**Web Services Contract-First**

**Objetivos**

* Desenvolver Web Services usando a abordagem *contract-first*

**Projeto:**

* Construir supplier-ws a partir do contrato WSDL
* Concretizar as operações searchProducts e buyProduct
* Testar as operações a partir do supplier-ws-cli

**Web Services**

Um serviço é uma funcionalidade de um sistema de informação que pode ser invocada remotamente através da rede.  
Um *Web Service* é um serviço que usa os protocolos de comunicação da *World Wide Web* - HTTP sobre TCP sobre IP - e protocolos adicionais para descrever mensagens e dados - SOAP sobre XML.

Para permitir uma definição rigorosa das operações e dos tipos de dados dos *Web Services* são usadas as linguagens WSDL (*Web Services Description Language*) e XSD (*XML Schema Definition*), respetivamente.

**Web Service contract**

Os *Web Services* também têm uma linguagem própria para descrever o seu **contrato** com os clientes.  
A **WSDL** permite especificar a interface funcional (*port type*) - operações com entradas, saídas e erros - e também a vinculação (*binding*) com tecnologias concretas - habitualmente SOAP sobre HTTP.  
A WSDL é baseada em XML de forma a ser independente da plataforma e usa XSD para definir o detalhe dos tipos de dados de entrada e saída (e erros) em cada operação.

Exemplo:

* [Ping.wsdl](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/05-ws1/ping/Ping.wsdl) - os comentários numerados explicam as diferentes seções do documento:  
  *namespaces*, *service definition*, *binding definitions*, *port type definitions*, *message definitions* e *type definitions*.

Mais informação:

* [WSDL](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/05-ws1/wsdl/index.html)
* [XML](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/05-ws1/xml/index.html) e [XSD](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/05-ws1/xsd/index.html)
* Livro Couloris, Capítulo 9

**Java API for XML Web Services**

As bibliotecas JAX (*Java API for XML*) são a família de bibliotecas da plataforma Java que lidam com tecnologias baseadas em XML, como é o caso dos *Web Services*.

A JAX-WS (Java API for XML Web Services) é uma biblioteca para Java que permite implementar *Web Services*, usando as normas: HTTP/TCP/IP para mensagens, SOAP/XML para mensagens, WSDL e XSD para descrição.

É possível implementar Web Services partindo de um contrato WSDL (e XSD) já existente. Esta abordagem ao desenvolvimento de serviços é chamada *contract-first*.

Mais informação: [JAX-WS](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/05-ws1/jaxws/index.html)

Exemplo Ping Web Service:

* [server [ZIP](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/05-ws1/ping/ping-ws.zip)](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/05-ws1/ping/ping-ws.zip) [UML](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/05-ws1/ping/uml_server.png)
* [client [ZIP](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/05-ws1/ping/ping-ws-cli.zip)](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/05-ws1/ping/ping-ws-cli.zip) [UML](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/05-ws1/ping/uml_client.png)

Nota: As pastas que contêm o código não devem ter espaços nem caracteres acentuados no seu caminho.

**JUnit Integration Tests**

Para garantir a qualidade do código do Web Service, é necessário produzir testes de integração (IT) que verificam o comportamento de todo o sistema através de invocações *remotas*.

O JUnit pode também ser usado para fazer testes de integração.

No contexto dos Web Services, os testes de integração são um programa cliente (ws-cli) que faz invocações remotas a um programa servidor (ws), verificando o contrato das operações remotas definidas no WSDL.  
Assume-se que todos os Web Services já foram previamente lançados antes de correr os testes de integração.

Mais informação: [JUnit](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/05-ws1/junit/index.html)

**Exercício**

**Primeira parte do projeto (P1)**

O objectivo é construir o Web Service *contract-first* da primeira parte do projeto.

O ponto de partida é o [projeto base](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/proj/P1_start.zip). Este código inclui o servidor incompleto (supplier-ws [UML](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/proj/uml/uml_server.png)) e o cliente incompleto (supplier-ws-cli [UML](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/proj/uml/uml_client.png)). Antes de começar, e para prevenir conflitos de módulos Maven, alterar os pom.xmlsubstituindo as referências CXX pelo identificador do grupo.

Vamos começar pelo **servidor** supplier-ws

1. As classes de domínio da aplicação já estão implementadas.
   1. Consulte o pacote domain.
   2. Identifique o *Domain Root* e as restantes entidades representadas nas classes.
   3. Veja os mecanismos de sincronização que são utilizados para garantir que as classes podem ser chamadas corretamente por múltiplas tarefas (*threads*).
2. De seguida, consultar o contrato WSDL do serviço a implementar:
   1. [Supplier WSDL](http://disciplinas.tecnico.ulisboa.pt/leic-sod/2016-2017/labs/proj/supplier.1_0.wsdl)
   2. Analise o contrato para ver que operações define e quais os respectivos parâmetros e exceções.
3. Vamos gerar código Java a partir do WSDL. O Maven está configurado para chamar a ferramenta wsimport.
   1. Copie o ficheiro WSDL do serviço a implementar para a pasta src/main/resources do servidor
   2. cd supplier-ws
   3. mvn generate-sources  
      Caso o WSDL esteja bem formado e válido, a ferramenta wsimport gera vários ficheiros que suportam o web service. Entre eles, estarão as classes para os tipos complexos usados como parâmetros e a interface Java que define o Web Service.
   4. Faça *refresh* no Eclipse e consulte as classes geradas na pasta: target/generated-sources/wsimport.  
      Em especial, consulte a classe ...Service, e descubra a interface Java ...PortType que foi gerada a partir do WSDL.
4. Vamos agora concretizar o serviço.
   1. Consulte a classe de implementação do serviço ...PortImpl, que deverá implementar a interface Java gerada.
   2. Deverá associar a classe PortImpl ao WSDL através da anotação @WebService com os seguintes atributos: *endpoint interface* (nome do tipo Java do PortType), *wsdlLocation* (nome do ficheiro WSDL), *name*(definido no WSDL), *portName* (WSDL), *targetNamespace* (WSDL) e *serviceName* (WSDL).
   3. Além da anotação, todos os métodos listados na interface PortType devem ser implementados na classe do serviço. Cada método é uma operação do Web Service, com entradas, saídas e excepções.  
      Para cada operação, confira se está corretamente implementada.  
      Adicione a anotação @Override antes de cada método de operação, para que o compilador confirme que está a implementar corretamente o método definido na interface.  
      Note que as operações searchProducts e buyProducts não estão implementadas. Para já vamos compilar e executar o servidor sem estas operações estarem concluídas.
5. Executar o servidor:
   1. mvn compile exec:java  
      O nome da classe a executar e os argumentos estão definidos no pom.xml  
      O servidor deve executar sem erros, disponibilizando o *endpoint address*.
   2. Confirmar que o servidor está à espera de pedidos no endereço:
      * <http://localhost:8081/supplier-ws/endpoint?wsdl>
      * O contrato disponibilizado deve ser o documento original, com os respetivos comentários.

Vamos agora usar o **cliente** supplier-ws-cli para testar o servidor.

1. Vamos gerar o código Java para invocação do serviço.
   1. Consultar o pom.xml do cliente para confirmar que o WSDL está a ser corretamente referenciado (propriedades wsdl.directory e wsdl.filename)
   2. cd supplier-ws-cli
   3. mvn generate-sources  
      As classes são geradas na pasta: target/generated-sources/wsimport.
2. Vamos fazer uma chamada simples, correndo a aplicação cliente.
   1. mvn compile exec:java  
      A operação auxiliar ping deverá ser invocada.
3. Depois do teste pontual, vamos correr os testes de integração já existentes.
   1. mvn verify  
      O Maven executa todos os testes contidos em classes ...IT e um resumo é apresentado.

**O que falta fazer?**

* No servidor:
  + Implementar searchProducts e buyProduct de acordo com o enunciado
* No cliente:
  + Fazer testes de integração das operações implementadas
    - Testar casos mais importantes
    - Não esquecer os casos com entradas incorretas: null, "", valores inesperados, etc
  + Os testes de integração correm com mvn verify

**Bom trabalho!**