RESTful Web Services

1. Web services are headless web apps which means they have no UI but exchange data in XML or JSON formats for requests and responses. The back ends are the same as Spring MVC with respect to domain, DAO and business classes, but the controllers are slightly different.
2. RESTful web services are the most widely used web services in last 6-7 years with advent of mobile devices.
3. REST Web services use case: Imagine a home insurance company that has to track all products owned by customers and has desktop, mobile and tablet apps distributed globally. Desktops will have their own UI written in JavaFX, Spring, etc. Mobile and tablet devices will have Android, iPhone or other apps. All these apps need CRUD functionality for customers and their products.
4. REST stands for representational style.
5. Everything on Internet uses HTTP and is available on every device. How can HTTP be used to implement CRUD? Typically HTTP GET is used when a link is clicked and POST is used when forms are submitted. GET is used to read data and POST is used to update or create data. However, HTTP has two more protocol methods that have not been used in last 20 years, DELETE and PUT. Developers saw the opportunity to use GET to read data, POST to update (or create), DELETE to delete and PUT to create (or update). These methods can be used to implement CRUD on any database table for all platforms without introducing a new protocol.
6. RESTful apps are simple. With REST, URLs are mapped to Class methods with @Path annotation. These methods can receive objects coded in JSON or XML, update or query the database and return objects also coded in JSON or XML. Under the hood the framework automatically converts between Java and JSON/XML. JSON by default is built into every browser. JSON stands for JavaScript object notation. REST methods will be annotated with @GET, @POST, @DELETE or @PUT to match the HTTP method on the URL of their @PATH.
7. Web apps can access web services and this is the current way. Spring MVC used mostly for desktop apps.
8. Apache crossfire CFX (http://cxf.apache.org/) is used to facilitate building REST and SOAP web services.

Creating a RESTful Web Project Using Apache CXF in Eclipse

These instructions assume use of a recent version of Eclipse including Kepler or Luna and which is already configured with JBoss Tools. To install JBoss Tools in Eclipse, go to the Eclipse Marketplace from the Help menu, search for “jboss tools”, select the version matching your version of Eclipse and install it following the prompts.

1. In Eclipse create a new Maven project by selecting File->New->Maven Project.
2. In the New Maven Project popup click Next.
3. In the Select Archetype popup filter for “cxf” and select the cxf-jaxrs-service archetype. If no cxf-jaxrs-service archetype appears, check http://cxf.apache.org/download.html for latest version and add it to the Maven archetype library in order to be able to select and use it for a project. Unselect “Show latest version of archetype only” to see additional selections.
4. After the archetype has been selected click Next.

5. Define group id as *yourName.web* (or whatever you prefer) and artifact id as the project name you want, such as DemoRESTproject. Set the package name to the value of the *group id*. Click Finish to create the project.
6. Notice that in src/main/java/*group id*/ are HelloWorld.java and JsonBean.java.
7. Configure tomcat as the server for the project after installing tomcat if needed. Tomcat can be downloaded from http://tomcat.apache.org/. Its zip or tar.gz archive just needs to be unbundled to install it. Instructions for configuring tomcat in Eclipse Luna are available at Configuring the Server.
8. How do we execute the HelloWorld web service?
    1. The GET method can be tested with browser.
        i. Right click on project and run as -> run on server using tomcat.
        ii. In the URL of web browser that appears in Eclipse add "hello/echo/*string*" and submit the new URL to return a page with *string* printed in it in upper case.
    2. The GET method can also be tested with by running HelloWorldIT.testPing() as a JUnit test after running the project on an app server (tomcat).
    3. The POST method can be tested by running HelloWorldIT.testJsonRoundtrip() as a JUnit test after running the project on an app server (tomcat). It can also be tested with an AJAX application running on an external device like an Android.
    4. Examine src/main/java/*group id*/HelloWorld.java to learn why the web service echoes input it GETs in upper case and sets the value of JsonBean.val2 with the value of JsonBean.val1 when the latter is POSTed.