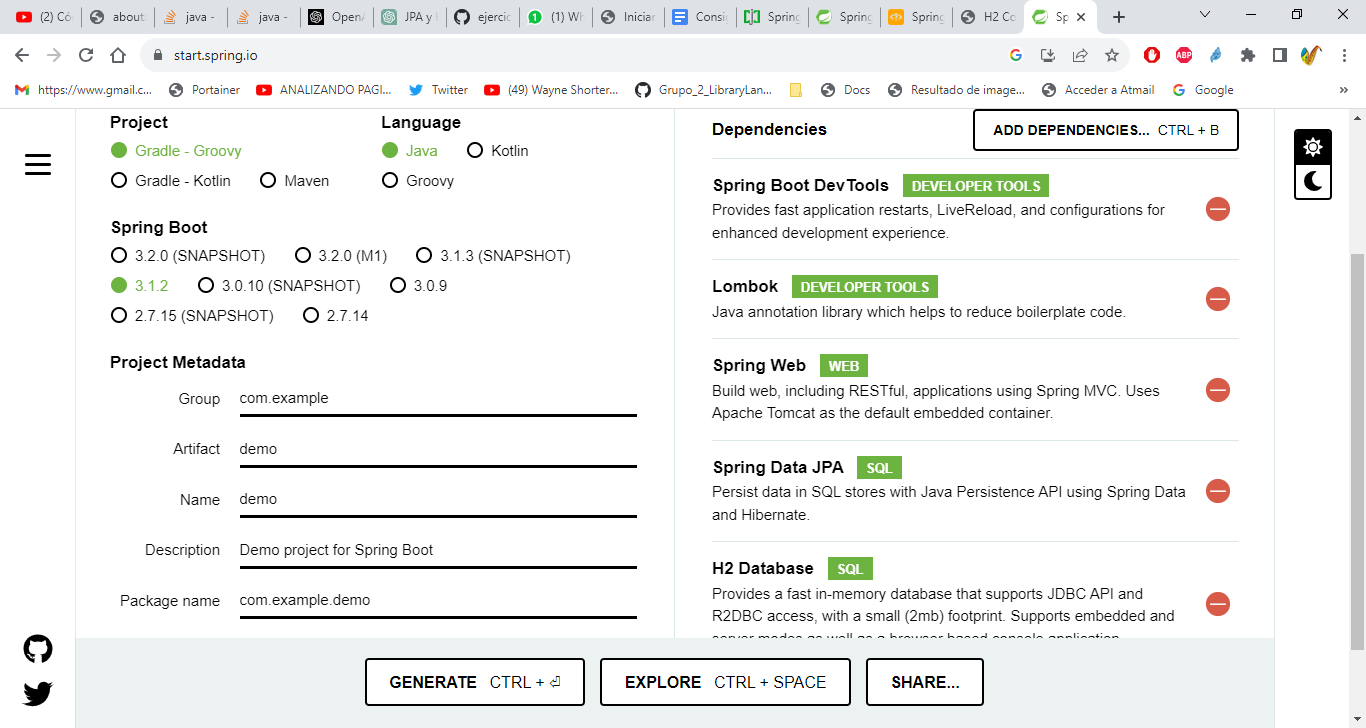
Desarrollo de Software

Persistencia con Hibernate:

1. Configuración inicial

Sitio: <https://start.spring.io/>



**Recomendaciones:**

**Levantar la base de datos en memoria con el siguiente link:**

***http://localhost:8080/h2-console/***

**Utilizar la siguiente configuración en el aplication.properties**

*# Configuraci?n de la base de datos H2  
#Habilitar la consola de H2*spring.h2.console.enabled=true  
*#Parámetros para conectarnos a una base d edatos*spring.datasource.url=jdbc:h2:mem:./betodb;  
spring.datasource.driverClassName=org.h2.Driver  
spring.datasource.username=sa  
spring.datasource.password=  
  
*# Configuraci?n de JPA*spring.jpa.database-platform=org.hibernate.dialect.H2Dialect  
*# EL update le dice que si no existe n las tablas que las cree y luego actualice*spring.jpa.hibernate.ddl-auto=create  
*#Muestra pr la consola como se desarrollan las consultas SQL*spring.jpa.show-sql = true  
  
*# puerto de ejecución, donde escucha el server embebido*server.port=8080  
  
  
*# OPEN API  
# http://localhost:8080/swagger-ui/index.html ASI HAY QUE LLAMARLO*springdoc.swagger-ui.enabled=true  
springdoc.api-docs.enabled=true  
springdoc.swagger-ui.path=/swagger-ui.html  
springdoc.api-docs.path=/api-docs  
  
*# LEVANTAR H2  
# http://localhost:8080/h2-console/*

**Información Util: Para qué sirve el método CommandLineRunner?**

La interfaz CommandLineRunner es parte del framework Spring Boot y se utiliza para ejecutar código después de que la aplicación Spring Boot se haya iniciado. Proporciona un mecanismo conveniente para realizar tareas específicas al comienzo de la ejecución de la aplicación.

La interfaz CommandLineRunner tiene un método llamado run, que acepta como argumentos los argumentos de línea de comandos pasados a la aplicación. Puedes implementar este método para ejecutar tareas iniciales, configuraciones, carga de datos o cualquier otra actividad que desees realizar al inicio de la aplicación.

En resumen, CommandLineRunner es útil cuando necesitas ejecutar cierto código justo después de que la aplicación Spring Boot se haya iniciado. Puedes usarlo para configurar inicializaciones, cargar datos iniciales en la base de datos, ejecutar tareas programadas, entre otros casos de uso.

Por ejemplo, en el método main de tu clase principal, puedes agregar un bean CommandLineRunner para ejecutar el código deseado:

java

Copy code

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

@SpringBootApplication

public class MainApplication {

public static void main(String[] args) {

SpringApplication.run(MainApplication.class, args);

}

@Bean

public CommandLineRunner init() {

return args -> {

// Código a ejecutar después de la inicialización de la aplicación

System.out.println("La aplicación se ha iniciado. Realizando tareas iniciales...");

};

}

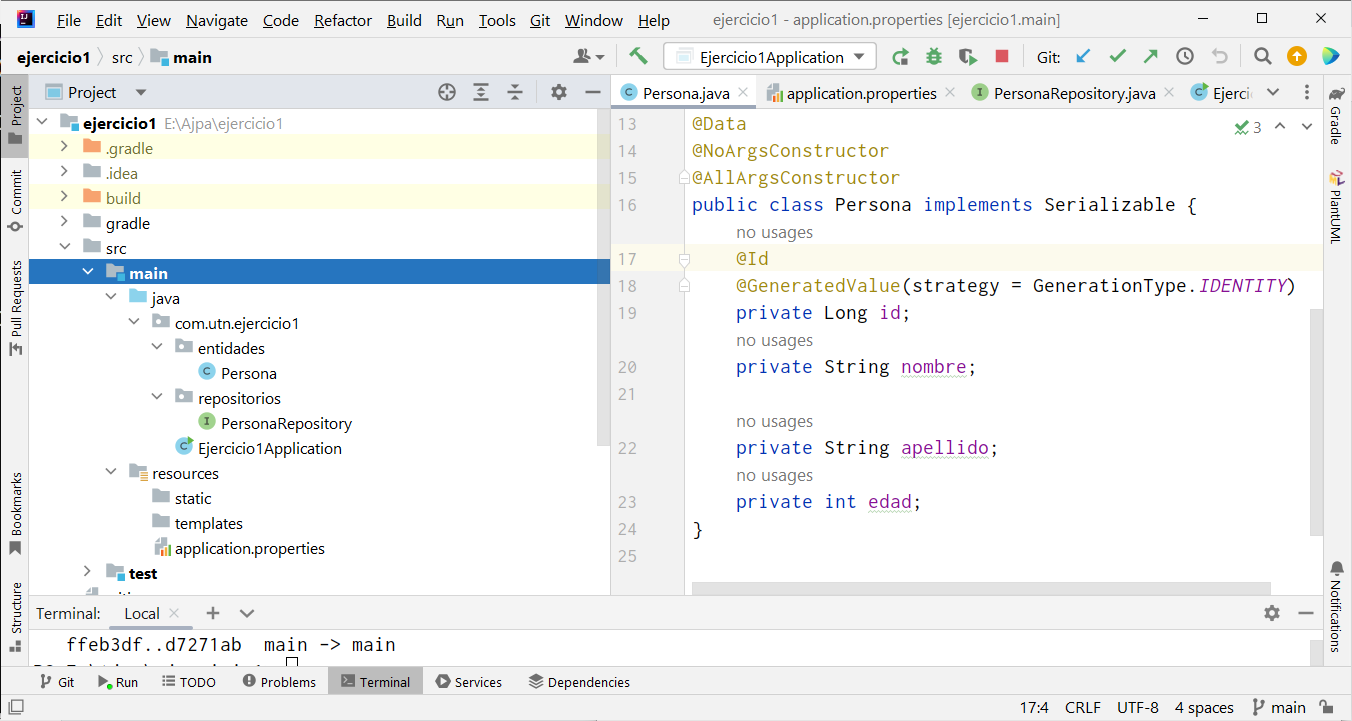
}

En este ejemplo, la aplicación ejecutará el código en el método run de CommandLineRunner después de que la aplicación se haya iniciado. Puedes agregar tantos beans CommandLineRunner como necesites para realizar diferentes tareas al inicio de la aplicación.

**Ejercicio 1:**

Dada la siguiente entidad Persona, realizar la persistencia con un ORM

**Estructura del Proyecto**



**Clase Persistente Persona:**

import java.io.Serializable;  
@Entity  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class Persona implements Serializable {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 private String nombre;  
  
 private String apellido;  
 private int edad;  
}

**Repositorio de Persona:**

package com.utn.ejercicio1.repositorios;  
  
import com.utn.ejercicio1.entidades.Persona;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface PersonaRepository extends JpaRepository <Persona,Long>{  
}

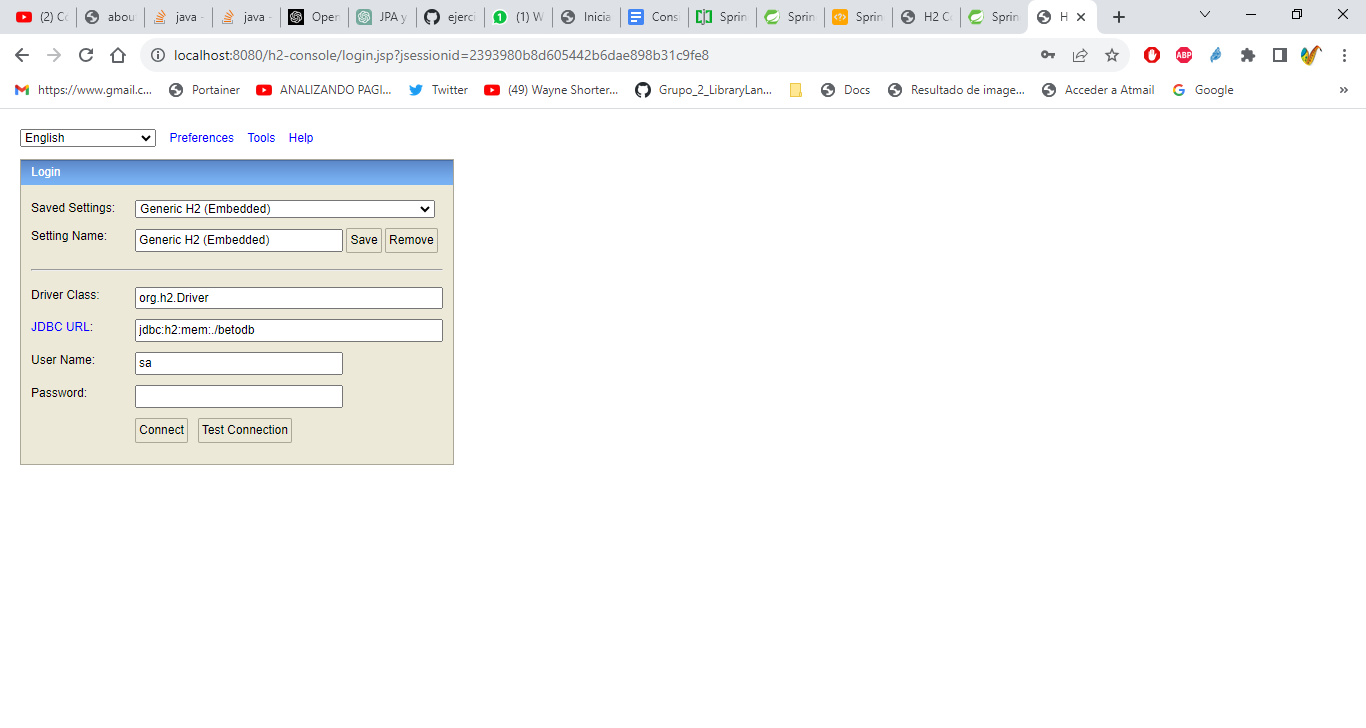
**Archivo de aplication properties:**

*Configuraci?n de la base de datos H2  
#Habilitar la consola de H2*spring.h2.console.enabled=true  
*#Parámetros para conectarnos a una base d edatos*spring.datasource.url=jdbc:h2:mem:./betodb;  
spring.datasource.driverClassName=org.h2.Driver  
spring.datasource.username=sa  
spring.datasource.password=  
  
*# Configuraci?n de JPA*spring.jpa.database-platform=org.hibernate.dialect.H2Dialect  
*# EL update le dice que si no existe n las tablas que las cree y luego actualice*spring.jpa.hibernate.ddl-auto=update  
*#Muestra pr la consola como se desarrollan las consultas SQL*spring.jpa.show-sql = true  
  
*# puerto de ejecución, donde escucha el server embebido  
#server.port=8080  
  
  
# OPEN API  
# http://localhost:8080/swagger-ui/index.html ASI HAY QUE LLAMARLO*springdoc.swagger-ui.enabled=true  
springdoc.api-docs.enabled=true  
springdoc.swagger-ui.path=/swagger-ui.html  
springdoc.api-docs.path=/api-docs  
  
*# LEVANTAR H2  
# http://localhost:8080/h2-console/*

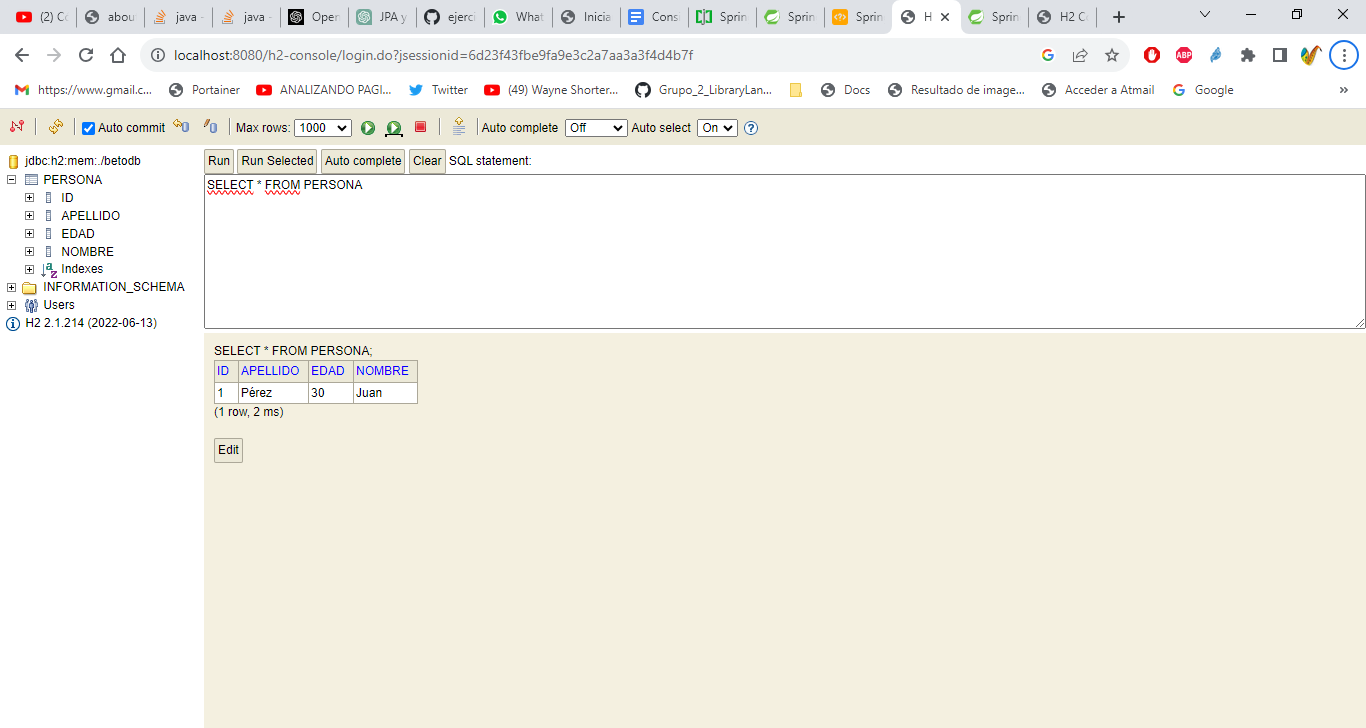
**Main de ejecución :**

package com.utn.ejercicio1;  
  
import com.utn.ejercicio1.entidades.Persona;  
import com.utn.ejercicio1.repositorios.PersonaRepository;  
import jakarta.annotation.PostConstruct;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.CommandLineRunner;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;import org.springframework.context.ConfigurableApplicationContext;  
import org.springframework.context.annotation.Bean;  
  
  
  
@SpringBootApplication  
public class Ejercicio1Application {  
 @Autowired  
 PersonaRepository personaRepository;  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(Ejercicio1Application.class, args);  
  
  
  
 }  
  
  
  
 @Bean  
 CommandLineRunner init(PersonaRepository personaRepo) {  
 return args -> {  
 System.*out*.println("-----------------ESTOY FUNCIONANDO---------");  
 Persona persona = new Persona();  
 persona.setNombre("Juan");  
 persona.setApellido("Pérez");  
 persona.setEdad(30);  
  
 *// Guardar el objeto Persona en la base de datos  
// PersonaRepository personaRepository = context.getBean(PersonaRepository.class);* personaRepository.save(persona);  
  
 *// Recuperar el objeto Persona desde la base de datos* Persona personaRecuperada = personaRepository.findById(persona.getId()).orElse(null);  
 if (personaRecuperada != null) {  
 System.*out*.println("Nombre: " + personaRecuperada.getNombre());  
 System.*out*.println("Apellido: " + personaRecuperada.getApellido());  
 System.*out*.println("Edad: " + personaRecuperada.getEdad());  
 }  
  
 };  
  
 }  
  
  
}

**Ingreso a Base de datos H2:**



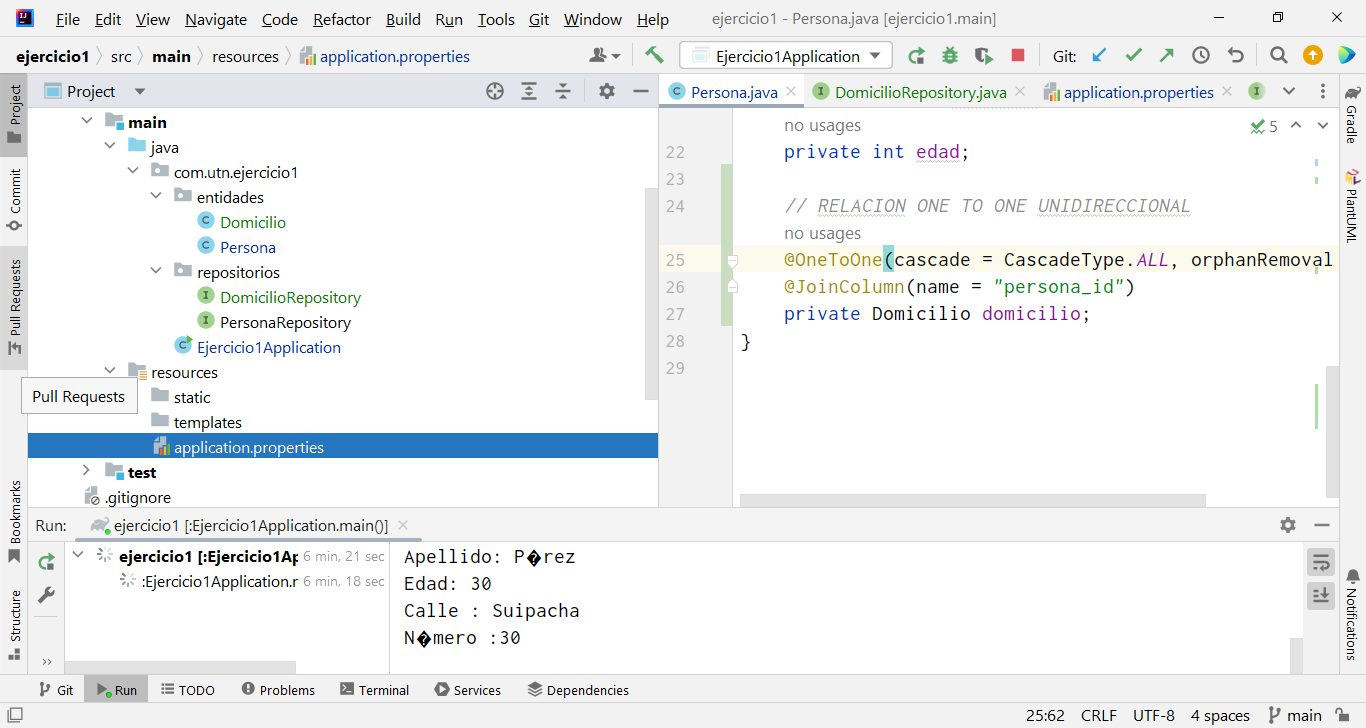
**Entidad Persona mapeada a Base de datos H2:**



**Ejercicio 2: RELACION ONE TO ONE UNIDIRECCIONAL**

Dada las siguientes entidades Persona y Domicilio, realizar la persistencia con un ORM estableciendo una relación Unidireccional ONeToOne

**Estructura del Proyecto**



**Clase Persistente Persona:**

package com.utn.ejercicio1.entidades;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import java.io.Serializable;  
@Entity  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class Persona implements Serializable {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 private String nombre;  
  
 private String apellido;  
 private int edad;  
  
 *// RELACION ONE TO ONE UNIDIRECCIONAL  
 // El Cascadeo propaga las operaciones y orphanRemoval asegura que se elimine la relacionada* @OneToOne(cascade = CascadeType.*ALL*, orphanRemoval = true)  
 @JoinColumn(name = "domicilio\_id")  
 private Domicilio domicilio;  
}

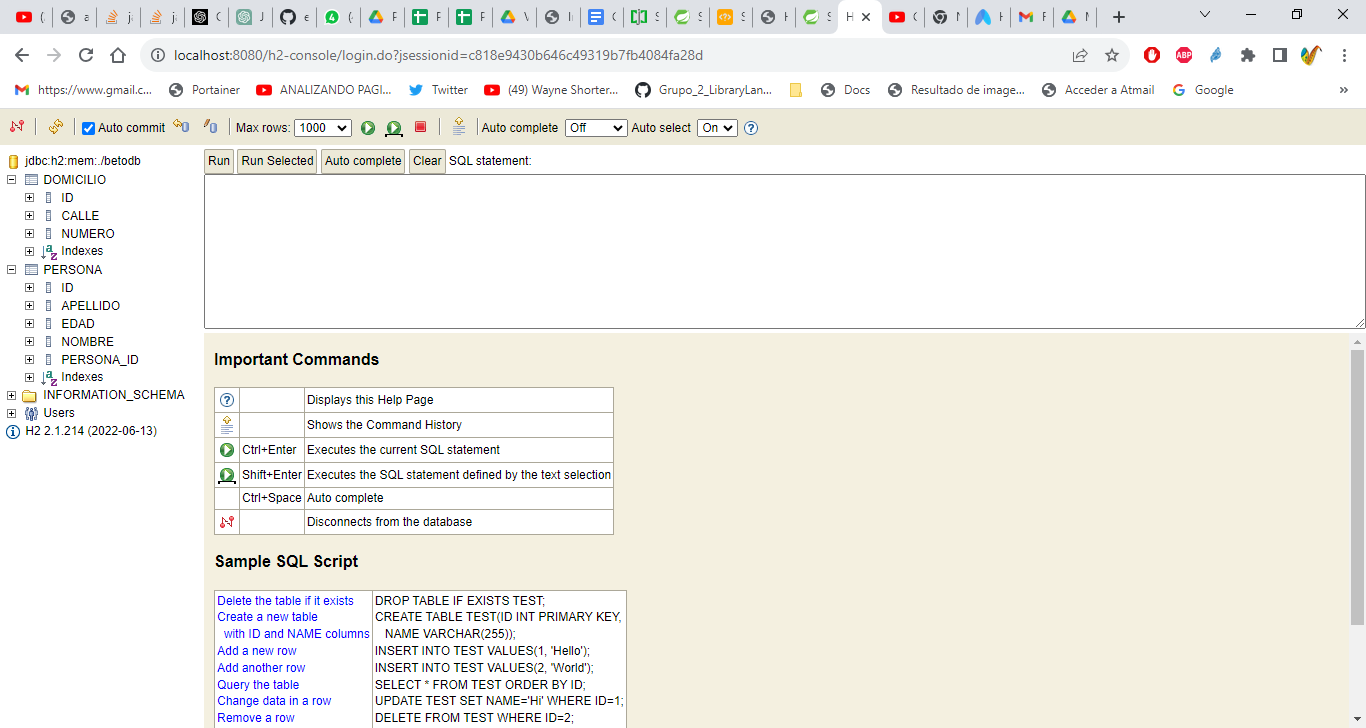
**Clase Persistente Domicilio:**

package com.utn.ejercicio1.entidades;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Id;  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import java.io.Serializable;  
@Entity  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class Domicilio implements Serializable {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 private String calle;  
 private int numero;  
  
}

**Main de ejecución :**

package com.utn.ejercicio1;  
  
import com.utn.ejercicio1.entidades.Domicilio;  
import com.utn.ejercicio1.entidades.Persona;  
import com.utn.ejercicio1.repositorios.PersonaRepository;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.CommandLineRunner;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;import org.springframework.context.ConfigurableApplicationContext;  
import org.springframework.context.annotation.Bean;  
  
  
  
@SpringBootApplication  
public class Ejercicio1Application {  
 @Autowired  
 PersonaRepository personaRepository;  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(Ejercicio1Application.class, args);  
  
  
  
 }  
  
  
  
 @Bean  
 CommandLineRunner init(PersonaRepository personaRepo) {  
 return args -> {  
 System.*out*.println("-----------------ESTOY FUNCIONANDO---------");  
  
*/\*El método builder() se genera automáticamente por Lombok  
y te permite crear una instancia de Persona.Builder.  
Luego, puedes encadenar llamadas a los métodos  
setters generados automáticamente para establecer los  
valores de los atributos de la clase.  
Finalmente, build() crea la instancia  
 de la clase Persona con los valores proporcionados.  
  
 \*/* Persona persona = Persona.*builder*()  
 .nombre("Juan")  
 .apellido("Pérez")  
 .edad(30)  
 .build();  
  
 Domicilio domicilio = Domicilio.*builder*()  
 .calle("Suipacha")  
 .numero(30)  
 .build();  
  
 *// Asocio el domicilio* persona.setDomicilio(domicilio);  
  
 *// Guardar el objeto Persona en la base de datos* personaRepository.save(persona);  
  
 *// Recuperar el objeto Persona desde la base de datos* Persona personaRecuperada = personaRepository.findById(persona.getId()).orElse(null);  
 if (personaRecuperada != null) {  
 System.*out*.println("Nombre: " + personaRecuperada.getNombre());  
 System.*out*.println("Apellido: " + personaRecuperada.getApellido());  
 System.*out*.println("Edad: " + personaRecuperada.getEdad());  
 System.*out*.println("Calle : " + personaRecuperada.getDomicilio().getCalle());  
 System.*out*.println("Número :" + personaRecuperada.getDomicilio().getNumero());  
 }  
  
 };  
  
 }  
   
}

**Entidad Persona y Domicilio mapeada unidireccionalmente a Base de datos H2**

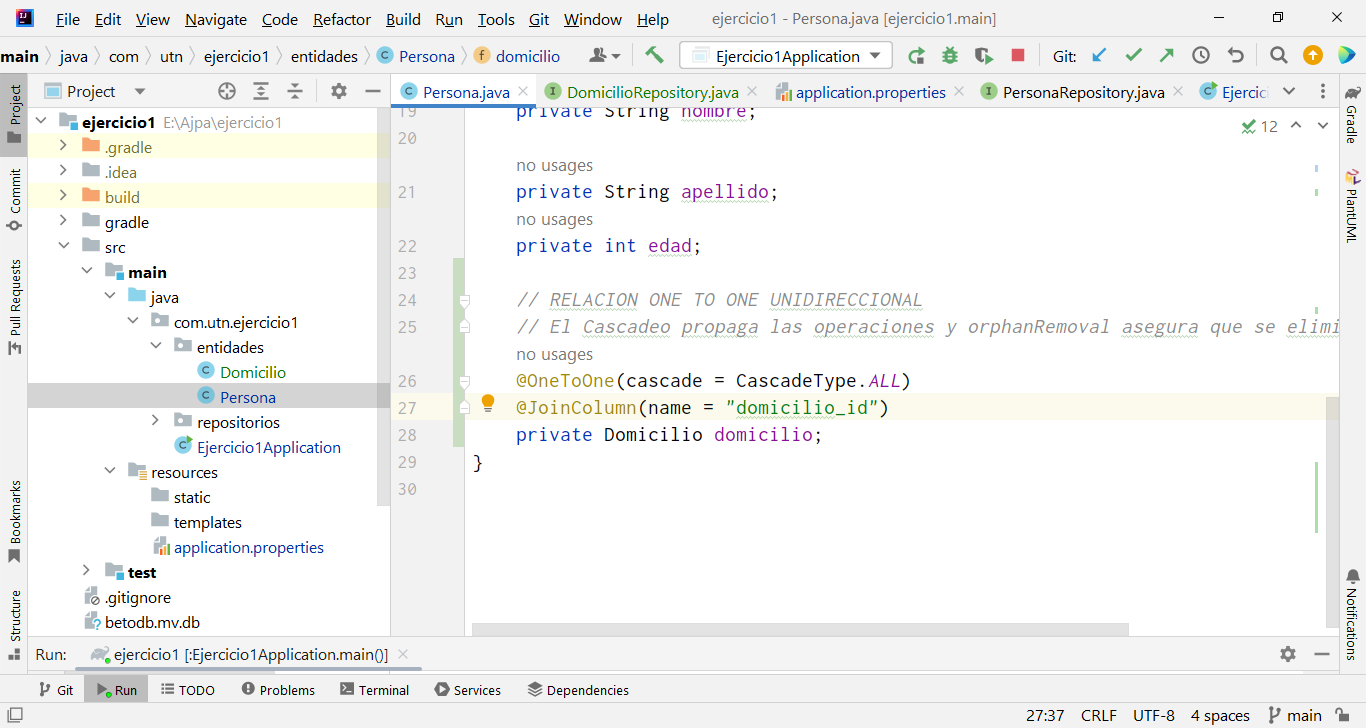


package com.utn.ejercicio1;  
  
import com.utn.ejercicio1.entidades.Domicilio;  
import com.utn.ejercicio1.entidades.Persona;  
import com.utn.ejercicio1.repositorios.PersonaRepository;  
import jakarta.annotation.PostConstruct;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.CommandLineRunner;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;import org.springframework.context.ConfigurableApplicationContext;  
import org.springframework.context.annotation.Bean;  
  
  
  
@SpringBootApplication  
public class Ejercicio1Application {  
 @Autowired  
 PersonaRepository personaRepository;  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(Ejercicio1Application.class, args);  
  
  
  
 }  
  
  
  
 @Bean  
 CommandLineRunner init(PersonaRepository personaRepo) {  
 return args -> {  
 System.*out*.println("-----------------ESTOY FUNCIONANDO---------");  
  
*/\*El método builder() se genera automáticamente por Lombok  
y te permite crear una instancia de Persona.Builder.  
Luego, puedes encadenar llamadas a los métodos  
setters generados automáticamente para establecer los  
valores de los atributos de la clase.  
Finalmente, build() crea la instancia  
 de la clase Persona con los valores proporcionados.  
  
 \*/* Persona persona = Persona.*builder*()  
 .nombre("Juan")  
 .apellido("Pérez")  
 .edad(30)  
 .build();  
  
 Domicilio domicilio = Domicilio.*builder*()  
 .calle("Suipacha")  
 .numero(30)  
 .build();  
  
 *// Asocio el domicilio* persona.setDomicilio(domicilio);  
  
 *// Guardar el objeto Persona en la base de datos* personaRepository.save(persona);  
  
 *// Recuperar el objeto Persona desde la base de datos* Persona personaRecuperada = personaRepository.findById(persona.getId()).orElse(null);  
 if (personaRecuperada != null) {  
 System.*out*.println("Nombre: " + personaRecuperada.getNombre());  
 System.*out*.println("Apellido: " + personaRecuperada.getApellido());  
 System.*out*.println("Edad: " + personaRecuperada.getEdad());  
 System.*out*.println("Calle : " + personaRecuperada.getDomicilio().getCalle());  
 System.*out*.println("Número :" + personaRecuperada.getDomicilio().getNumero());  
 }  
  
 };  
  
 }  
   
}

**Ejercicio 3: RELACION ONE TO ONE BIDIRECCIONAL**

Dada las siguientes entidades Persona y Domicilio, realizar la persistencia con un ORM estableciendo una relación Bidireccional ONeToOne

**Estructura del Proyecto**



**Clase Persistente Persona:**

package com.utn.ejercicio1.entidades;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import java.io.Serializable;  
@Entity  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class Persona implements Serializable {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 private String nombre;  
  
 private String apellido;  
 private int edad;  
  
 *// RELACION ONE TO ONE BIDIRECCIONAL  
 // El Cascadeo propaga las operaciones y orphanRemoval asegura que se elimine la relacionada***@OneToOne(cascade = CascadeType.*ALL*)  
 @JoinColumn(name = "domicilio\_id")  
 private Domicilio domicilio;**  
}

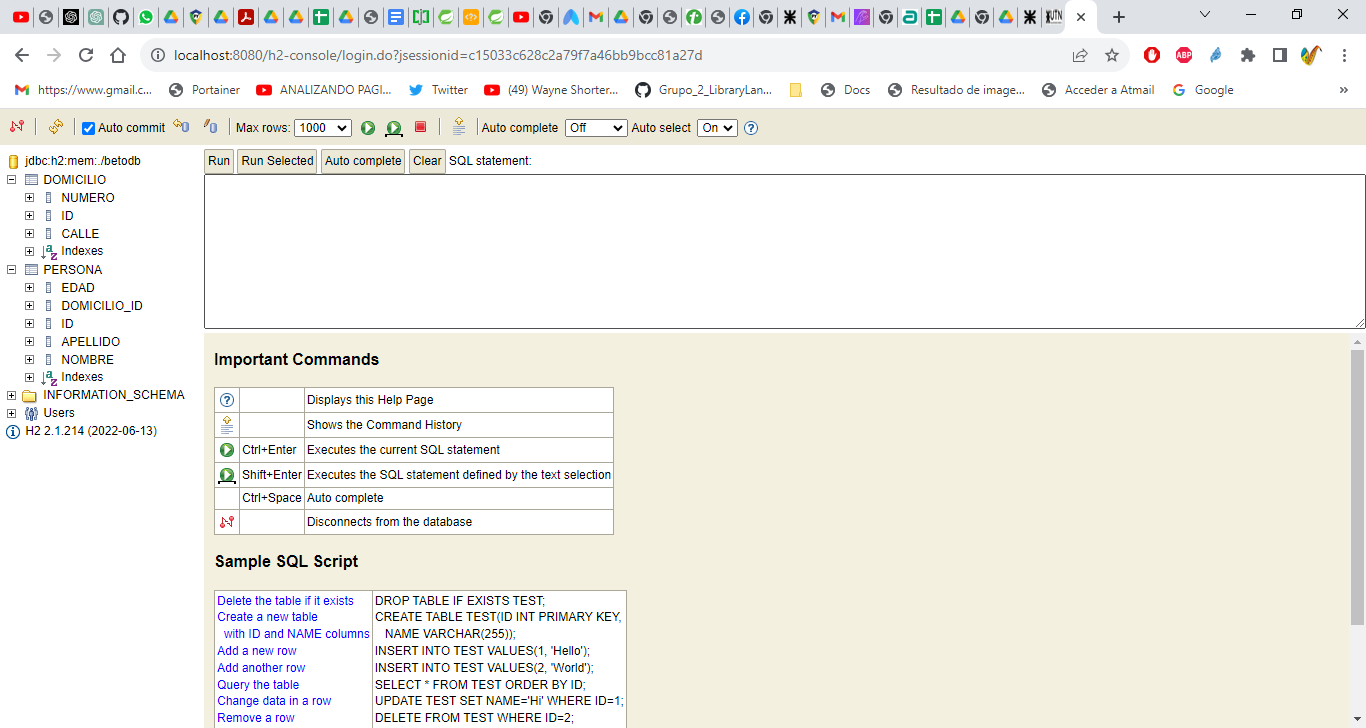
**Clase Persistente Domicilio:**

package com.utn.ejercicio1.entidades;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import java.io.Serializable;  
@Entity  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class Domicilio implements Serializable {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 private String calle;  
 private int numero;  
  
 **@OneToOne(mappedBy = "domicilio")  
  
 private Persona persona;**  
  
}

**Main de ejecución :**

package com.utn.ejercicio1;  
  
import com.utn.ejercicio1.entidades.Domicilio;  
import com.utn.ejercicio1.entidades.Persona;  
import com.utn.ejercicio1.repositorios.DomicilioRepository;  
import com.utn.ejercicio1.repositorios.PersonaRepository;  
import jakarta.annotation.PostConstruct;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.CommandLineRunner;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;import org.springframework.context.ConfigurableApplicationContext;  
import org.springframework.context.annotation.Bean;  
  
  
  
@SpringBootApplication  
public class Ejercicio1Application {  
 @Autowired  
 PersonaRepository personaRepository;  
 @Autowired  
 DomicilioRepository domicilioRepository;  
 public static void main(String[] args) {  
 SpringApplication.*run*(Ejercicio1Application.class, args);  
  
  
  
 }  
  
  
  
 @Bean  
 CommandLineRunner init(PersonaRepository personaRepo,DomicilioRepository domicilioRepo) {  
 return args -> {  
 System.*out*.println("-----------------ESTOY FUNCIONANDO---------");  
  
*/\*El método builder() se genera automáticamente por Lombok  
y te permite crear una instancia de Persona.Builder.  
Luego, puedes encadenar llamadas a los métodos  
setters generados automáticamente para establecer los  
valores de los atributos de la clase.  
Finalmente, build() crea la instancia  
 de la clase Persona con los valores proporcionados.  
  
 \*/* Domicilio domicilio = Domicilio.*builder*()  
 .calle("Suipacha")  
 .numero(30)  
 .build();  
  
 Persona persona = Persona.*builder*()  
 .nombre("Juan")  
 .apellido("Pérez")  
 .edad(30)  
 .build();  
  
 *// Asocio el domicilio* persona.setDomicilio(domicilio);  
  
 *// Asocio la persona* domicilio.setPersona(persona);  
  
 *// Guardar el objeto Persona en la base de datos* personaRepository.save(persona);  
  
  
  
  
 *// Recuperar el objeto Persona desde la base de datos* Persona personaRecuperada = personaRepository.findById(persona.getId()).orElse(null);  
 if (personaRecuperada != null) {  
 System.*out*.println("Nombre: " + personaRecuperada.getNombre());  
 System.*out*.println("Apellido: " + personaRecuperada.getApellido());  
 System.*out*.println("Edad: " + personaRecuperada.getEdad());  
 System.*out*.println("Calle : " + personaRecuperada.getDomicilio().getCalle());  
 System.*out*.println("Número :" + personaRecuperada.getDomicilio().getNumero());  
 }  
  
 System.*out*.println("...... Muestro la bidireccionalidad.......");  
 *// Recuperar el objeto Persona desde la base de datos* Domicilio domicilioRecuperado = domicilioRepository.findById(persona.getId()).orElse(null);  
 if (domicilioRecuperado!= null) {  
 System.*out*.println("Nombre: " + domicilioRecuperado.getPersona().getNombre());  
 System.*out*.println("Apellido: " + domicilioRecuperado.getPersona().getApellido());  
 System.*out*.println("Edad: " + domicilioRecuperado.getPersona().getEdad());  
 System.*out*.println("Calle : " + domicilioRecuperado.getCalle());  
 System.*out*.println("Número :" + domicilioRecuperado.getNumero());  
 }  
  
  
  
 };  
  
 }  
  
}

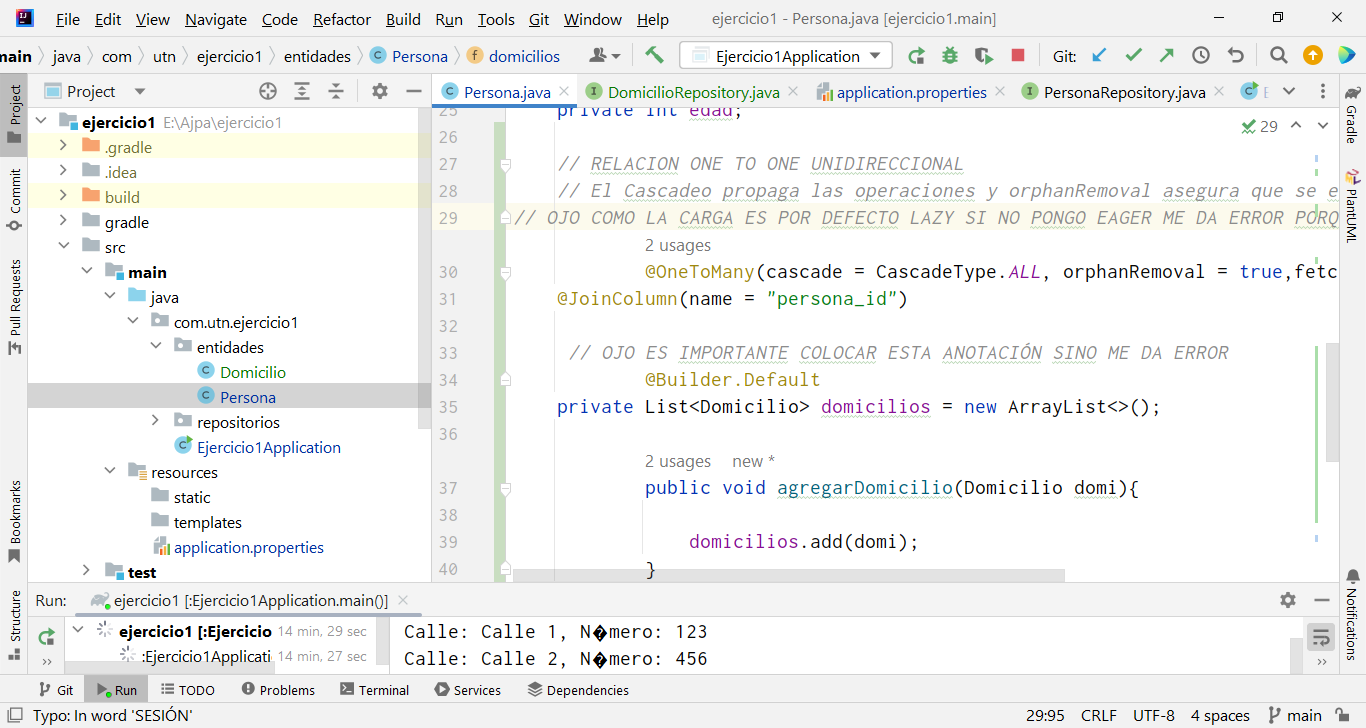
**Entidad Persona y Domicilio mapeada Bidireccionalmente a Base de datos H2**



**Ejercicio 3: RELACION ONE TO MANY UNIDIRECCIONAL**

Dada las siguientes entidades Persona y Domicilio, realizar la persistencia con un ORM estableciendo una relación Unidireccional ONeToMany

**Estructura del Proyecto**



**Clase Persistente Persona:**

package com.utn.ejercicio1.entidades;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import java.io.Serializable;  
import java.util.ArrayList;  
import java.util.List;  
  
@Entity  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class Persona implements Serializable {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 private String nombre;  
  
 private String apellido;  
 private int edad;  
  
 ***// RELACION ONE TO ONE UNIDIRECCIONAL  
 // El Cascadeo propaga las operaciones y orphanRemoval asegura que se elimine la relacionada  
// OJO COMO LA CARGA ES POR DEFECTO LAZY SI NO PONGO EAGER ME DA ERROR PORQUE CIERRA LA SESIÓN*@OneToMany(cascade = CascadeType.*ALL*, orphanRemoval = true,fetch = FetchType.*EAGER*)**

**@JoinColumn(name = "persona\_id")  
  
 *// OJO ES IMPORTANTE COLOCAR ESTA ANOTACIÓN SINO ME DA ERROR***

**@Builder.Default  
 private List<Domicilio> domicilios = new ArrayList<>();**  
 public void agregarDomicilio(Domicilio domi){  
  
 domicilios.add(domi);  
 }  
  
  
 public void mostrarDomicilios() {  
 System.*out*.println("Domicilios de " + nombre + " " + apellido + ":");  
 for (Domicilio domicilio : domicilios) {  
 System.*out*.println("Calle: " + domicilio.getCalle() + ", Número: " + domicilio.getNumero());  
 }  
  
  
}

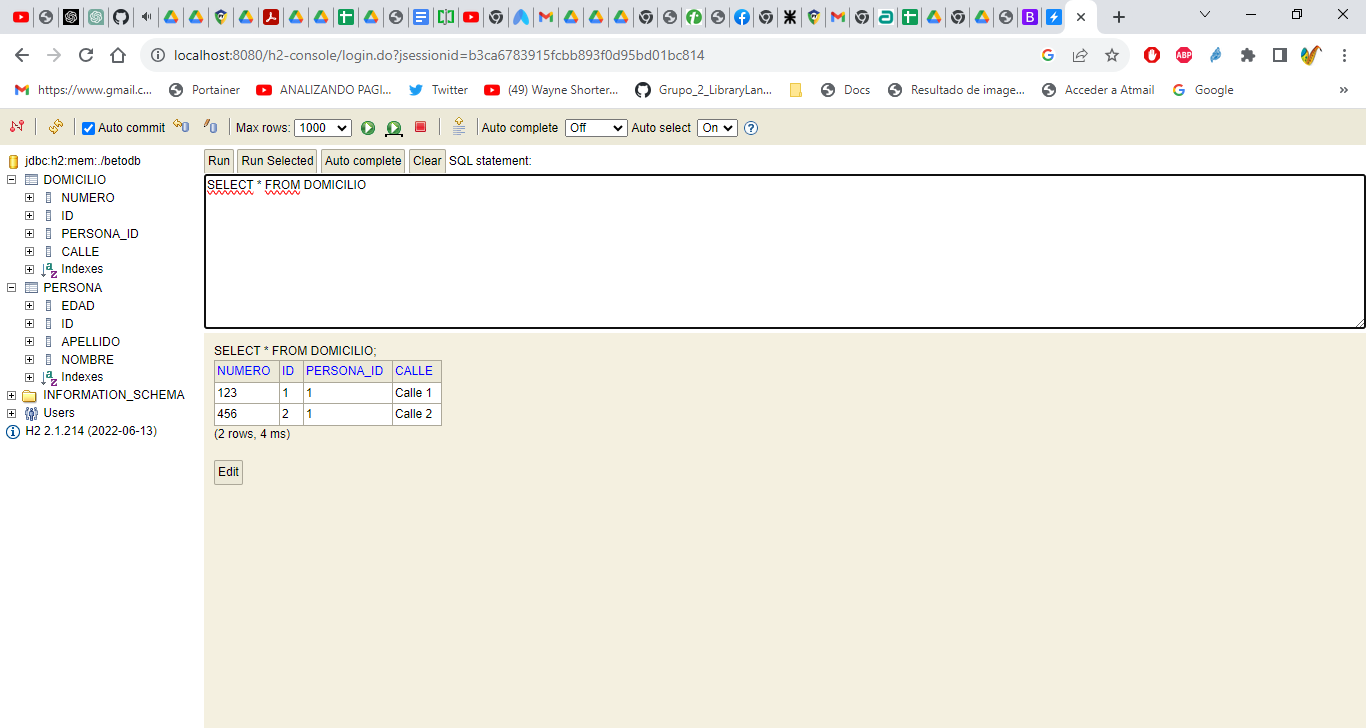
**Clase Persistente Domicilio:**

package com.utn.ejercicio1.entidades;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import java.io.Serializable;  
@Entity  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class Domicilio implements Serializable {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 private String calle;  
 private int numero;  
  
  
}

**Clase Main:**

package com.utn.ejercicio1;  
  
import com.utn.ejercicio1.entidades.Domicilio;  
import com.utn.ejercicio1.entidades.Persona;  
import com.utn.ejercicio1.repositorios.DomicilioRepository;  
import com.utn.ejercicio1.repositorios.PersonaRepository;  
  
import org.hibernate.Hibernate;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.CommandLineRunner;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;import org.springframework.context.ConfigurableApplicationContext;  
import org.springframework.context.annotation.Bean;  
  
  
  
@SpringBootApplication  
public class Ejercicio1Application {  
 @Autowired  
 PersonaRepository personaRepository;  
 @Autowired  
 DomicilioRepository domicilioRepository;  
 public static void main(String[] args) {  
 SpringApplication.*run*(Ejercicio1Application.class, args);  
  
  
  
 }  
  
  
  
 @Bean  
 CommandLineRunner init(PersonaRepository personaRepo,DomicilioRepository domicilioRepo) {  
 return args -> {  
 System.*out*.println("-----------------ESTOY FUNCIONANDO---------");  
  
*/\*El método builder() se genera automáticamente por Lombok  
y te permite crear una instancia de Persona.Builder.  
Luego, puedes encadenar llamadas a los métodos  
setters generados automáticamente para establecer los  
valores de los atributos de la clase.  
Finalmente, build() crea la instancia  
 de la clase Persona con los valores proporcionados.  
  
 \*/* Domicilio domicilio1 = Domicilio.*builder*()  
 .calle("Calle 1")  
 .numero(123)  
 .build();  
  
 Domicilio domicilio2 = Domicilio.*builder*()  
 .calle("Calle 2")  
 .numero(456)  
 .build();  
  
 *// Crear instancia de Persona y agregar domicilios* Persona persona = Persona.*builder*()  
 .nombre("Juan")  
 .apellido("Pérez")  
 .edad(30)  
 .build();  
  
 persona.agregarDomicilio(domicilio1);  
 persona.agregarDomicilio(domicilio2);  
  
  
  
 *// Guardar el objeto Persona en la base de datos* personaRepository.save(persona);  
  
  
  
  
 *// Recuperar el objeto Persona desde la base de datos* Persona personaRecuperada = personaRepository.findById(persona.getId()).orElse(null);  
  
  
 if (personaRecuperada != null) {  
 System.*out*.println("Nombre: " + personaRecuperada.getNombre());  
 System.*out*.println("Apellido: " + personaRecuperada.getApellido());  
 System.*out*.println("Edad: " + personaRecuperada.getEdad());  
 personaRecuperada.mostrarDomicilios();  
  
  
 }  
  
  
  
  
  
 };  
  
 }  
  
}

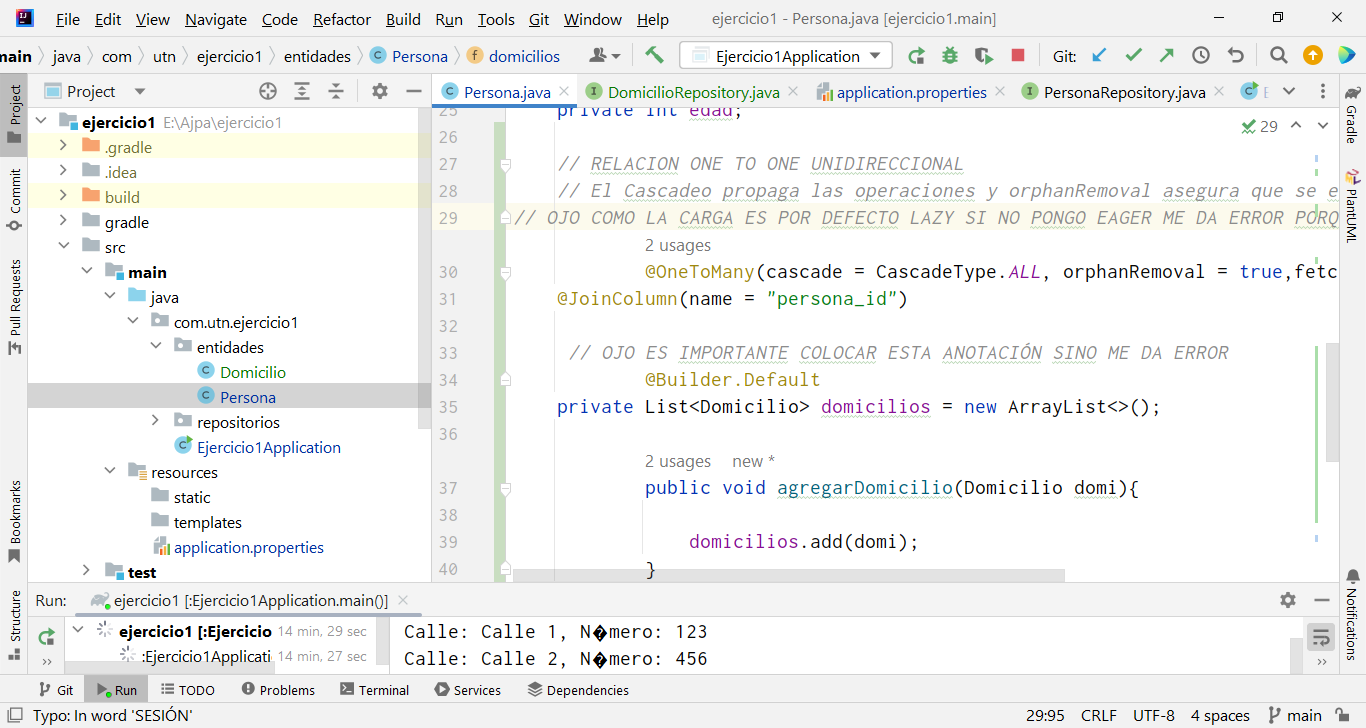
**Entidad Persona y Domicilio mapeada Unidireccionalmente a Base de datos H2**



**Ejercicio 4: RELACION ONE TO MANY BIDIRECCIONAL**

Dada las siguientes entidades Persona y Domicilio, realizar la persistencia con un ORM estableciendo una relación Bidireccional ONeToMany

**Estructura del Proyecto**



**Clase Persistente Persona:**

package com.utn.ejercicio1.entidades;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import java.io.Serializable;  
import java.util.ArrayList;  
import java.util.List;  
  
@Entity  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class Persona implements Serializable {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 private String nombre;  
  
 private String apellido;  
 private int edad;  
  
 *// RELACION ONE TO ONE UNIDIRECCIONAL  
 // El Cascadeo propaga las operaciones y orphanRemoval asegura que se elimine la relacionada  
// OJO COMO LA CARGA ES POR DEFECTO LAZY SI NO PONGO EAGER ME DA ERROR PORQUE CIERRA LA SESIÓN***@OneToMany(mappedBy = "persona", cascade = CascadeType.*ALL*, orphanRemoval = true,fetch = FetchType.*EAGER*)  
  
 *// OJO ES IMPORTANTE COLOCAR ESTA ANOTACIÓN SINO ME DA ERROR***

**@Builder.Default**

**private List<Domicilio> domicilios = new ArrayList<>();**  
 public void agregarDomicilio(Domicilio domi){  
  
 domicilios.add(domi);  
 }  
  
  
 public void mostrarDomicilios() {  
 System.*out*.println("Domicilios de " + nombre + " " + apellido + ":");  
 for (Domicilio domicilio : domicilios) {  
 System.*out*.println("Calle: " + domicilio.getCalle() + ", Número: " + domicilio.getNumero());  
 }  
  
  
}  
}

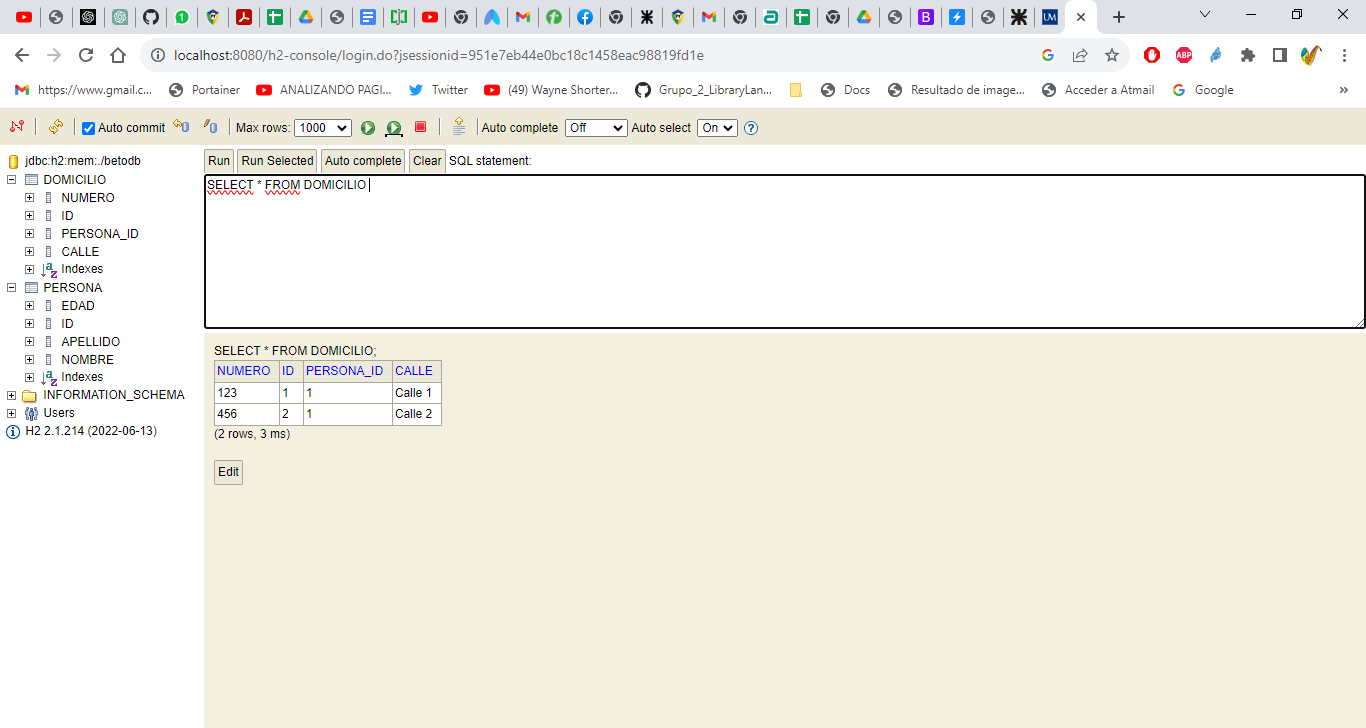
**Clase Persistente Domicilio:**

package com.utn.ejercicio1.entidades;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import java.io.Serializable;  
@Entity  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class Domicilio implements Serializable {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 private String calle;  
 private int numero;  
 @ManyToOne()  
 @JoinColumn(name = "persona\_id")  
 private Persona persona;  
  
}

**Clase Main:**

package com.utn.ejercicio1;  
  
import com.utn.ejercicio1.entidades.Domicilio;  
import com.utn.ejercicio1.entidades.Persona;  
import com.utn.ejercicio1.repositorios.DomicilioRepository;  
import com.utn.ejercicio1.repositorios.PersonaRepository;  
  
import org.hibernate.Hibernate;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.CommandLineRunner;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;import org.springframework.context.ConfigurableApplicationContext;  
import org.springframework.context.annotation.Bean;  
  
import java.util.List;  
  
  
@SpringBootApplication  
public class Ejercicio1Application {  
 @Autowired  
 PersonaRepository personaRepository;  
 @Autowired  
 DomicilioRepository domicilioRepository;  
 public static void main(String[] args) {  
 SpringApplication.*run*(Ejercicio1Application.class, args);  
  
  
  
 }  
  
  
  
 @Bean  
 CommandLineRunner init(PersonaRepository personaRepo,DomicilioRepository domicilioRepo) {  
 return args -> {  
 System.*out*.println("-----------------ESTOY FUNCIONANDO---------");  
  
*/\*El método builder() se genera automáticamente por Lombok  
y te permite crear una instancia de Persona.Builder.  
Luego, puedes encadenar llamadas a los métodos  
setters generados automáticamente para establecer los  
valores de los atributos de la clase.  
Finalmente, build() crea la instancia  
 de la clase Persona con los valores proporcionados.  
  
 \*/* Domicilio domicilio1 = Domicilio.*builder*()  
 .calle("Calle 1")  
 .numero(123)  
 .build();  
  
 Domicilio domicilio2 = Domicilio.*builder*()  
 .calle("Calle 2")  
 .numero(456)  
 .build();  
  
 *// Crear instancia de Persona y agregar domicilios* Persona persona = Persona.*builder*()  
 .nombre("Juan")  
 .apellido("Pérez")  
 .edad(30)  
 .build();  
  
 persona.agregarDomicilio(domicilio1);  
 persona.agregarDomicilio(domicilio2);  
  
*// Asignar la persona a los domicilios* domicilio1.setPersona(persona);  
 domicilio2.setPersona(persona);  
  
 *// Guardar el objeto Persona en la base de datos* personaRepository.save(persona);  
  
  
  
  
 *// Recuperar el objeto Persona desde la base de datos* Persona personaRecuperada = personaRepository.findById(persona.getId()).orElse(null);  
 if (personaRecuperada != null) {  
 System.*out*.println("Nombre: " + personaRecuperada.getNombre());  
 System.*out*.println("Apellido: " + personaRecuperada.getApellido());  
 System.*out*.println("Edad: " + personaRecuperada.getEdad());  
 System.*out*.println("----------------------------------------");  
 personaRecuperada.mostrarDomicilios();  
 System.*out*.println(" ------ finalicé desde persona");  
 }  
  
 System.*out*.println("...... Muestro la bidireccionalidad.......");  
 *// Recuperar los domicilios de la base de datos* List<Domicilio> domiciliosRecuperados = domicilioRepository.findAll();  
  
 if (domiciliosRecuperados!= null) {  
  
  
 *// Iterar a través de los domicilios y mostrar información* System.*out*.println("Información de los Domicilios:");  
 for (Domicilio domicilio : domiciliosRecuperados) {  
 System.*out*.println("Domicilio en " + domicilio.getCalle() + " " + domicilio.getNumero());  
 System.*out*.println("Perteneciente a " + domicilio.getPersona().getNombre() + " " + domicilio.getPersona().getApellido());  
 }  
  
  
 }  
  
  
  
 };  
  
 }  
  
}

**Entidad Persona y Domicilio mapeada bidireccionalmente a Base de datos H2**



**Ejercicio 5: RELACION MANY TO MANY bidi**

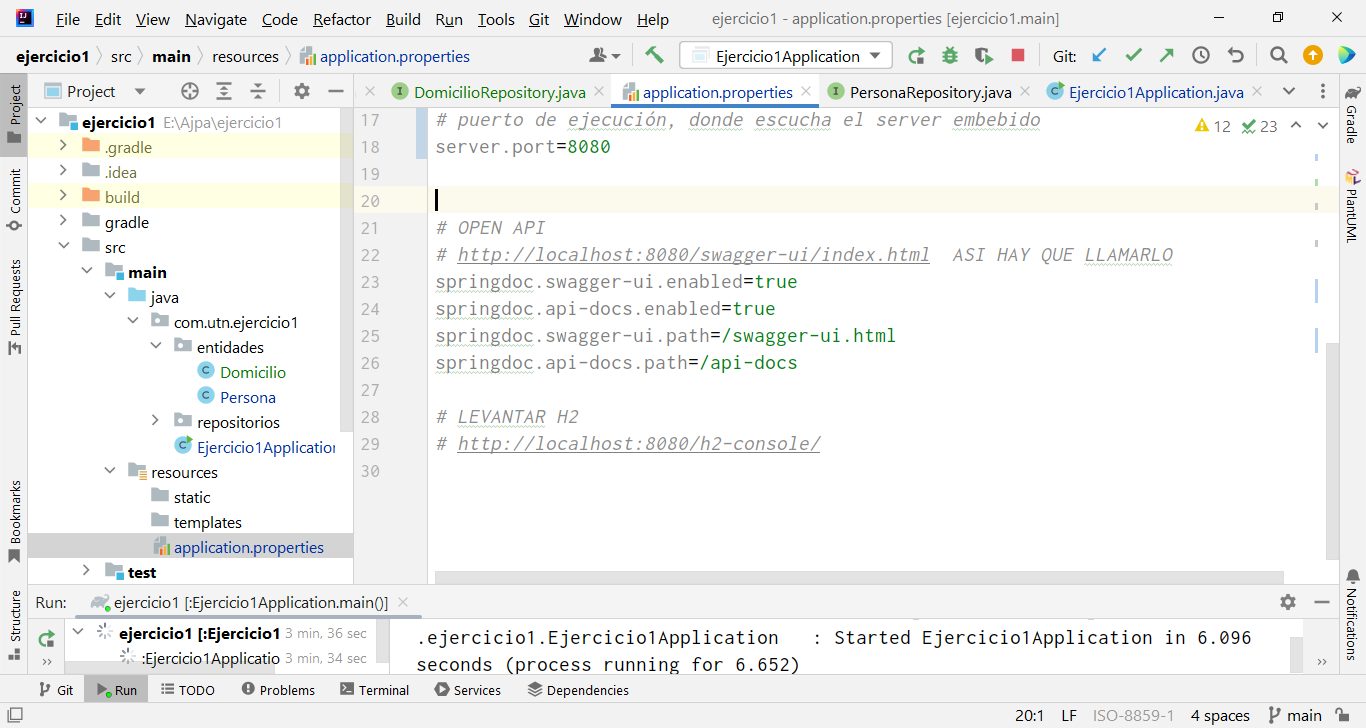
En este ejemplo, se utiliza la anotación @ManyToMany en ambas clases para indicar la relación. Se crea una tabla intermedia llamada persona\_domicilio utilizando la anotación @JoinTable en la clase Persona. Esta tabla contendrá las relaciones entre las personas y los domicilios.

La anotación @JoinTable especifica los nombres de las columnas que se utilizarán como claves foráneas en la tabla intermedia. En este caso, joinColumns se refiere a la clave foránea que hace referencia a la tabla de Persona, y inverseJoinColumns se refiere a la clave foránea que hace referencia a la tabla de Domicilio.

La anotación @ManyToMany(mappedBy = "domicilios") en la clase Domicilio indica que la relación es bidireccional y que la propiedad domicilios en la clase Persona es la propietaria de la relación.

Con estas anotaciones, JPA se encargará de administrar la tabla intermedia y las operaciones relacionadas con la relación Many-to-Many entre Persona y Domicilio.

**Estructura del Proyecto:**



**Clase Persistente Persona:**

package com.utn.ejercicio1.entidades;  
  
import jakarta.persistence.\*;  
import lombok.\*;  
  
import java.io.Serializable;  
import java.util.ArrayList;  
import java.util.HashSet;  
import java.util.List;  
import java.util.Set;  
  
@Entity  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class Persona implements Serializable {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 private String nombre;  
  
 private String apellido;  
 private int edad;  
  
 *// RELACION ONE TO ONE UNIDIRECCIONAL  
 // El Cascadeo propaga las operaciones y orphanRemoval asegura que se elimine la relacionada  
// OJO COMO LA CARGA ES POR DEFECTO LAZY SI NO PONGO EAGER ME DA ERROR PORQUE CIERRA LA SESIÓN* **@ManyToMany(cascade = {  
 CascadeType.*PERSIST*,  
 CascadeType.*MERGE* })  
 @JoinTable(  
 name = "persona\_domicilio",  
 joinColumns = @JoinColumn(name = "persona\_id"),  
 inverseJoinColumns = @JoinColumn(name = "domicilio\_id")  
 )  
  
 *// Se puede producir ERRORES DE RECURSIVIDAD POR ESO EN LOMBOCK HAY QUE  
 // eXCLUIR LOS MÉTODOS Equals y HashCode* @EqualsAndHashCode.Exclude  
 *// OJO ES IMPORTANTE COLOCAR ESTA ANOTACIÓN SINO ME DA ERROR* @Builder.Default  
 private Set<Domicilio> domicilios = new HashSet<>();**  
  
  
  
  
 public void agregarDomicilio(Domicilio domi){  
  
 domicilios.add(domi);  
 }  
  
  
 public void mostrarDomicilios() {  
 System.*out*.println("Domicilios de " + nombre + " " + apellido + ":");  
 for (Domicilio domicilio : domicilios) {  
 System.*out*.println("Calle: " + domicilio.getCalle() + ", Número: " + domicilio.getNumero());  
 }  
  
  
}  
}

**Clase Persistente Domicilio:**

package com.utn.ejercicio1.entidades;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Builder;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import java.io.Serializable;  
import java.util.HashSet;  
import java.util.Set;  
  
@Entity  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Builder  
public class Domicilio implements Serializable {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 private String calle;  
 private int numero;  
  
 **@ManyToMany(mappedBy = "domicilios")  
*// OJO NO OLVIDARSE DE ESTO PORQUE NO LE GUSTA A LOMBOCK* @Builder.Default  
 private Set<Persona> personas = new HashSet<>();**  
  
  
}

**Clase Main:**

package com.utn.ejercicio1;  
  
import com.utn.ejercicio1.entidades.Domicilio;  
import com.utn.ejercicio1.entidades.Persona;  
import com.utn.ejercicio1.repositorios.DomicilioRepository;  
import com.utn.ejercicio1.repositorios.PersonaRepository;  
  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.CommandLineRunner;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;import org.springframework.context.ConfigurableApplicationContext;  
import org.springframework.context.annotation.Bean;  
  
import java.util.List;  
  
  
@SpringBootApplication  
public class Ejercicio1Application {  
 @Autowired  
 PersonaRepository personaRepository;  
 @Autowired  
 DomicilioRepository domicilioRepository;  
 public static void main(String[] args) {  
 SpringApplication.*run*(Ejercicio1Application.class, args);  
  
  
  
 }  
  
  
  
 @Bean  
 CommandLineRunner init(PersonaRepository personaRepo,DomicilioRepository domicilioRepo) {  
 return args -> {  
 System.*out*.println("-----------------ESTOY FUNCIONANDO---------");  
  
*/\*El método builder() se genera automáticamente por Lombok  
y te permite crear una instancia de Persona.Builder.  
Luego, puedes encadenar llamadas a los métodos  
setters generados automáticamente para establecer los  
valores de los atributos de la clase.  
Finalmente, build() crea la instancia  
 de la clase Persona con los valores proporcionados.  
  
 \*/  
  
 // Crear instancias de Domicilio* Domicilio domicilio1 = Domicilio.*builder*()  
 .calle("Calle 1")  
 .numero(123)  
 .build();  
  
 Domicilio domicilio2 = Domicilio.*builder*()  
 .calle("Calle 2")  
 .numero(456)  
 .build();  
  
 *// Crear instancias de Persona y agregar domicilios* Persona persona1 = Persona.*builder*()  
 .nombre("Juan")  
 .apellido("Pérez")  
 .edad(30)  
 .build();  
  
 Persona persona2 = Persona.*builder*()  
 .nombre("María")  
 .apellido("Gómez")  
 .edad(25)  
 .build();  
  
 *// Agregar domicilios a las personas* persona1.agregarDomicilio(domicilio1);  
 persona1.agregarDomicilio(domicilio2);  
  
 persona2.agregarDomicilio(domicilio2);  
  
  
 *// Asignar las personas a los domicilios* domicilio1.getPersonas().add(persona1);  
 domicilio2.getPersonas().add(persona1);  
 domicilio2.getPersonas().add(persona2);  
  
 *// Guardar las personas y los domicilios en la base de datos* personaRepository.save(persona1);  
 *// personaRepository.save(persona2);* };  
  
 }  
  
}

**Entidad Persona y Domicilio mapeada bidireccionalmente a Base de datos H2**

