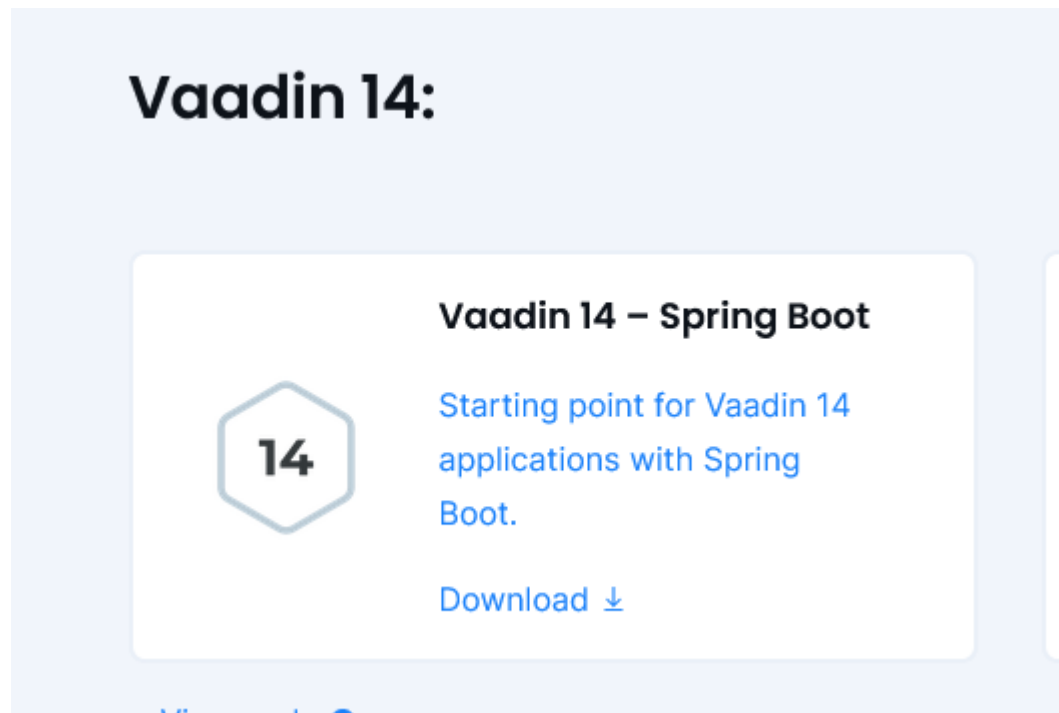


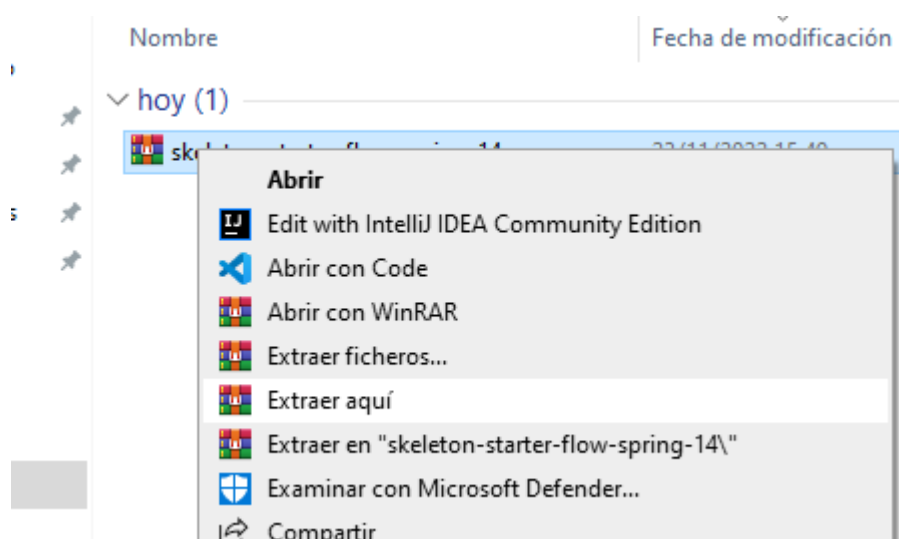
Para este ejercicio vamos a utilizar una de las plantillas de inicio que ofrece Vaadin en su web:

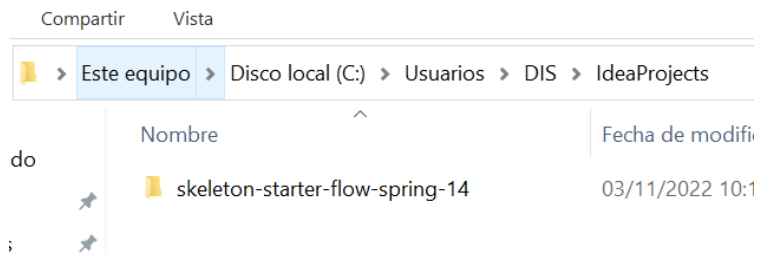
<https://vaadin.com/hello-world-starters>



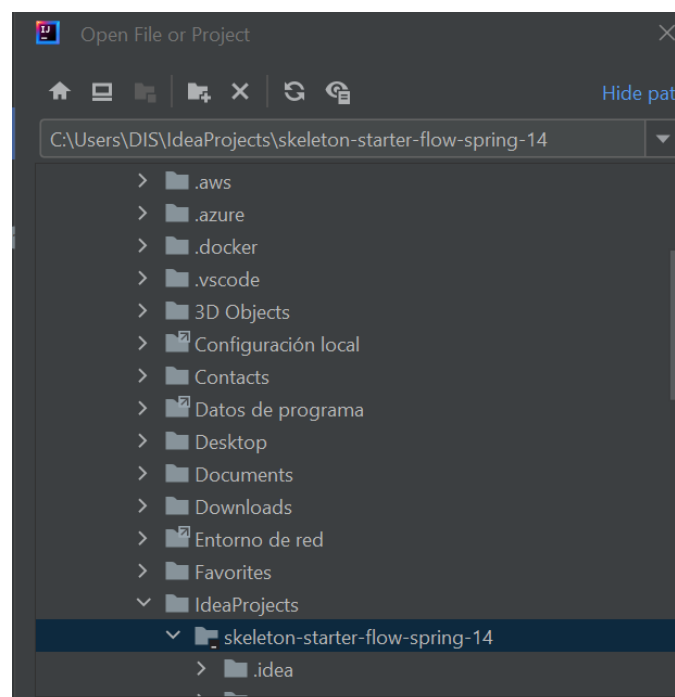
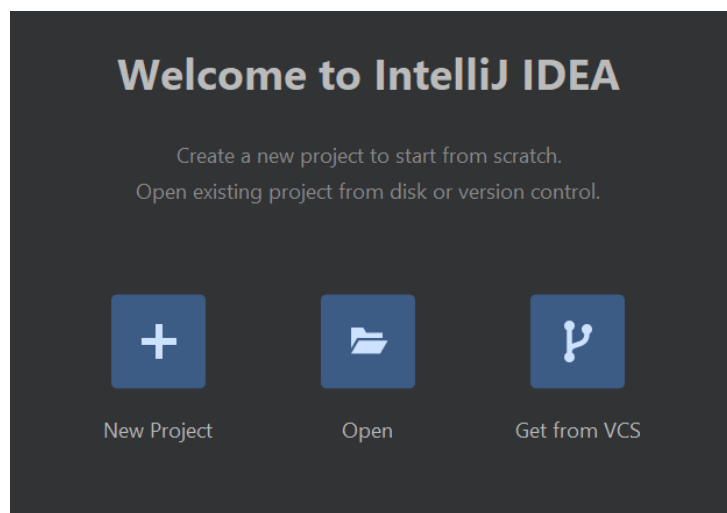
Nos bajamos un zip que vamos a descomprimir en la carpeta de proyectos del IntelliJ (o en otro sitio, pero acuérdate de dónde para abrirlo ahora desde IntelliJ):

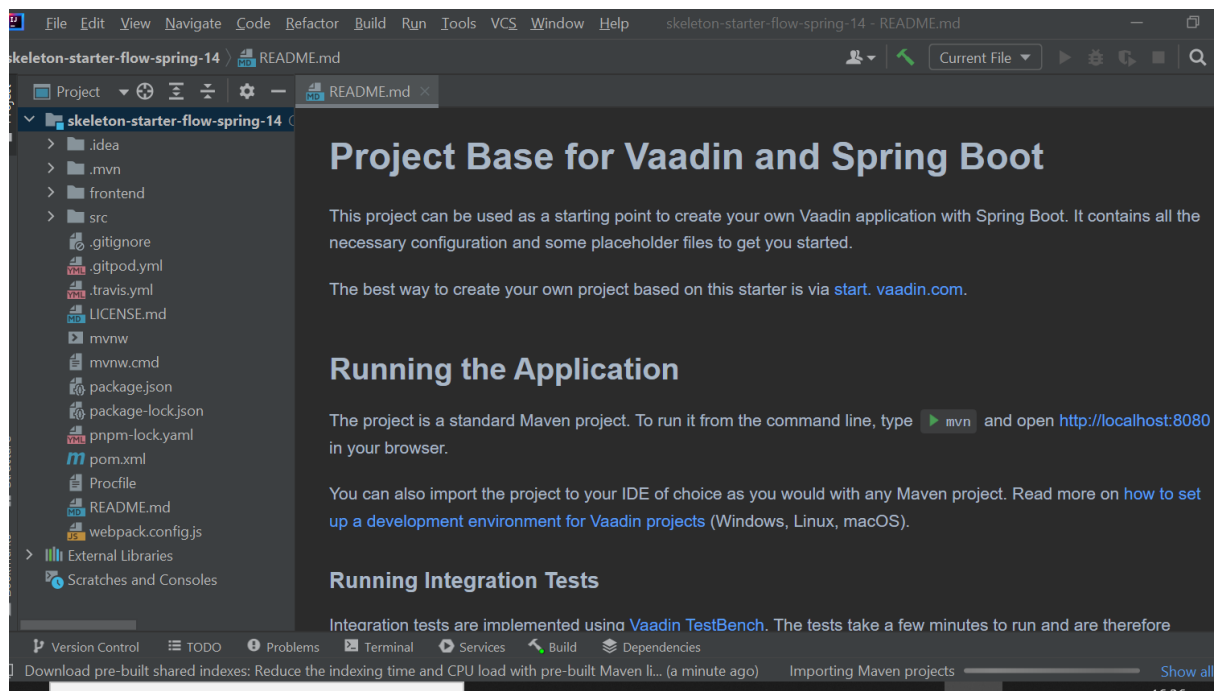
(la carpeta de proyectos de IntelliJ está por defecto en C:\Users\DIS\IdeaProjects)





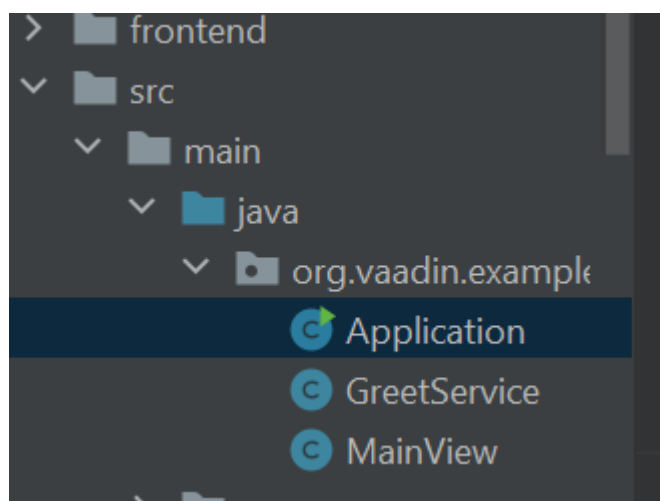
Lo abrimos desde IntelliJ usando la opción Open project





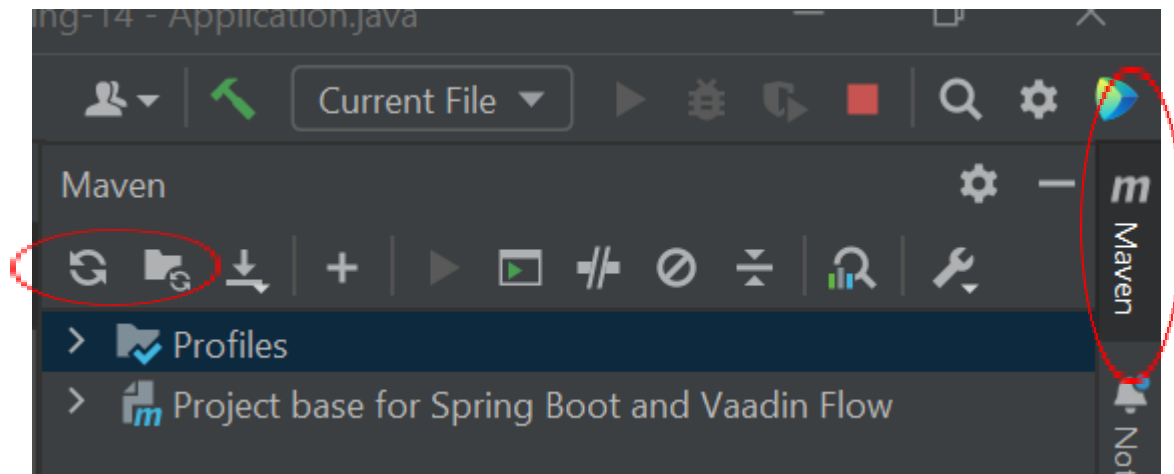
Recuerda que el proyecto puede tardar en indexar bastantes minutos, dependerá de si es la primera vez que usas IntelliJ y de tu conexión a internet.

Ejecuta el proyecto. Para ello, tienes que ir a la carpeta “src” -> “main” -> “java” -> “Application”:



y pulsar run, arriba a la derecha

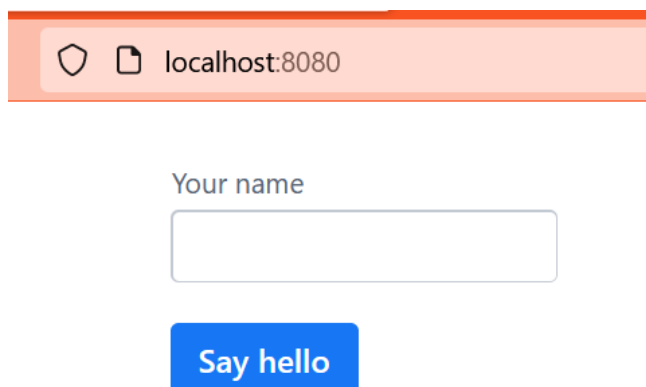
NOTA 1, si no funciona, prueba a que Maven actualice todo:



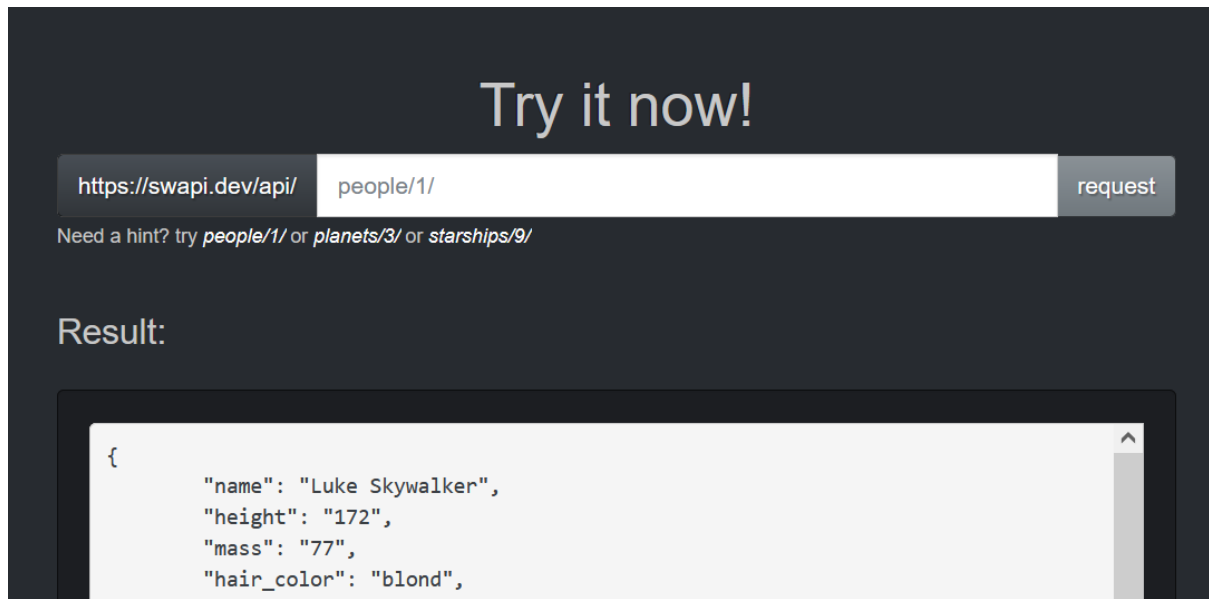
(NOTA: si sigue sin funcionar al ejecutar, prueba a reinstalar node. Cuando lo instales, marca si en las opciones de Chocolatey)



Si todo ha ido bien, tienes que ver la siguiente página:



Vamos a utilizar el API de la página swapi: <https://swapi.dev/>



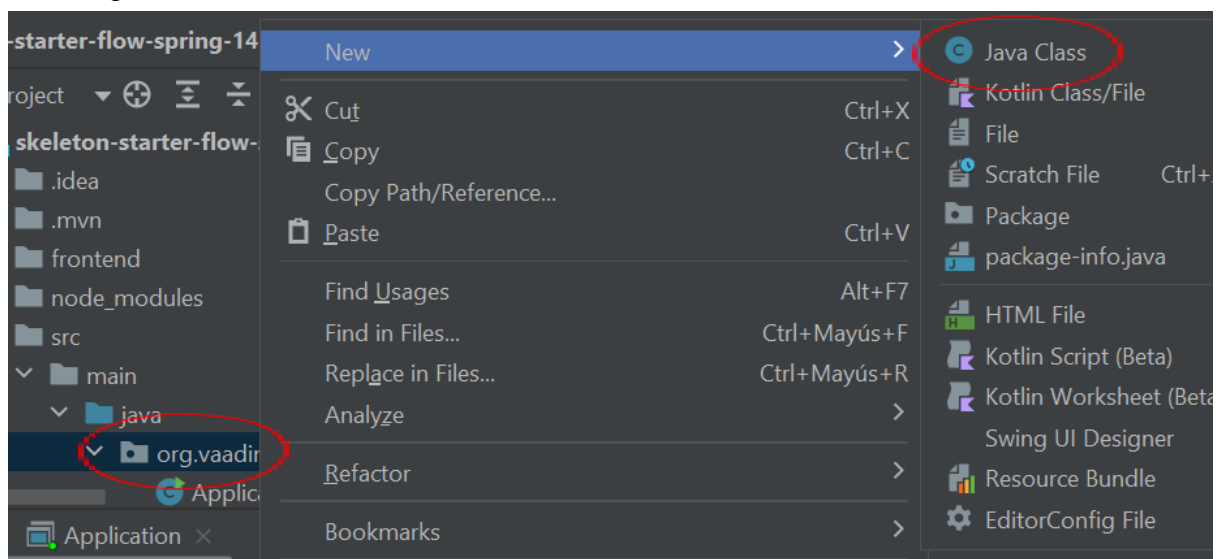
con ese api, podemos obtener datos de Star Wars sin necesidad de registro ni api key, con lo que es ideal para estos ejercicios.

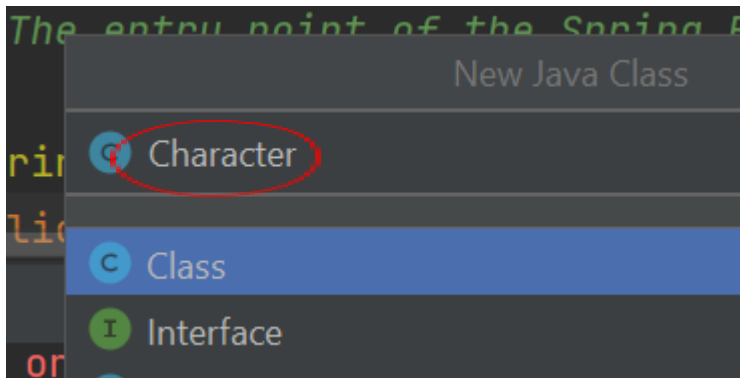
Para trabajar con los datos que vengan de ese api, vamos a crearnos una clase Java que se encargue de guardarlos adecuadamente. No hace falta que almacene todo, con que pueda guardar los 6 siguientes vale:

name, height, mass, hair_color, skin_color, eye_color

Por comodidad van a ser todos String:

Nos vamos a IntelliJ, y en la carpeta de org.vaadin.example hacemos new-> java class:

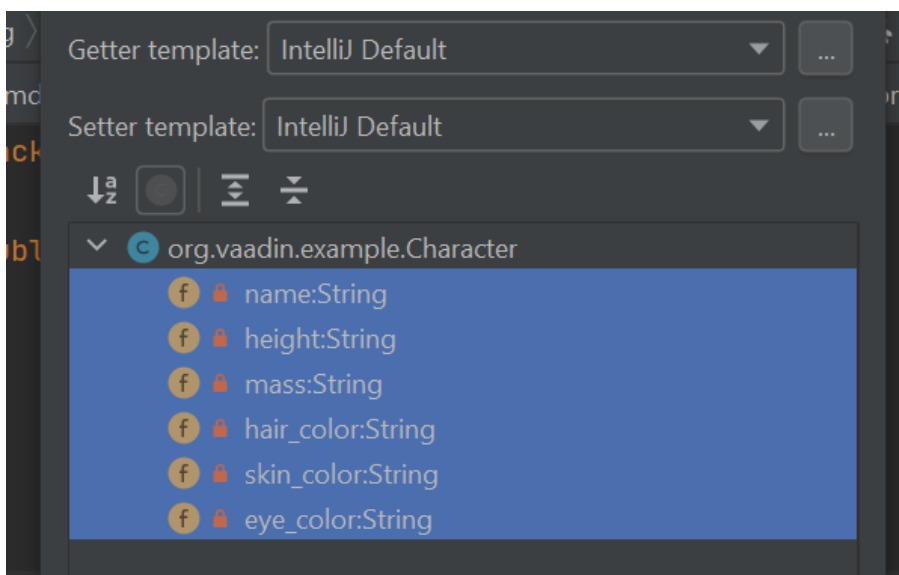




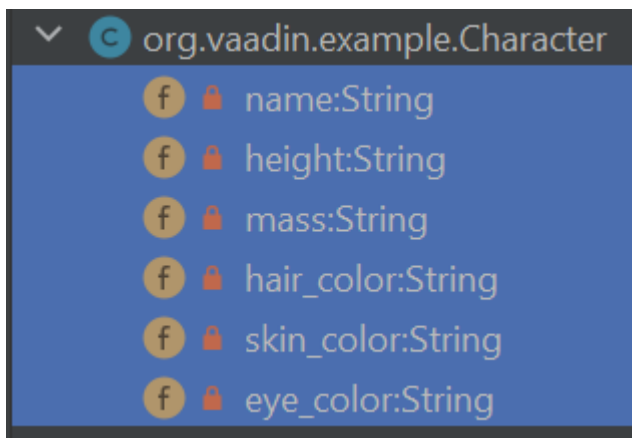
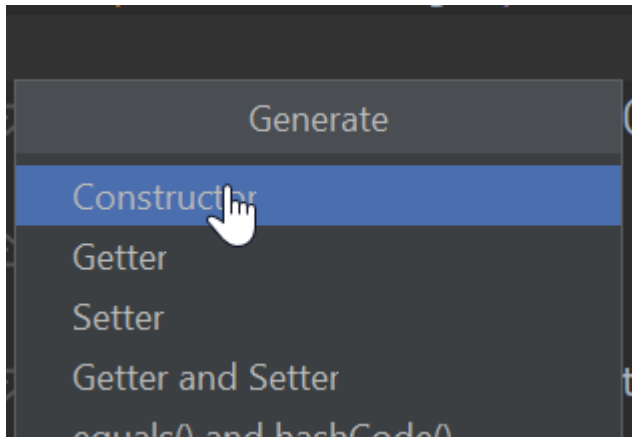
```
package org.vaadin.example;

public class Character {
    private String name;
    private String height;
    private String mass;
    private String hair_color;
    private String skin_color;
    private String eye_color;
}
```

Añadimos los getter y setter, pulsando con el botón derecho y seleccionando "Generate" -> "Getter & Setter"



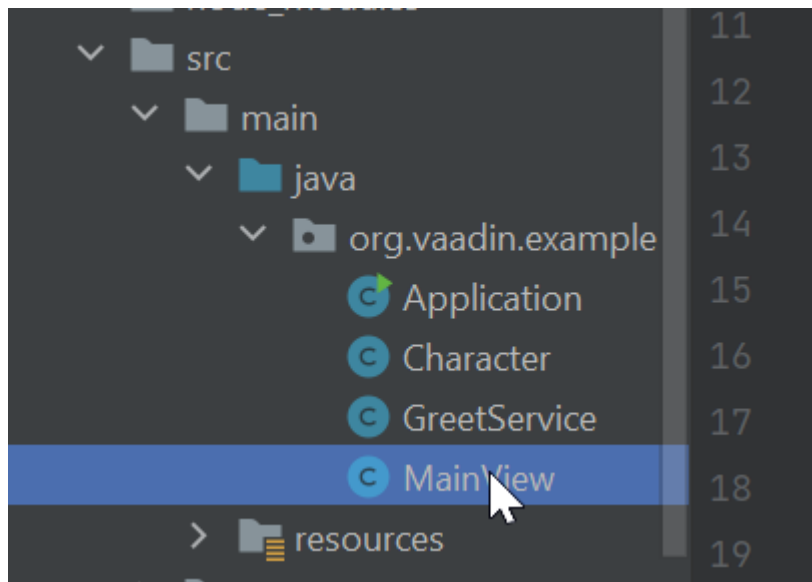
Volvemos a pulsar en “Generate”, y ahora elige “Constructor”



```
public Character(String name, String height, Str  
    this.name = name;  
    this.height = height;  
    this.mass = mass;  
    this.hair_color = hair_color;  
    this.skin_color = skin_color;  
    this.eye_color = eye_color;  
}
```

Vamos a necesitar un input y un botón en la página principal, que nos va a servir para seleccionar el personaje de Star Wars que queremos consultar.

Para ello nos vamos a la clase `MainView.java`, que está en la misma carpeta:



En esa clase, cambia el `textField` y el `button` para que se llamen distinto:

```
public MainView(@Autowired GreetService service) {

    // Use TextField for standard text input
    TextField requestType = new TextField( label: "Request type");
    requestType.addThemeName("bordered");

    // Button click listeners can be defined as lambda expressions
    Button boton1 = new Button( text: "Say hello",
        e -> Notification.show(service.greet(requestType.getValue())));

    // Theme variants give you predefined extra styles for components
    // Example: Primary button has a more prominent look.
    boton1.addThemeVariants(ButtonVariant.LUMO_PRIMARY);

    // You can specify keyboard shortcuts for buttons.
    // Example: Pressing enter in this view clicks the Button.
    boton1.addClickShortcut(Key.ENTER);

    // Use custom CSS classes to apply styling. This is defined in the theme
    addClassName("centered-content");
}
```

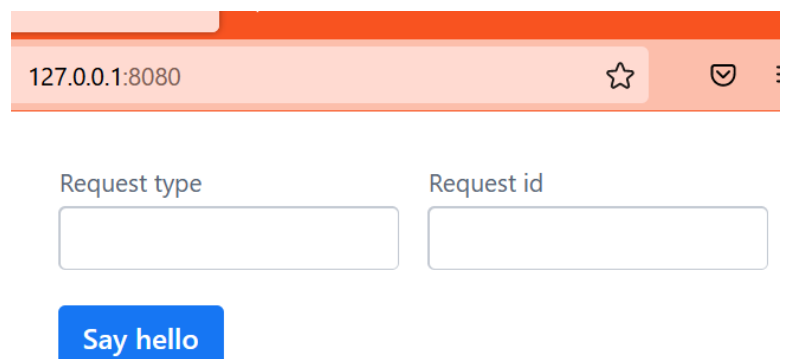
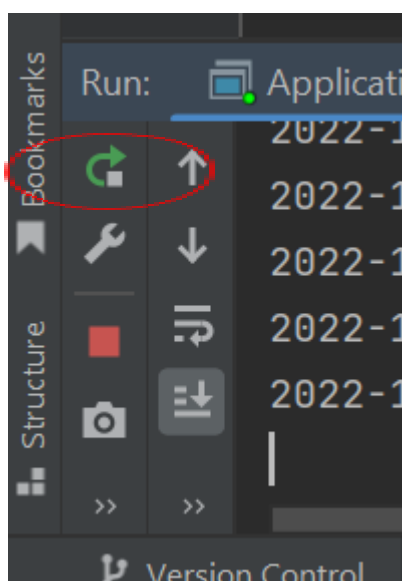

Y añade otro textfield, copiando el anterior:

```
*/
1 usage
public MainView(@Autowired GreetService service) {
    HorizontalLayout inputs = new HorizontalLayout();

    TextField requestType = new TextField( label: "Request type");
    requestType.addThemeName("bordered");
    TextField requestId = new TextField( label: "Request id");
    requestId.addThemeName("bordered");
    inputs.add(requestType, requestId);

    Button boton1 = new Button( text: "Say hello",
        e -> Notification.show(service.greet(requestType.getValue())));
    boton1.addThemeVariants(ButtonVariant.LUMO_PRIMARY);
    boton1.addClickShortcut(Key.ENTER);
    // Use custom CSS classes to apply styling. This is defined in
    addClassName("centered-content");

    add(inputs, boton1);
}
```

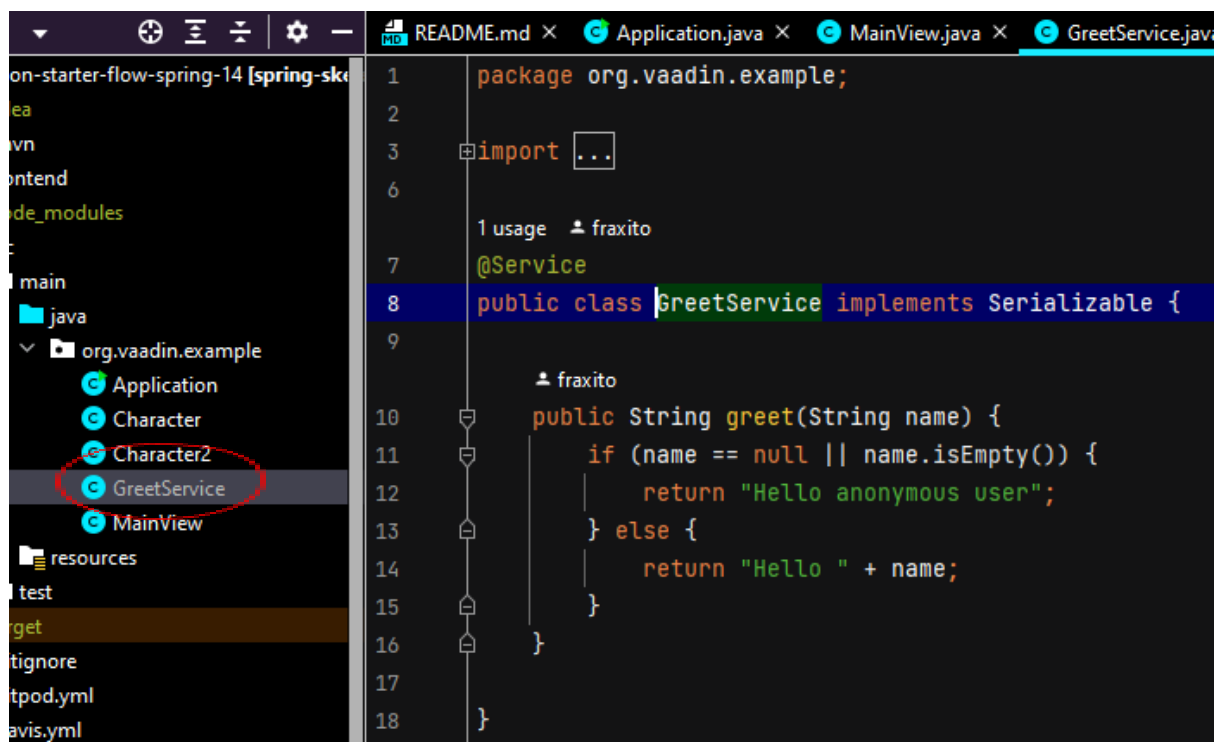


Si te has fijado, el método `MainView` recibe un objeto de la clase `GreetService`, que es el que se encarga de reaccionar al clic cuando pulsamos en el botón. Si quisiéramos que se ejecutara otro código, habría que crear otra clase. Para este ejemplo, vamos a editar la clase `GreetService`

```
1 usage  👤 fraxito
public MainView(@Autowired GreetService service) {
    HorizontalLayout inputs = new HorizontalLayout();

    TextField requestType = new TextField(label: "Request type");
    requestType.addThemeName("bordered");
    TextField requestId = new TextField(label: "Request id");
    requestId.addThemeName("bordered");
    inputs.add(requestType, requestId);
    Button boton1 = new Button(text: "Say hello",
        e -> Notification.show(service.greet(requestType.getValue())));
    boton1.addThemeVariants(ButtonVariant.LUMO_PRIMARY);
    boton1.addClickShortcut(Key.ENTER);
    // Use custom CSS classes to apply styling. This is defined in shared-styles.css.
    addClassName("centered-content");

    add(inputs, boton1);
}
```



The screenshot shows an IDE with a project structure on the left and the `GreetService.java` file open in the editor. The project structure shows the following files:

- on-starter-flow-spring-14 [spring-sk...
- ea
- vn
- ontend
- de_modules
- main
- java
- org.vaadin.example
 - Application
 - Character
 - Character2
 - GreetService
 - MainView
- resources
- test
- get
- ignore
- tpod.yml
- avis.yml

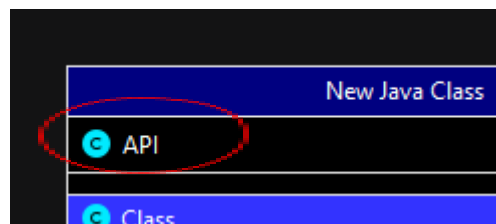
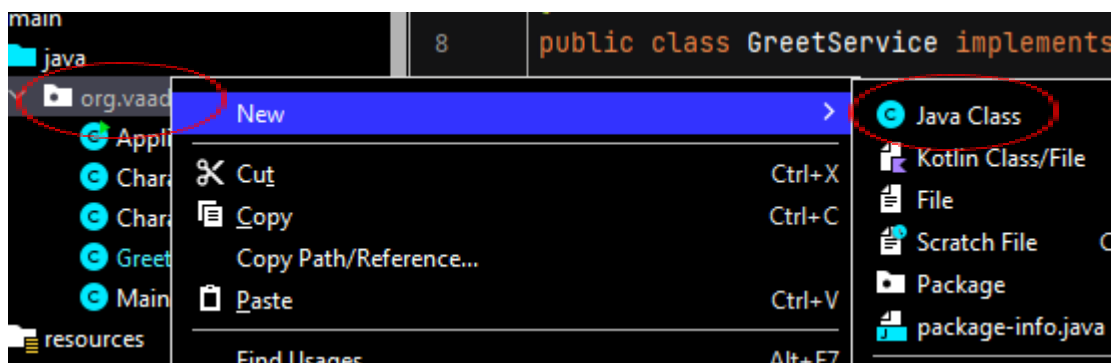
The `GreetService.java` file is shown with the following code:

```
1 package org.vaadin.example;
2
3 import ...
4
5
6
7 @Service
8 public class GreetService implements Serializable {
9
10     👤 fraxito
11     public String greet(String name) {
12         if (name == null || name.isEmpty()) {
13             return "Hello anonymous user";
14         } else {
15             return "Hello " + name;
16         }
17     }
18 }
```

Añade el siguiente método en esa clase, después del método greet:

```
}  
  
new *  
public String getSWAPI(int id) {  
  
    return "";  
}  
  
}
```

Y ahora para que ese get funcione, vamos a crear una clase API que se encargue de hacer las llamadas:



Añade este código:

```

package org.vaadin.example;

import java.net.URI;
import java.net.http.HttpRequest;

public class API {
    1 usage
    private static final String urlPrefix = "https://swapi.dev/api/%s/%s";

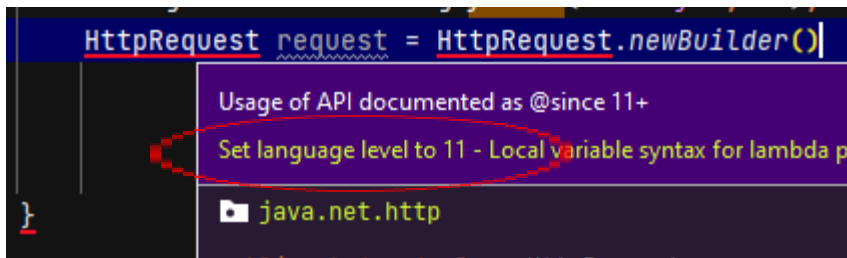
    public String getCharacter(String tipo, int id) throws URISyntaxException,
        String fullUrl = String.format(urlPrefix, tipo, id);
        HttpRequest request = HttpRequest.newBuilder()
            .uri(new URI(fullUrl))

```

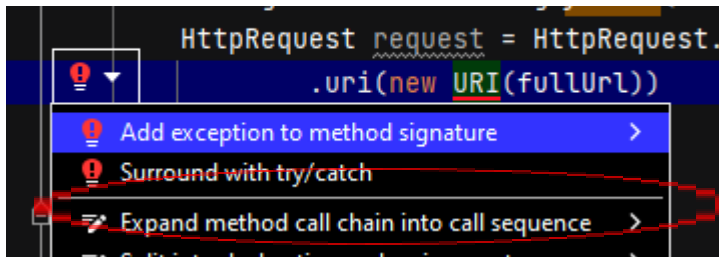
el urlPrefix es la parte común en cada petición que hagamos al api de swapi. Por cada %, el método String.format va a “pegar” correctamente el dato que pasemos como parámetro a la URL.

En el método getCharacter, tienes que importar el HttpRequest del paquete java.net

Y para arreglar el error de HttpRequest, te pones encima:

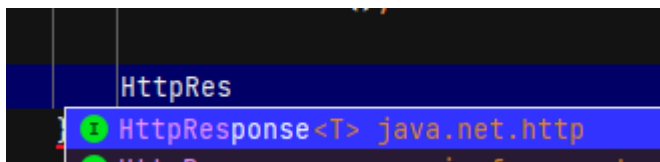


A continuación, tienes que manejar los posibles errores que pueda arrojar URI. Podrías hacerlo con un try-catch, pero en este caso vamos a usar throws:



```
public String getCharacter(String tipo, int id) throws URISyntaxException  
String fullUrl = String.format(urlPrefix, tipo, id);  
HttpRequest request = HttpRequest.newBuilder()  
    .uri(new URI(fullUrl))  
    .GET()  
    .build();
```

Seguimos, añadiendo un `HttpResponse`:

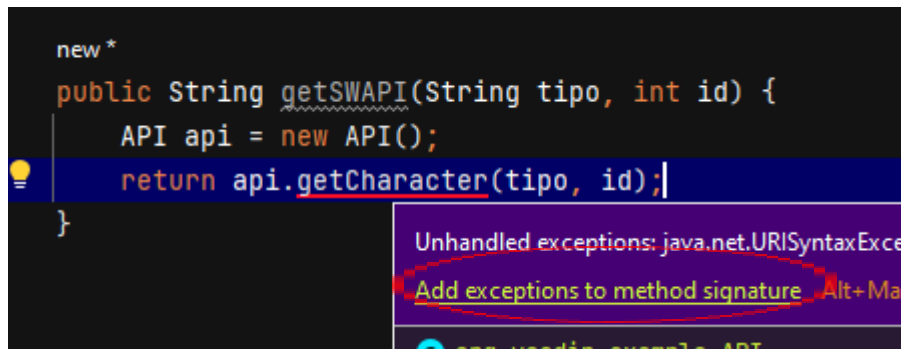


Para el `send`, tenemos que añadir otro `throws` como el anterior

```
public String getCharacter(String tipo, int id) throws URISyntaxException, IOException, InterruptedException {  
    String fullUrl = String.format(urlPrefix, tipo, id);  
    HttpRequest request = HttpRequest.newBuilder()  
        .uri(new URI(fullUrl))  
        .GET()  
        .build();  
  
    HttpResponse<String> response = HttpClient  
        .newBuilder()  
        .build()  
        .send(request, HttpResponse.BodyHandlers.ofString());  
  
    System.out.println(response.body());  
    return response.body();  
}
```

Nos volvemos a la clase GreetService:

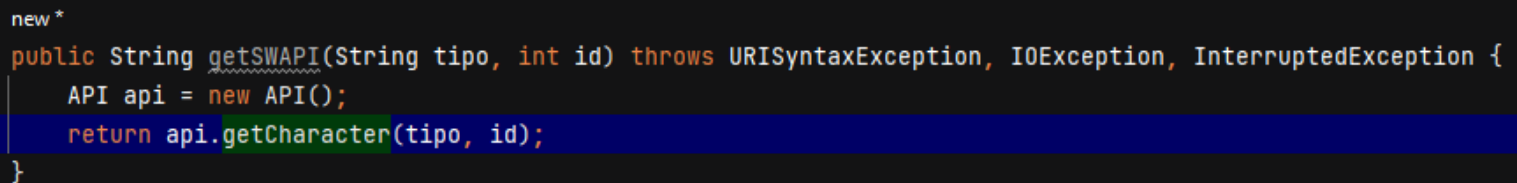
y modificamos el código del método getSWAPI:



```
new *
public String getSWAPI(String tipo, int id) {
    API api = new API();
    return api.getCharacter(tipo, id);
}
```

Unhandled exceptions: java.net.URISyntaxException
Add exceptions to method signature Alt+Ma
org.vaadin.example.API

y así es como queda:



```
new *
public String getSWAPI(String tipo, int id) throws URISyntaxException, IOException, InterruptedException {
    API api = new API();
    return api.getCharacter(tipo, id);
}
```

Ahora volvemos a la clase MainView, para añadir el código que hará que se muestre la llamada:

1 usage  fraxito *

```
public MainView(@Autowired GreetService service) {  
    HorizontalLayout inputs = new HorizontalLayout();  
    VerticalLayout results = new VerticalLayout();  
    TextField requestType = new TextField(label: "Request type");  
    requestType.addThemeName("bordered");
```

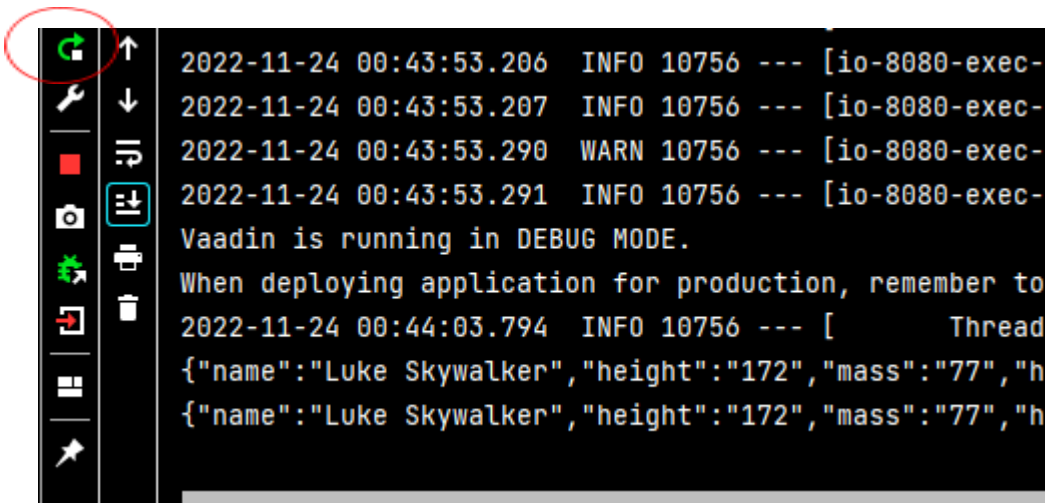
Cambia el código del botón para que quede así

```
Button boton1 = new Button(text: "Lee caracter",  
    e -> {  
        String tipo = requestType.getValue();  
        int id = Integer.parseInt(requestId.getValue());  
        try {  
            System.out.println(service.getSWAPI(tipo,id));  
        } catch (Exception ex) {  
        }  
    });
```

Ahora si ponemos :

Request type	Request id
<input type="text" value="people"/>	<input type="text" value="1"/>
<input type="button" value="Lee caracter"/>	

obtenemos la siguiente salida en el terminal del run:
(recuerda recargar el run)



```
2022-11-24 00:43:53.206 INFO 10756 --- [io-8080-exec-
2022-11-24 00:43:53.207 INFO 10756 --- [io-8080-exec-
2022-11-24 00:43:53.290 WARN 10756 --- [io-8080-exec-
2022-11-24 00:43:53.291 INFO 10756 --- [io-8080-exec-
Vaadin is running in DEBUG MODE.
When deploying application for production, remember to
2022-11-24 00:44:03.794 INFO 10756 --- [      Thread
{"name":"Luke Skywalker","height":"172","mass":"77","h
{"name":"Luke Skywalker","height":"172","mass":"77","h
```

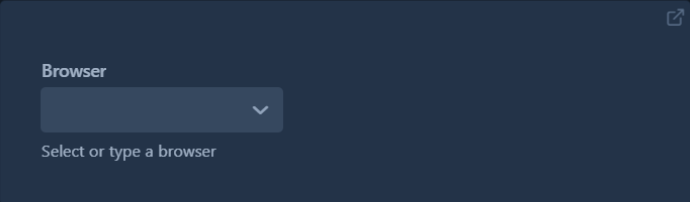

Segunda parte: Añadir comboBox en Vaadin 14

Vamos a copiar el código para generar un combobox que hay en la web de Vaadin: <https://vaadin.com/docs/v14/ds/components/combo-box>

Concretamente, vamos a usar el que pone “Custom Value Entry”

Custom Value Entry

Combo Box can be configured to allow entering custom values that are not included in the list of options.



^ Hide code

```
ComboBoxCustomEntry1.java
package com.vaadin.demo.component.combobox;

import com.vaadin.flow.component.combobox.ComboBox;
import com.vaadin.flow.component.html.Div;
import com.vaadin.flow.router.Route;

@Route("combo-box-custom-entry-1")
public class ComboBoxCustomEntry1 extends Div {

    public ComboBoxCustomEntry1() {
        ComboBox<String> comboBox = new ComboBox<>("Browser");
        comboBox.setAllowCustomValue(true);
        add(comboBox);
        comboBox.setItems("Chrome", "Edge", "Firefox", "Safari");
        comboBox.setHelperText("Select or type a browser");
    }

}
```

```
ComboBox<String> comboBox = new ComboBox<>("Browser");
comboBox.setAllowCustomValue(true); //este deja que el usuario
escriba lo que quiera en la caja del combobox. Si se pone a false no
deja
comboBox.setItems("people", "planets", "starships");
comboBox.setHelperText("Select or type a browser");
```

```

public MainView(@Autowired GreetService service) {
    HorizontalLayout inputs = new HorizontalLayout();
    VerticalLayout results = new VerticalLayout();
    ComboBox<String> comboBox = new ComboBox<>("Browser");
    comboBox.setAllowCustomValue(false); //este deja que el usuario
    comboBox.setItems("people", "planets", "starships");
    comboBox.setHelperText("Selecciona el tipo de petición");

    TextField requestId = new TextField("Request id");
    requestId.addThemeName("bordered");
    inputs.add(comboBox, requestId);
    Button boton1 = new Button("Lee character",
        e -> {
            String tipo = comboBox.getValue();
            int id = Integer.parseInt(requestId.getValue());
            try {
                System.out.println(service.getSWAPI(tipo, id));
            } catch (Exception ex) {
            }
        });
    boton1.addThemeVariants(ButtonVariant.LUMO_PRIMARY);
}

```

Para hacer que los resultados se muestren en un verticalLayout (en modo JSON a palo seco) vamos a declarar el siguiente componente (si no lo has hecho ya)

```

public MainView(@Autowired GreetService service) {
    HorizontalLayout inputs = new HorizontalLayout();
    VerticalLayout results = new VerticalLayout();
}

```

y en el código del botón, dejamos lo siguiente en el try:

```
Button boton1 = new Button( text: "Lee caracter",
    e -> {
        String tipo = comboBox.getValue();
        int id = Integer.parseInt(requestId.getValue());
        try {
            results.removeAll(); //para borrar lo que hubiera
            results.add(service.getSWAPI(tipo,id));
        } catch (Exception ex) {
        }
    });
```

y añade el `verticalLayout` al `add` final

```
        } catch (Exception ex) {
        }
    });

    boton1.addThemeVariants(ButtonVariant.LUMO_PRIMARY);
    boton1.addClickShortcut(Key.ENTER);
    // Use custom CSS classes to apply styling. The
    addClassName("centered-content");

    add(inputs, boton1, results);
```

Browser

Request id

starships

2

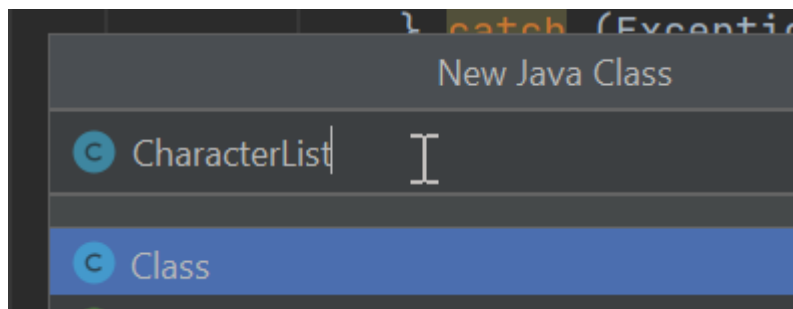
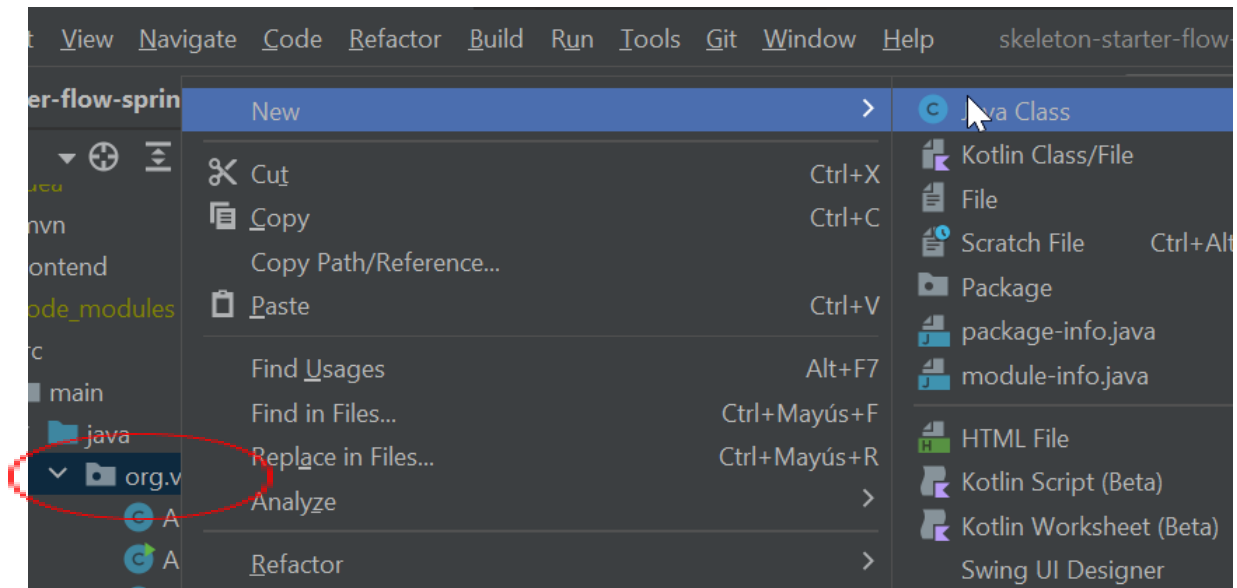
Selecciona el tipo de petición

Lee character

```
{
  "name": "CR90 corvette",
  "model": "CR90 corvette",
  "manufacturer": "Corellian Engineering Corporation",
  "cost_in_credits": 3500000,
  "length": 150,
  "max_atmosphering_speed": 950,
  "hyperdrive_rating": 2.0,
  "MGLT": 60,
  "starship_class": "corvette",
  "pilots": [
    {
      "name": "Wedge Antilles",
      "species": "Human",
      "films": [
        "The Force Awakens"
      ]
    }
  ],
  "created": "2014-12-10T14:20:33.369000Z",
  "edited": "2014-12-20T14:20:33.369000Z"
}
```

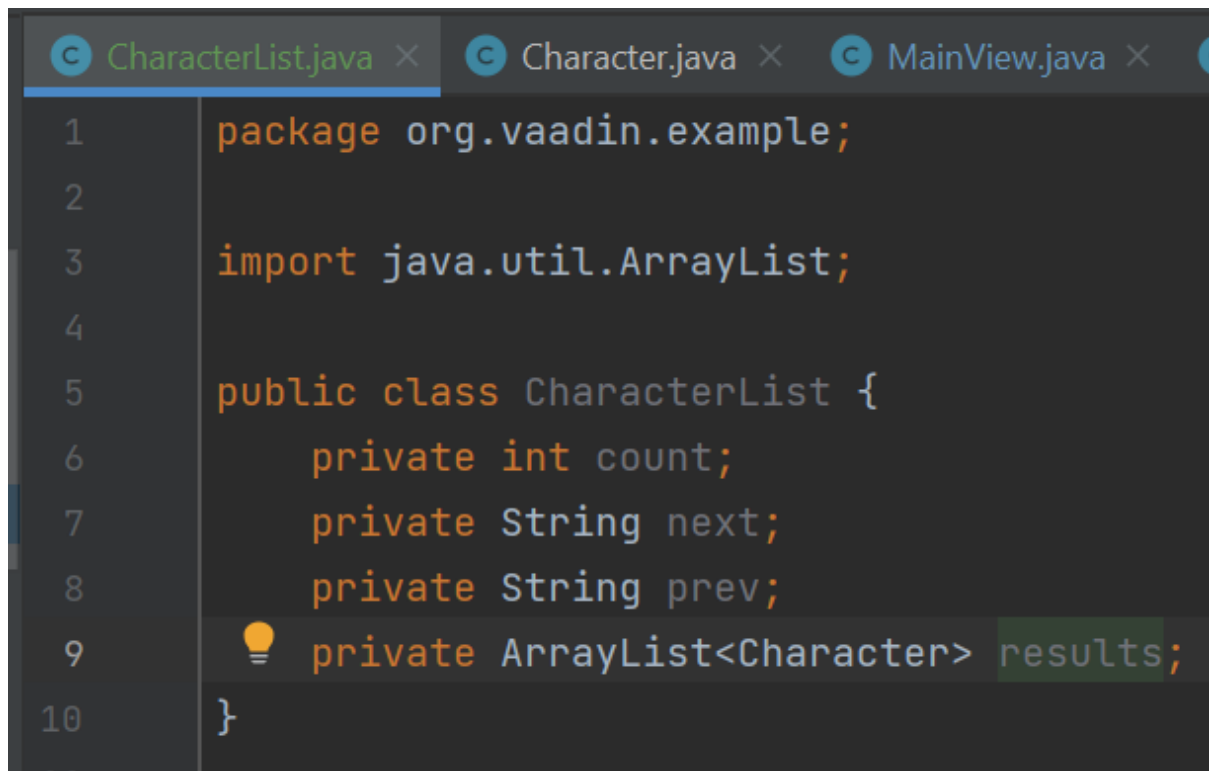
3ª parte: Vaadin Grid y Data

Para la 3ª parte, vamos a crear una clase que gestione los datos que recibimos del JSON.



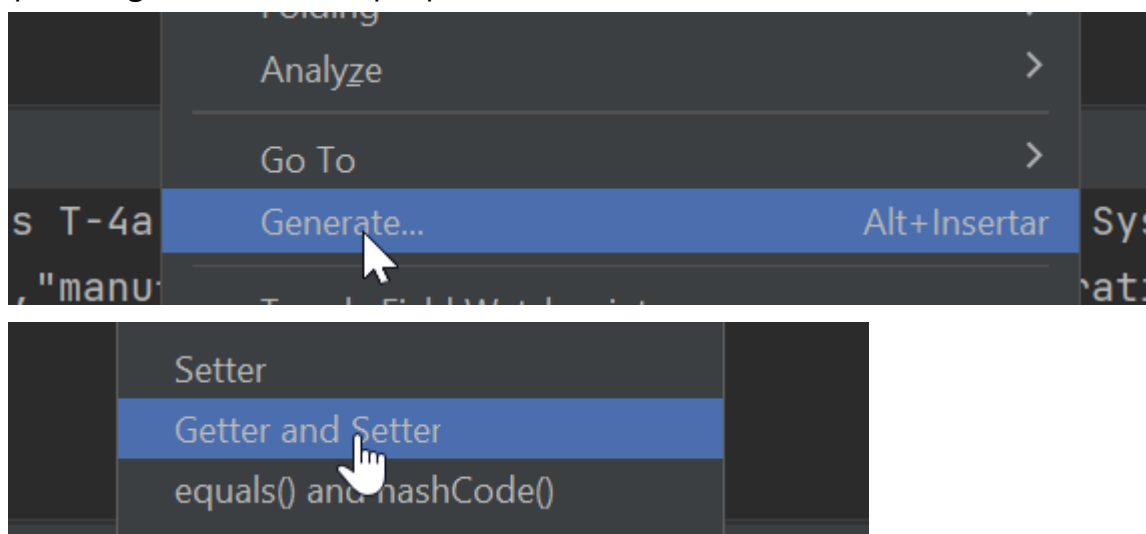
va a guardar una lista de caracteres de los que leemos del swapi

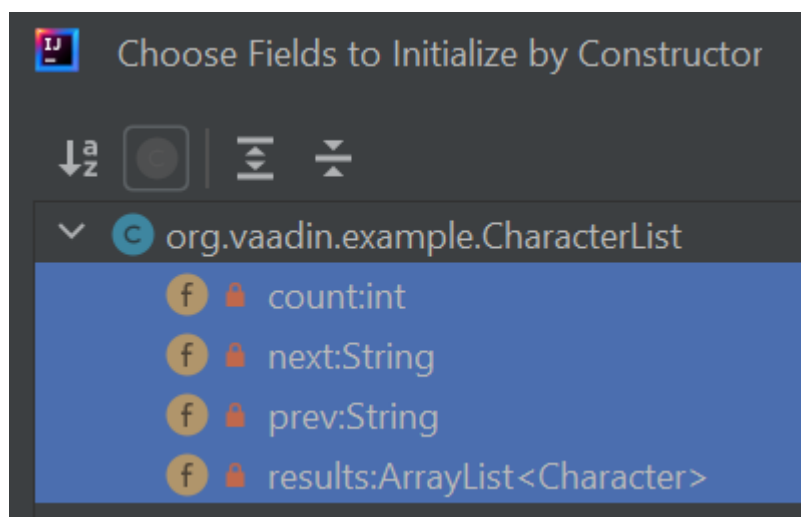
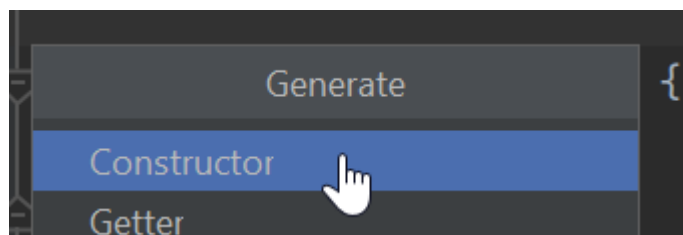
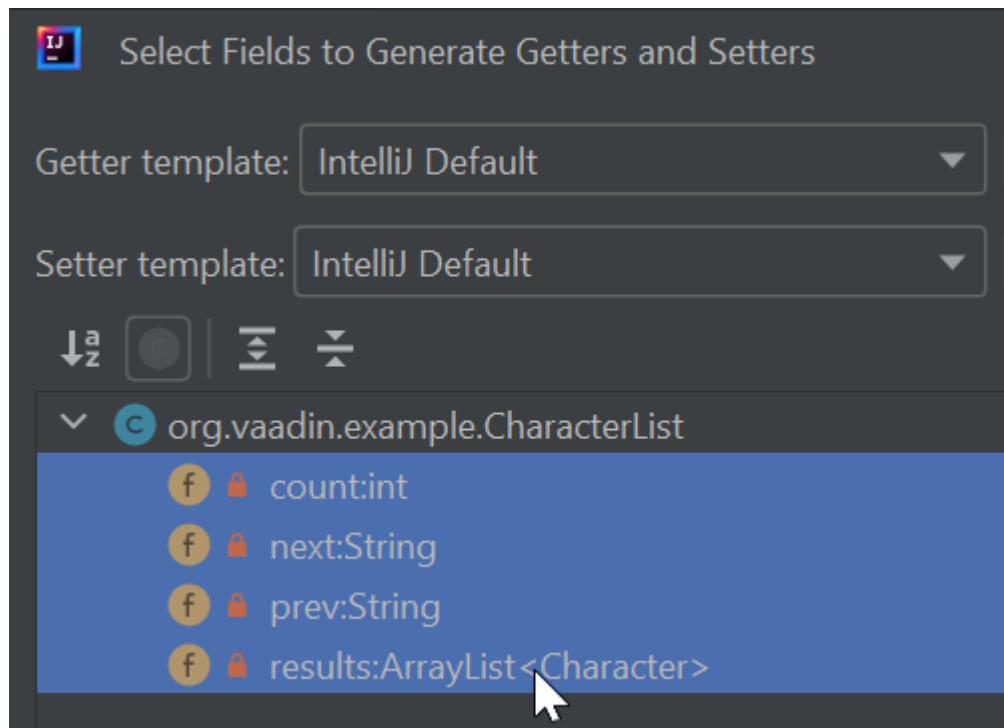
tiene los siguientes campos:



```
1 package org.vaadin.example;
2
3 import java.util.ArrayList;
4
5 public class CharacterList {
6     private int count;
7     private String next;
8     private String prev;
9     private ArrayList<Character> results;
10 }
```

Haz los getter y setter de las cuatro propiedades, y un constructor que tenga las cuatro propiedades:





```
public class CharacterList {  
    3 usages  
    private int count;  
    3 usages  
    private String next;  
    3 usages  
    private String prev;  
    3 usages  
    private ArrayList<Character> results;  
  
    public CharacterList(int count, String next,  
        this.count = count;  
        this.next = next;  
        this.prev = prev;  
        this.results = results;  
}
```

Nos vamos a la clase API, para crear un lector del api que permita leer de golpe todos los caracteres:

Copia y pega el método getCharacter, y cambia lo siguiente:


```

public String getCharacterList(String tipo) throws URISyntaxException
    String fullUrl = String.format(urlPrefix, tipo, "");
    HttpRequest request = HttpRequest.newBuilder()
        .uri(new URI(fullUrl))
        .GET()
        .build();

    HttpResponse<String> response = HttpClient
        .newBuilder()
        .build()
        .send(request, HttpResponse.BodyHandlers.ofString());

    return response.body();
}

```

Nos vamos a la clase GreetService, y copiamos el método getSWAPI, modificando lo siguiente:

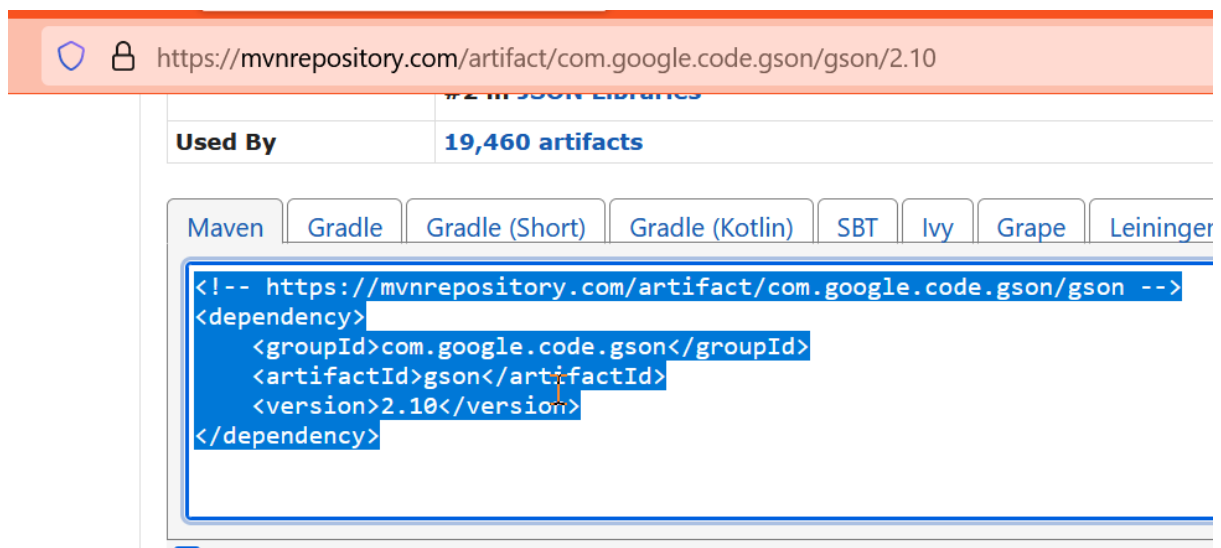
```

1 usage  Jorge Cisneros
public String getSWAPI(String tipo, int id) throws URISyntaxException
    API api = new API();
    return api.getCharacter(tipo, id);
}
new *
public ArrayList<Character> getCharList(String tipo) throws URISy
    API api = new API();
    String resultsAPI = api.getCharacterList(tipo);
}

```

y ahora para poder seguir, necesitamos importar GSON para que pueda funcionar el JSON parser de Google. Para ello, editamos el pom.xml y añadimos la siguiente dependencia:

```
<!-- https://mvnrepository.com/artifact/com.google.code.gson/gson -->
<dependency>
  <groupId>com.google.code.gson</groupId>
  <artifactId>gson</artifactId>
  <version>2.10</version>
</dependency>
```



recuerda que hay que refrescar Maven:



Con GSON cargado, vamos a usarlo en la clase GreetService:

```
public ArrayList<Character> getCharList(String tipo) throws URISyntaxException,
    API api = new API();
    String resultsAPI = api.getCharacterList(tipo);
    Gson gson = new Gson();
    CharacterList charList = gson.fromJson(resultsAPI, CharacterList.class);
    return charList.getResults();
}
```

Ahora en el MainView, vamos a añadir un Grid para mostrar los resultados de la consulta a la Api. Un grid es el típico display en forma de tabla. En la documentación de Vaadin puedes encontrar un ejemplo:

<https://vaadin.com/docs/v14/ds/components/grid>

```
Grid<Character> grid = new Grid<>(Character.class, false);
grid.addColumn(Character::getName).setHeader("Nombre");
grid.addColumn(Character::getHeight).setHeader("Altura");
grid.addColumn(Character::getMass).setHeader("Peso");
grid.addColumn(Character::getHair_color).setHeader("Color de pelo");
grid.addColumn(Character::getEye_color).setHeader("Color de pelo");
```

Lo añadimos después del comboBox

```
comboBox.setHelperText("Selecciona el tipo de petición");

Grid<Character> grid = new Grid<>(Character.class, autoCreateColumns: false);
grid.addColumn(Character::getName).setHeader("Nombre");
grid.addColumn(Character::getHeight).setHeader("Altura");
grid.addColumn(Character::getMass).setHeader("Peso");
grid.addColumn(Character::getHair_color).setHeader("Color de pelo");
grid.addColumn(Character::getEye_color).setHeader("Color de pelo");
```

Hacemos un segundo botón, para llamar al nuevo código. Copia el primero y modifícalo para que quede así:

```
Button boton2 = new Button( text: "Lee lista de caracteres",
    e -> {
        String tipo = comboBox.getValue();
        try {
            //añadimos al grid los datos que lea
            grid.setItems(service.getCharList(tipo));
            results.add(grid);
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    });
```

y no te olvides de añadir el botón 2 al final:

```
add(inputs, boton1, boton2, results);
}
```

Queda algo tal que así, recuerda que hay que seleccionar un tipo para que funcione:

Browser

people

Seleccióna el tipo de petición

Request id

Lee caracter

Lee lista de caracteres

Nombre	Altura
Luke Sk...	172
C-3PO	167



