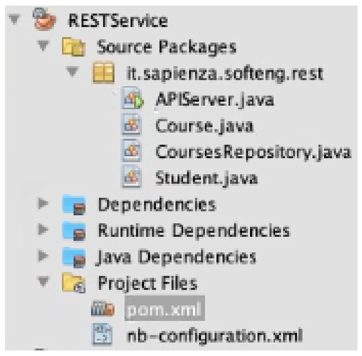
# LAB 5 - REST

POSTman simulates a REST client(useful to test rest servers )

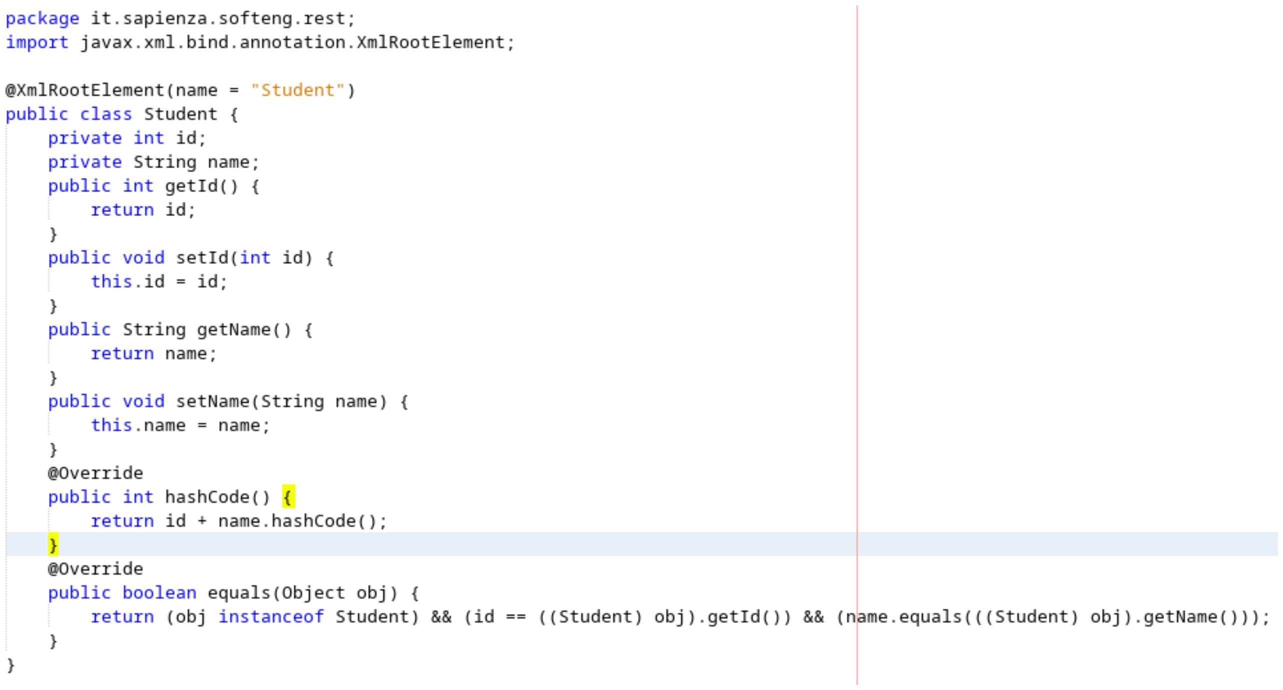


```
RESTService
  Source Packages
    it.sapienza.softeng.rest
      APIServer.java
      Course.java
      CoursesRepository.java
      Student.java
  Dependencies
  Runtime Dependencies
  Java Dependencies
  Project Files
    pom.xml
    nb-configuration.xml
```

## WebService

pom should work.
In this example data is exchange (marshalled and unmarshalled) in xml.
small notes: @Path("serviceBaseFolder") specifies the first parameter that identifies the entire service in a domain; @Produces("text/xml") specifies the type of response (there can be more than one); @GET/@PUT/@POST/@DELETE specify the API, if followed by a @Path("...") then it appends it after the method (basically a parameter for the method.

- Student: this class is called a dto class (data transfer obejct) and has no functionality. It's role is to only be a data structure, and has only the role of sending data back and forth. It has the properties and methods (set and get for each property), and then java prescribes the overriding of the equals method (and hashCode() too if necessary). Note the "@XmlRootElement": this tells the library that this class provides the marshaling for the tag "Student".



```java
package it.sapienza.softeng.rest;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "Student")
public class Student {
    private int id;
    private String name;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    @Override
    public int hashCode() {
        return id + name.hashCode();
    }
    @Override
    public boolean equals(Object obj) {
        return (obj instanceof Student) && (id == ((Student) obj).getId()) && (name.equals(((Student) obj).getName()));
    }
}
```

- Course: this class is not only a tag, but also contains other tags. still has the @ that tells this is the reference for marshaling course objects. As before it has the properties and the set/get methods for them. On top of that, it exposes the API using the keywords @GET, @POST etc. that implement the appropriate logic. Notice the private method findById

```java
@GET
@Path("{studentId}")
public Student getStudent(@PathParam("studentId") int studentId) {
    return findById(studentId);
}

@POST
public Response createStudent(Student student) {
    for (Student element : students) {
        if (element.getId() == student.getId()) {
            return Response.status(Response.Status.CONFLICT).build();
        }
    }
    students.add(student);
    return Response.ok(student).build();
}

@DELETE
@Path("{studentId}")
public Response deleteStudent(@PathParam("studentId") int studentId) {
    Student student = findById(studentId);
    if (student == null) {
        return Response.status(Response.Status.NOT_FOUND).build();
    }
    students.remove(student);
    return Response.ok().build();
}
```

- CourseRepository: It's the root repo of our service (@Path and @produces keywords). It maintains a data structure( HashMap over int id and obj course) for manging all the courses . It only has get and put active. Then it also has a "subroute" "{courseid}/students" that tells how to move from the courses to the students in the given course. Also contains any data to initialise the server data (Static allocation).

-

- CourseRepository: It's the root repo of our service (@Path and @produces keywords). It maintains a data structure( HashMap over int id and obj course) for manging all the courses . It only has get and put active. Then it also has a "subroute" "{courseid}/students" that tells how to move from the courses to the students in the given course. Also contains any data to initialise the server data (Static allocation).