1. Crea la clase Persona. Va a tener como atributos:
   - Id de tipo entero
   - Nombre de tipo cadena
   - Casado de tipo booleano
   - Sexo de tipo cadena

   Generar sus getters y setters y permitir la opción de serializado/deserializado automático.
2. Crea la clase GestionaPersona ubicada en la URI persona. Tendrá un atributo estático de tipo persona y dos métodos:
   1. Un método leer que retornará una persona en formato XML o JSON.
   2. Un método guardar, en el atributo de clase persona, que guarda una persona pasada como parámetro en formato XML o JSON
3. Crea la clase Personas ubicada en la URI personas. Tendrá como atributo estático un ArrayList de personas llamado personas y los siguientes métodos:
   1. Un método guardar que pasándole una persona en XML o JSON inserte esa persona en el ArrayList.
   2. Un método listar que devuelva el ArrayList personas como XML.
   3. Un método ver que pasándole en la URI un nombre liste, en caso de existir, los datos de esa persona como JSON.
   4. Un método ver que en el *path* buscar y pasándole una cadena como query en la URI, muestre las personas que tengan esa cadena en el nombre ignorando las mayúsculas y minúsculas.
   5. Crea un formulario para insertar datos de personas.
   6. Crea un método que permita insertar personas en el ArrayList personas mediante el formulario anterior.
   7. Crea, en el *path* add, un método que permita insertar varias personas de forma simultánea en el ArrayList personas.
   8. Crea un método, que pasándole en el *path* id una persona permita borrarla del ArrayList personas.
   9. Modifica el ejercicio del punto 4 para que los parámetros de la query tengan valores por defecto.
   10. Crea un método, en el *path* XML, que devuelva los datos de una persona indicada en el *path* en XML y en JSON siendo el id un atributo de persona. ¿Qué diferencias encuentras entre la representación obtenida en XML y en JSON?
   11. Nos piden que los nombre de los atributos devueltos deben estar en gallego. Crea un método en el *path* galego que realice esta acción.
   12. Modifica los ejercicios anteriores para que devuelvan el Response adecuado.

4. Crea la clase deportistas la cual usaremos como *front-end web* para acceder a la tabla deportistas.
    1. Crea la clase Deportista. Va a tener como atributos:
        - Id de tipo entero
        - Nombre de tipo cadena
        - Deporte de tipo cadena
        - Activo de tipo booleano
        - Genero de tipo cadena

    Nuestro servicio se compone de un API en el *path* /deportistas formado por operaciones sobre personas:

    2. Todos (/): devuelve un listado con todos los deportistas del sistema.
    3. Buscar jugador (/{id}): devuelve la información relativa al deportista id.
    4. Por deporte (/deporte/{nombreDeporte}): Lista los deportistas de un deporte.
    5. Activos (/activos): Lista los deportistas activos.
    6. Retirados (/retirados): Lista los deportistas retirados.
    7. Masculinos (/masculinos): Lista los deportistas masculinos.
    8. Femeninos (/femeninos): Lista los deportistas femeninos.
    9. Deportes por genero (/xg): Lista un array con dos elementos: uno con todos los deportistas masculinos y otro con todos los deportistas femeninos.
    10. Activos por deporte (/deporte/{nombreDeporte}/activos): Lista los deportistas activos de un deporte.
    11. Contar deportistas (/sdepor): Cuenta el número de deportistas distintos.
    12. Lista deportes (/deportes): Lista los deportes existentes ordenados alfabéticamente sin repeticiones.
    13. Crear deportista (/): Añade un deportista en el sistema.
    14. Crear deportista formulario (/): Añade un deportista mediante un formulario.
    15. Crear deportistas (/adds): crea deportistas en el sistema.
    16. Actualizar deportista (/): actualiza la información relativa a un deportista.
    17. Borrar deportista (/del/{id}): elimina la información relativa a un deportista id.
    18. Imagen deportista (/img/{id}/{num}): Muestra la imagen num del deportista id como image/jpg.
    19. Imágenes deportistas (/img/{id}): Muestra el nombre y las imágenes del deportista id como html.

20. Buscar Jugador /{id}: Realiza un método equivalente al 2, pero que en HTML devuelva los datos y las fotos del deportista id.
21. Todos html (/): Realiza en HTML una visualización de los datos de los deportistas y de sus imágenes en páginas. Cada página ha de contar con 10 deportistas. Debe además mostrar una enlace hacia la página siguiente si no es la última y un enlace a la página anterior si no es la primera. El número de página a visualizar se ha de pasar como el parámetro pag en la query (si no existe el parámetro pag ha de mostrar la primera página). Construye los enlaces con UriBuilder y uriInfo. Además añade links (en la cabecera) a cada jugador en particular.
22. Busca sin mime (/mime/{id}: Crea el método que devuelve la información relativa al deportista id. El media type producido se puede decidir con el parámetro de query *tipo*. Tendrá tres posibles valores: HTML, JSON o XML.
23. Modifica el método del *path* /activos para que permita determinar el tipo MIME producido (XML o Json) en función del parámetro de query *tipo*. Establece como valor predeterminado JSON.
24. Crea la clase ClienteJava en eclipse la cual se encargara, desde Java, de realizar peticiones al servicio web:
    - Obtén del *path* /activos la lista de deportistas como un Objeto JsonArray, visualizando los objetos JSON que contiene.
    - Obtén del *path* /activos la lista de deportistas como XML procesándolo con DOM o con SAX.
25. Mediante un cliente Rest:
    - Obtén del *path* /activos la lista de deportistas como un ArrayList visualizándolo. Usa para ello:

```
List<Deportista>deport=target.request().accept(MediaType.APPLICATION_JSON).get(new
GenericType<List<Deportista>>(){});
```