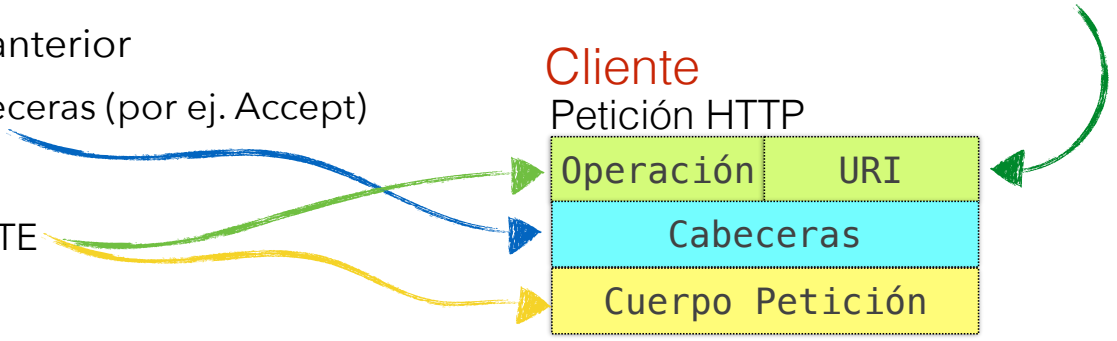
Servicios REST

Sesión 5:
API Cliente. Procesamiento JSON. Pruebas

Índice

API Cliente

- JAX-RS 2.0 incluye un API cliente de servicios REST que permite interactuar con otros servicios RESTful
- Muy útil para definir tests que prueben los propios servicios que estamos desarrollando
- Para acceder a un recurso REST mediante el API cliente es necesario seguir los siguientes pasos:
  - Obtener una instancia de la interfaz `javax.ws.rs.client.Client`
  - Configurar la instancia `Client` a través de un *target* (instancia de `javax.ws.rs.client.WebTarget`)
  - Crear una petición basada en el *target* anterior
    - Especificaremos valores para las cabeceras (por ej. Accept)
  - Invocar la petición
    - Invocaremos a GET, PUT, POST, DELETE



Ejemplos (1)

Es MUY importante **liberar** el socket para que pueda ser reutilizado por una instancia Client.

Sólo es necesario cerrar la conexión explícitamente cuando la respuesta recibida es de tipo Response

Liberar la conexión NO implica cerrar el socket

```
//creamos e invocamos una petición POST
Client client = ClientBuilder.newClient();
WebTarget target = client.target("http://localhost:8080/clientes");
Response response = target.request()
                          .post(Entity.xml(new Cliente("Alvaro", "Gomez")));
response.close();


//reutilizamos el target anterior para invocar una petición GET
Cliente cliente = target.queryParam("nombre", "Alvaro Gomez")
                        .request()
                        .get(Cliente.class);
client.close();
```

La clase javax.ws.rs.client.Entity representa la entidad del mensaje con el media type asociado

Cuando terminemos de utilizar la instancia Client, tenemos que **cerrar** la conexión (cerrar el socket) para que el socket pueda reutilizarse por el sistema

Ejemplos (2): peticiones GET

```
Client cli = ClientBuilder.newClient();
//petición get que devuelve una instancia de Cliente
Cliente cliRespuesta = cli.target("http://ejemplo/clientes/123")
                          .request("application/json")
                          .get(Cliente.class);

//petición get que devuelve una lista de objetos Cliente
List<Cliente> cliRespuesta2 =
         cli.target("http://ejemplo/clientes")
            .request("application/xml")
            .get(new GenericType<List<Cliente>>() {});

//petición get que devuelve un objeto de tipo Response
Response respuesta = cli.target("http://ejemplo/clientes/245")
                        .request("application/json")
                        .get();
try {
  if (respuesta.getStatus() == 200) {
      Cliente cliRespuesta = respuesta.readEntity(Cliente.class);
  }
} finally {
   respuesta.close();
}
```

Una respuesta HTTP con éxito se convertirá a los tipos Java específicos indicados como parámetros del método

Tipo aceptado como respuesta. Es equivalente a:

```
.request()
.accept("application/json");
```

Una respuesta HTTP con éxito se convertirá al tipo Java Response

Ejemplos (3): peticiones POST

```
Client cliente = ClientBuilder.newClient();
PedidoAlmacen pedido = new PedidoAlmacen(...);
WebTarget miRecurso = client.target("http://ejemplo/webapi/escritura");
NumeroSeguimiento numSeg =
                  miRecurso.request(MediaType.APPLICATION_XML)
                           .post(Entity.xml(pedido), NumeroSeguimiento.class);
```

El segundo parámetro indica el tipo java al que se convertirá la respuesta recibida

Ejemplo de envío de datos de un formulario en una petición POST

```
Form form = new Form().param("nombre", "Pedro")
                      .param("apellido", "Garcia");
...
response = client.target("http://ejemplo/clientes")
                 .request().
                 .post(Entity.form(form));
response.close();
```

Captura de respuestas de error

- Las respuestas de error HTTP se capturan como excepciones

```
try {
        Cliente cli = client.target("http://tienda.com/clientes/123")
                           .request("application/json")
                           .get(Cliente.class);
} catch (NotAcceptableException notAcceptable) {
        ...
} catch (NotFoundException notFound) {
        ...
}
```

La clase WebApplicationException se utiliza para producir errores específicos de HTTP. Esta clase hereda de RuntimeException e (*unckecked* exception), y sus clases hijas son:

- La clase ClientErrorException cubre cualquier código de error en la franja del 400.
- La clase ServerErrorException cubre cualquier código de error en la franja del 500.

Índice

- API Cliente
- **Procesamiento JSON**
- Pruebas

JSON

- JSON (JavaScript Object Notation) es un formato muy popular para el intercambio de datos basado en texto y bastante menos verboso que la representación XML
- Permite representar objetos con una gramática muy sencilla. Sólo se requieren dos estructuras:
  - **Objetos** (cada objeto es un conjunto de pares "nombre": "valor")
  - **Arrays**, son listas de valores
- El API for JSON processing (*JSR 353*) es un muy reciente, de 2013

```
{ "nombre": "John",
  "apellidos": "Smith",
  "edad": 25,
  "direccion": { "calle": "21 2nd Street",
                 "ciudad": "New York",
                 "codPostal": "10021"
               },
  "telefonos": [
         { "tipo": "fijo",
           "numero": "212 555-1234"
         },
         {
           "tipo": "movil",
           "numero": "646 555-4567"
         }
  ]
}
```

Hay 7 tipos de valores posibles: respuesta HTTP con éxito se convertirá al tipo Java string, number, object, array, true, false, y null

Procesamiento de JSON

- El API de procesamiento de JSON permite:
  - Crear modelos de objetos de forma programativa desde el código de la aplicación
  - Procesar objetos leídos, navegando por el modelo de objetos
  - Leer los objetos y escribirlos desde/en un stream

Creación de un modelo de objetos de forma programativa

```
import javax.json.Json;
import javax.json.JsonObject;
...
JsonObject modelo =
       Json.createObjectBuilder()
           .add("nombre", "Duke")
           .add("apellidos", "Java")
           .add("edad", 18)
           .add("calle", "100 Internet Dr")
           .add("ciudad", "JavaTown")
           .add("codPostal", "12345")
           .add("telefonos",
               Json.createArrayBuilder()
                   .add(Json.createObjectBuilder()
                           .add("tipo", "casa")
                           .add("numero", "111-111-1111"))
                   .add(Json.createObjectBuilder()
                           .add("tipo", "movil")
                           .add("numero", "222-222-2222")))
.build();
```

Navegar por el modelo de objetos

```
import javax.json.JsonValue;
import javax.json.JsonObject;
import javax.json.JsonArray;
import javax.json.JsonNumber;
import javax.json.JsonString;
...
public static void navegarPorElArbol(JsonValue arbol, String clave) {
  if (clave != null)
    System.out.print("Clave " + clave + ": ");
  switch(arbol.getValueType()) {
    case OBJECT:
       System.out.println("OBJETO");
       JsonObject objeto = (JsonObject) arbol;
       for (String nombre : object.keySet())
          navegarPorElArbol(object.get(nombre), name);
    break;
    case ARRAY:
       System.out.println("ARRAY");
       JsonArray array = (JsonArray) arbol;
       for (JsonValue val : array)
          navegarPorElArbol(val, null);
       break;
    case STRING:
      JsonString st = (JsonString) arbol;
      System.out.println("STRING " + st.getString());
      break;
    case NUMBER:
       JsonNumber num = (JsonNumber) arbol;
       System.out.println("NUMBER " + num.toString());
       break;
    case TRUE: case FALSE: case NULL:
        System.out.println(arbol.getValueType().toString());
        break;
  }
}
```

Escritura del modelo de objetos en un stream

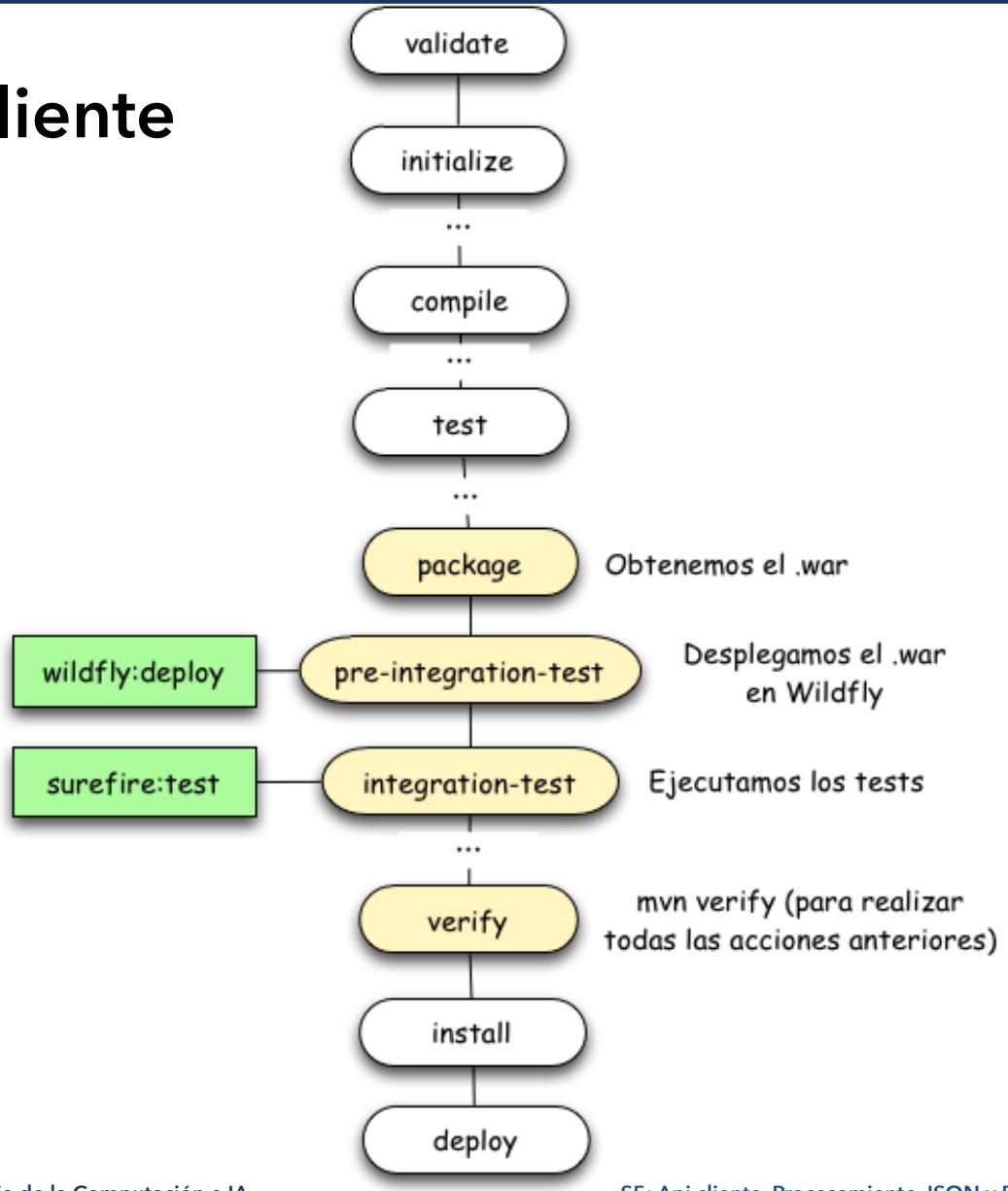
```
import java.io.StringWriter;
import javax.json.JsonWriter;
...
StringWriter stWriter = new StringWriter();
JsonWriter jsonWriter = Json.createWriter(stWriter);
jsonWriter.writeObject(modelo);
jsonWriter.close();
String datosJson = stWriter.toString();
System.out.println(datosJson);
```

Índice

- API Cliente
- Procesamiento JSON
- **Pruebas**

Tests en Maven con el API cliente

- Los tests se ejecutarán después de empaquetar y desplegar el .war en Wildfly
- Los tests NO se ejecutarán en el servidor, por lo que no vamos a utilizar Arquillian, sólo Maven y JUnit
- Pero tendremos que realizar algunos cambios en el pom.xml para asegurarnos de que se realizan las acciones que nos interesan (empaquetado-despliegue-ejecución de tests)



Modificaciones en la configuración de la construcción

```
<!-- forzamos el despliegue del war generado
 durante la fase pre-integration-test,
 justo después de obtener dicho .war -->
<plugin>
    <groupId>org.wildfly.plugins</groupId>
    <artifactId>wildfly-maven-plugin</artifactId>
    <version>1.0.2.Final</version>
    <configuration>
        <hostname>localhost</hostname>
        <port>9990</port>
    </configuration>
    <executions>
        <execution>
            <id>wildfly-deploy</id>
            <phase>pre-integration-test</phase>
            <goals>
                <goal>deploy</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

```
<!--ejecutaremos los test JUnit en la fase
    integration-test, inmediatamente después de la
    fase pre-integration-test,
    y antes de la fase verify-->
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>2.18</version>
    <configuration>
        <skip>true</skip>
    </configuration>
    <executions>
        <execution>
            <id>surefire-it</id>
            <phase>integration-test</phase>
            <goals>
                <goal>test</goal>
            </goals>
            <configuration>
                <skip>false</skip>
            </configuration>
        </execution>
    </executions>
</plugin>
```

Librerías necesarias

- Junit (para implementar los tests)
- resteasy-client (api jaxrs Client)
- resteasy-jaxb-provider (para serializar-deserializar a xml)
- resteasy-jackson-provider (para serializar-deserializar a json)
- resteasy-json-p-provider (api jaxrs Json)
- hamcrest-json (utilidades para utilizar Matchers y realizar aserciones sobre objetos Json)

JUnit: Aserciones AssertThat

- Sintaxis: **assertThat([value], [matcher statement]);**
  - ***value*** es el resultado real (valor sobre el que se quiere afirmar algo
  - ***[matcher statement]*** es un objeto Matcher que se utiliza para comprobar si el resultado real satisface el "patrón" indicado por dicho objeto Matcher
- Ejemplos:
  - assertThat(x, is(not(4)));
  - assertThat(responseStringJson, either(containsString("nombre")).and(containsString("apellido")));
  - assertThat(myList, hasItem("3"));
- JUnit incluye parte de los Matchers de la librería Hamcrest. Para trabajar con objetos Json utilizaremos la librería hamcrest-json, que nos permitirá utilizar el siguiente Matcher:
  - ***Assert.assertThat("{\"age\":43, \"friend_ids\":[16, 52, 23]}", sameJSONAs("{\"friend_ids\":[52, 23,16]}") .allowingExtraUnexpectedFields() .allowingAnyArrayOrdering());***

Ejemplo:

```
JsonObject json_object =
    client.target("http://localhost:8080/foro/usuarios")
          .request(MediaType.APPLICATION_JSON) .get(JsonObject.class);

//resultado REAL
String json_string = json_object.toString();

JsonObject usuarios =
      Json.createObjectBuilder()
          .add("usuarios",
             Json.createArrayBuilder()
                 .add(Json.createObjectBuilder()
                          .add("nombre", "Pepe Lopez")
                          .add("links",
                               Json.createArrayBuilder()
                                  .add(Json.createObjectBuilder()
                                    .add("uri", "http://localhost:8080/foro/usuarios/
                                    .add("type", "application/xml,application/json")
                                    .add("rel", "self"))))
                .add(Json.createObjectBuilder()
                         .add("nombre", "Ana Garcia")
                         .add("links",
                              Json.createArrayBuilder()
                                  .add(Json.createObjectBuilder()
                                    .add("uri", "http://localhost:8080/foro/usuarios/
                                    .add("type", "application/xml,application/json")
                                    .add("rel", "self")))))
        .build();

Assert.assertThat(json_string,
                  sameJSONAs(usuarios.toString())
                        .allowingExtraUnexpectedFields()
                        .allowingAnyArrayOrdering());
```

¿Alguna duda, pregunta…?