Um exemplo com tipos de dados mais complexos

- Um time com jogadores
- Os métodos getTeam() e getTeams()
- A classe TeamUtility simula um SGBD
- O estilo na anotação SOAPBinding
    - RPC – Usado em tipos mais simples como string e long
    - Document – Padrão. Usado para tipos mais complexos como Listas.
- Compilar: java teamws/*.java

REST

Padrão de desenvolvimento web service. É mais que um protocolo ou especificação, é uma arquitetura baseada em 5 princípios.

- Representational State Transfer
- Estilo de arquitetura de software para sistemas hipermídias distribuídos.
- Princípios
    - Dê a todas os recursos um Identificador
      Um recurso é um item que pode ser referenciado, como uma página, arquivo ou imagem. Para todo recurso deve-se ter um identificador associado (URI).
    - Vincule os recursos
      Deve-se usar links sempre que possível para vincular esses recursos.
      Exemplo: A web funciona dessa maneira.
    - Utilize métodos padronizados
      Toda aplicação que segue a arquitetura REST deve seguir os mesmo métodos padronizados (no caso da web é o protocolo HTTP)
    - Recursos com múltiplas representações
      Estabelecer formatos dos recursos para diferentes necessidades de acordo com a requisição.
      Exemplo: GET retorna resultados diferentes a partir do uso de parâmetros diferentes.
    - Comunique sem estado
      Dentro da requisição deve ter tudo que é necessário para receber uma resposta. Garante escalabilidade pois não

é necessário guardar informações específicas e se pode atender mais clientes.

PEDIR EXEMPLO

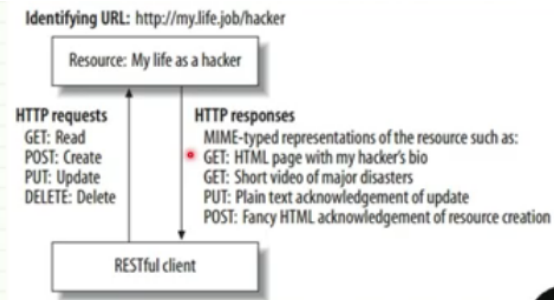
RESTful

Desenvolvimento de soluções que seguem os princípios da arquitetura REST.

- Manipulação dos princípios do REST para o desenvolvimento de soluções.
- URI(Unified Resource Identifier) - Identificação de um recurso.
  Tem formato semelhante a URL mas não obrigatoriamente permite localizar determinado recurso na Web.
  - A URI http://bedrock/citizens/fred não tem qualquer relação com http://bedrock/citizens
- Verbos do HTTP(CRUD) - Create, Read, Update e Delete.
  POST, GET, PUT, DELETE.

RESTful

O princípio chave do estilo RESTful é respeitar o significado original dos verbos HTTP.



Situação onde cliente RESTful faz requisições ao servidor.

Exemplos de verbos

HTTP verb/URI	Intended CRUD meaning
POST emps	Create a new employee from the request data
GET emps	Read a list of all employees
GET emps?id=27	Read a singleton list of employee 27
PUT emps	Update the employee list with the request data
DELETE emps	Delete the employee list
DELETE emps?id=27	Delete employee 27

Múltiplos significados para os verbos.

Uso dos verbos

- POST e PUT são utilizados em requisições que possuem corpo.
  Os dados são enviados no corpo da requisição HTTP.
- GET e DELETE em requisições sem corpo(os dados são enviados como strings na consulta).
  Não existe corpo, os dados são enviados na própria URL.
- Exemplo:
  http://www.onlineparlor.com/bets?horse=bigbrown&jockey=kent&amount=25

Versão Restful do serviço Team

```
@WebServiceProvider
@ServiceMode(value = javax.xml.ws.Service.Mode.MESSAGE)
@BindingType(value = HTTPBinding.HTTP_BINDING)
public class RestfulTeams implements Provider<Source> {
    ...
    public Source invoke(Source request) {
    ...
    }
}
```

"@WebServiceProvider" define que a arquitetura seguida será REST.

O método Invoke

Manipula as requisições e gera as respostas.

```
public Source invoke(Source request) {
    if (ws_ctx == null) throw new RuntimeException("DI failed on ws_ctx.");

    // Grab the message context and extract the request verb.
    MessageContext msg_ctx = ws_ctx.getMessageContext();
    String http_verb = (String)
       msg_ctx.get(MessageContext.HTTP_REQUEST_METHOD);
    http_verb = http_verb.trim().toUpperCase();

    // Act on the verb. To begin, only GET requests accepted.
    if (http_verb.equals("GET")) return doGet(msg_ctx);
    else throw new HTTPException(405); // method not allowed
}
```

O doGet()

```
private Source doGet(MessageContext msg_ctx) {
    // Parse the query string.
    String query_string = (String) msg_ctx.get(MessageContext.QUERY_STRING);

    // Get all teams.
    if (query_string == null)
        return new StreamSource(new ByteArrayInputStream(team_bytes));
    // Get a named team.
    else {
        String name = get_value_from_qs("name", query_string);
        // Check if named team exists.
        Team team = team_map.get(name);
        if (team == null) throw new HTTPException(404); // not found
        // Otherwise, generate XML and return.
        ByteArrayInputStream stream = encode_to_stream(team);
        return new StreamSource(stream);
    }
}
```

Códigos de status do HTTP

Códigos e seus significados gerais. É possível também customizar os códigos HTTP, com orientação de seguir os padrões gerais de intervalos utilizados.

- Em geral:
    - 100-199 - Informacional
    - 200-299 - Sucesso
    - 300-399 - Redirecionamento
    - 400-499 - Erros do cliente
    - 500-599 - Erros do servidor

HTTP status code	Official reason	Meaning
200	OK	Request OK.
400	Bad request	Request malformed.
403	Forbidden	Request refused.
404	Not found	Resource not found.
405	Method not allowed	Method not supported.
415	Unsupported media type	Content type not recognized.
500	Internal server error	Request processing failed.

Web Services X Objetos distribuídos

- Referência a objetos remotos X URIS

  O Webservice não cria ou instancia novos objetos para atender requisições, como em objetos distribuídos. O Webservice é um objeto único.
- Modelos de serviços web
- Serventes

  Nos objetos distribuídos os serventes serviam as requisições, e no webservice isso não existe, existe apenas um único objeto atendendo requisições através da rede.

Web services X CORBA

- CORBA foi projetado para uso dentro de uma única organização ou entre um pequeno número de organizações.
  Não possui escalabilidade tão grande quanto as aplicações de webservice.
- Problemas de atribuição de nomes
  O serviço de nomes é um dos motivos para os problemas de escalabilidade. O DNS é o serviço de nomes do webservice e permite uma escalabilidade enorme.
- Problemas de referência
  No CORBA usa-se o IOR para referenciar, o qual só faz sentido no repositório de interface e não em outros repositórios, ou seja, para funcionar a relação cliente e servidor é necessário estarem no mesmo repositório.
- Separação de ativação e localização
  No Webservice as URI e URL localizam um serviço web, e esse serviço é ativado quando o webservice é disponibilizado.
- Facilidade de uso
  WebService usa HTTP e XML, padrões muito conhecidos. O CORBA é grande e complexo necessitando instalar um middleware CORBA para desenvolvimento de aplicações.
- Eficiência
  O CORBA ganha do webservice neste ponto. Há uma diferença de desempenho gritante, porém o webservice é escalável e a facilidade de uso é maior. Como os hardwares e canais de comunicação estão evoluindo, a eficiência não é tão levada em consideração.
- Disponibilidade de serviços do CORBA é maior
  No CORBA há diversos serviços que podem ser previamente utilizados.