset/1.0.0
6         http://pmd.sf.net/ruleset_xml_schema.xsd"
7     xsi:noNamespaceSchemaLocation="http://pmd.sf.net/ruleset_xml_schema.xsd">
8 <description>Custom ruleset</description>
9
10 <rule ref="[">
11 <rule ref="[">
12 <rule ref="[">
13 </ruleset>
```

ruleset.xml

# PMD finomhangolása

Saját *ruleset* definiálása meglévők alapján

```
1 <rule ref="rulesets/basic.xml" />
2
3 <rule ref="rulesets/unusedcode.xml/UnusedLocalVariable" />
4
5 <rule ref="rulesets/braces.xml">
6   <exclude name="ForLoopsMustUseBraces" />
7 </rule>
8
9 <rule ref="rulesets/basic.xml/ForLoopShouldBeWhileLoop"
10   message="IMPORTANT! This for loop could be simplified to a
11     while loop">
12   <priority>1</priority>
13 </rule>
14 <rule ref="rulesets/codesize.xml/CyclomaticComplexity">
15   <properties>
16     <property name="reportLevel" value="10" />
17   </properties>
18 </rule>
```

ws-ejbsservice | config | pmd | ruleset.xml





A javadoc egy a JDK-val szállított (`[JDK-HOME]/bin`) dokumentum készítő eszköz. A forráskód helyét (`-sourcepath`) követően fel kell sorolni a csomagot (`hu.qwaevisz.webstore.dummy`), amelyre a dokumentációt szeretnénk elkészíteni (alapértelmezetten HTML formátum, *oracle java standard doclet*).

```
1 > javadoc -sourcepath ws-dummy/src/main/java  
    hu.qwaevisz.webstore.dummy
```

Ha a forráskód lefordításához külső források is szükségesek (pl. saját *doclet*-et készítünk), akkor meg kell adni ezt is paraméterben (`-classpath`).

```
1 > javadoc -classpath ws-common/build/classes/main -sourcepath  
    ws-dummy/src/main/java hu.qwaevisz.webstore.dummy
```

# javadoc és Maven integráció

javadoc készítése Maven segítségével



```
1 <project [..]>
2   [..]
3   <properties>
4     <version.plugin.javadoc>2.10.4</version.plugin.javadoc>
5   </properties>
6   [..]
7   <build>
8     <pluginManagement>
9       <plugins>
10        <plugin>
11          <groupId>org.apache.maven.plugins</groupId>
12          <artifactId>maven-javadoc-plugin</artifactId>
13          <version>${version.plugin.javadoc}</version>
14        </plugin>
15      </plugins>
16    </pluginManagement>
17  </build>
18 </project>
```

pom.xml

```
1 > mvn site
```

- ▶ A **doclet**-ek Java nyelven írt programok, melyek a **Doclet API**-t használják hogy definiálják a tartalmát és formátumát a javadoc alkalmazásnak.
- ▶ Alapértelmezés szerint a javadoc a *standard oracle (sun) java doclet*-et használja (API dokumentáció létrehozása HTML formátumban).
- ▶ A javadoc igazi ereje a saját doclet készítésének lehetőségében rejlik.

A `docletpath` a *doclet*-et tartalmazó forrás helye (megadható *jar* is<sup>1</sup>), míg a `doclet` értéke az osztály *full qualified* neve lesz.

```
1 > javadoc -classpath ws-common\build\classes\main -docletpath
    ws-doclet\build\classes\main -doclet
    hu.qwaevisz.webstore.doclet.WebStoreDoclet -sourcepath
    ws-dummy\src\main\java -ws-filename ws.xml
    hu.qwaevisz.webstore.dummy
```

<sup>1</sup>Ha több *jar*-t sorolunk fel a `classpath` vagy `docletpath` értékeiben, *win* esetén ";", míg *nix* esetén ":" karaktert kell használni elválasztásra.