

# Spring Boot Accessing Data with JPA

**Description:** Spring Boot is an open source, micro service-based Java web framework. The Spring Boot framework creates a fully production-ready environment that is completely configurable using its prebuilt code within its own codebase.

**Project:** Build an application that uses Spring Data JPA to store and retrieve data in a relational database. Build an application that stores Customer POJOs (Plain Old Java Objects) in a memory-based database. This project is based on [Accessing Data with JPA](#) and just expands on the screenshots and step by step instructions.

**Technology:** This project uses the following technology:

Integrated Development Environment (IDE):

[Spring Tool Suite 4](#) (Version: 4.15.0.RELEASE)

Java Development Kit (JDK):

[Oracle's JDK 8](#) (1.8)

## Table of Contents

Glossary of Terminology .....	3
Generate Spring Boot Download .....	3
Import the Spring Boot Download .....	3
Import the project: “rest-service” .....	3
Project accessing-data-jpa Discussion .....	5
Main Class: AccessingDataJpaApplication .....	5
Create Domain Object Model and Repository classes .....	5
Create Domain Model Class: Customer .....	5
Update Domain Model class: Customer .....	5
Copy and paste source code for class: Customer .....	8
Create Repository Query Interface: CustomerRepository .....	9
Update repository interface: CusterRepository .....	9
Copy and paste source code for class: CustomerRepository .....	10
Update Main Class: AccessingDataJpaApplication .....	11
Copy and paste source code for class: AccessingDataJpaApplication .....	14
Build and Run the Project .....	16
Use Executable JAR File to Test Program .....	16
Use IDE to Test Program .....	17

## Glossary of Terminology

For a list of key terms and definitions used throughout this and various Spring Boot demo documents see the document titled “Appendix 01 Glossary”.

## Generate Spring Boot Download

Follow the instructions in the document title “Appendix 02 Spring Initializr” to generate a spring boot download for this project.

When the document talks about adding dependencies add only these:

**Spring Data JPA**

**H2 Database**

Dependencies

ADD DEPENDENCIES... CTRL + B

Spring Data JPA **SQL**

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

H2 Database **SQL**

Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

When the document talks about the items in the “**Project Metadata**” use the values shown below:

“**Group**” use “**com.example**”

“**Artifact**” use “**accessing-data-jpa**”

“**Name**” use “**accessing-data-jpa**”

“**Package name**” use “**com.example.accessing-data-jpa**”

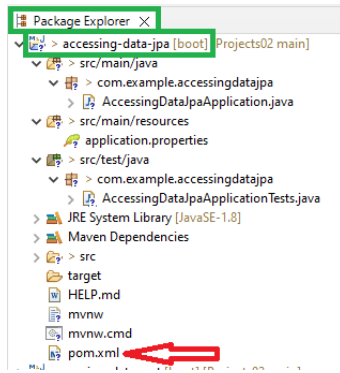
For other items in the “**Project Metadata**” use the defaults. Follow the instructions to extract the files from the zip file into the Sprint Tool Suite 4 workspace.

## Import the Spring Boot Download

Open the Spring Tool Suite 4 IDE.

Import the project: “rest-service”

Follow the instructions in the document title “Appendix 03 Import Spring Tool Suite Project” and import the “**accessing-data-jpa**” project that was created with Spring Initializr. After importing the project, it should look like the following using the IDE “Package Explorer”.



Look at the pom.xml and find the two dependencies for this project.



## Project accessing-data-jpa Discussion

This project uses the generated application to start the applications. Two other classes are created one to represent an entity a model class and a controller class.

### Main Class: AccessingDataJpaApplication

The AccessingDataJpaApplication class created when using the Spring Initializr is used with modification as the last section before testing. This class contains the “`public static void main(String[] args) {}`” method that starts the application listening when the internal web server is started.

## Create Domain Object Model and Repository classes

Create a Domain Object also known as a Model class with database annotation. The database annotation is what will create the Domain Object structure as a database table with columns. Since this project is using a H2 Database, the structure will only exist in memory with no physical representation.

Create a repository class which provide various operations involving the Domain Object.

### Create Domain Model Class: Customer

This is also referenced as a simple entity class.

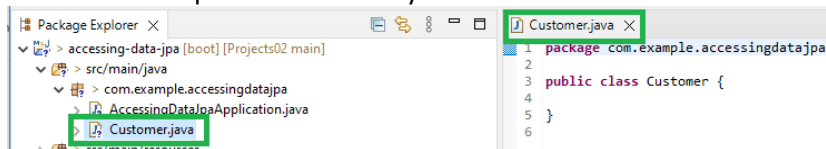
Right click on package “com.example.accessingdatajpa”

Right highlight “New” → select “Class”

Enter class “Name:” “Customer”

Click “Finish”

The new class opens automatically in an edit window.



### Update Domain Model class: Customer

If you are experienced with JPA concepts and correcting errors using the IDE you can skip to the section [“Copy and paste source code for class: Customer”](#).

Add database entity annotation. Specifies that the class is an entity. This annotation is applied to the entity class.

```
@Entity
```

```

Customer.java X
1 package com.example.accessingdatajpa;
2
3 @Entity
4 public class Customer {
5
6 }
7

```

Resolve the error.

```

3 @Entity
4 P Entity cannot be resolved to a type
5
6 } 5 quick fixes available:
7   Import 'Entity' (javax.persistence)

```

Add private class variable with database annotation.

@Id - Specifies the primary key of an entity.

@GeneratedValue - Provides for the specification of generation strategies for the values of primary keys.

@GenerationType - Defines the types of primary key generation strategies.

```

@Id
@GeneratedValue(strategy=GenerationType.AUTO)
private Long id;
private String firstName;
private String lastName;

```

```

6 public class Customer {
7   @Id
8   @GeneratedValue(strategy=GenerationType.AUTO)
9   private Long id;
10  private String firstName;
11  private String lastName;
12

```

Resolve the errors.

```

7 @Id
8 @
9 P Id cannot be resolved to a type
10 P 4 quick fixes available:
11 P Import 'Id' (javax.persistence)
12

```

```

9 @GeneratedValue(strategy=GenerationType.AUTO)
10 P GeneratedValue cannot be resolved to a type
11 P 5 quick fixes available:
12 P Import 'GeneratedValue' (javax.persistence)
13
14 }

```

```

10 @GeneratedValue(strategy=GenerationType.AUTO)
11 private Long id;
12 private String firstName;
13 private String lastName;
14
15 }

```

GenerationType cannot be resolved to a variable  
8 quick fixes available:  
Import 'GenerationType' (javax.persistence)

Add the rest of the method code.

```
protected Customer() {}

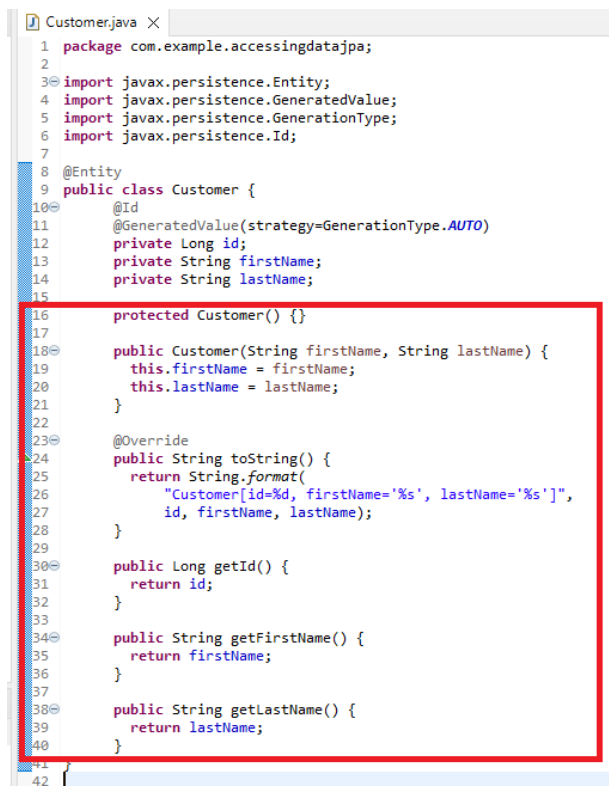
public Customer(String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
}

@Override
public String toString() {
    return String.format(
        "Customer[id=%d, firstName='%s', lastName='%s']",
        id, firstName, lastName);
}

public Long getId() {
    return id;
}

public String getFirstName() {
    return firstName;
}

public String getLastName() {
    return lastName;
}
```



```
Customer.java X
1 package com.example.accessingdatajpa;
2
3 import javax.persistence.Entity;
4 import javax.persistence.GeneratedValue;
5 import javax.persistence.GenerationType;
6 import javax.persistence.Id;
7
8 @Entity
9 public class Customer {
10     @Id
11     @GeneratedValue(strategy=GenerationType.AUTO)
12     private Long id;
13     private String firstName;
14     private String lastName;
15
16     protected Customer() {}
17
18     public Customer(String firstName, String lastName) {
19         this.firstName = firstName;
20         this.lastName = lastName;
21     }
22
23     @Override
24     public String toString() {
25         return String.format(
26             "Customer[id=%d, firstName='%s', lastName='%s']",
27             id, firstName, lastName);
28     }
29
30     public Long getId() {
31         return id;
32     }
33
34     public String getFirstName() {
35         return firstName;
36     }
37
38     public String getLastName() {
39         return lastName;
40     }
41
42 }
```

The no-arg constructor “Customer() {}” has the access modifier of “protected”. Access modifier of “protected” prevents instantiation of the class, outside the package. Basically, you can only create an object of a “protected” class within the same package in this case “com.example.accessingdatajpa”.

The @Override annotation does as its namesake meaning that it performs this method and not the toString() method from its parent the Object class. In Java all classes derive from the Object class.

“In this example, you store Customer objects, each annotated as a JPA entity.

The Customer class is annotated with `@Entity`, indicating that it is a JPA entity. (Because no `@Table` annotation exists, it is assumed that this entity is mapped to a table named Customer.)

The Customer object's `id` property is annotated with `@Id` so that JPA recognizes it as the object's ID. The `id` property is also annotated with `@GeneratedValue` to indicate that the ID should be generated automatically.

The other two properties, `firstName` and `lastName`, are left unannotated. It is assumed that they are mapped to columns that share the same names as the properties themselves.

The convenient `toString()` method print outs the customer's properties." -- [Accessing Data with JPA](#)

### Copy and paste source code for class: Customer

As an alternative you can copy the code below and replace all code in the class if you used the same package name.

```
package com.example.accessingdatajpa;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Customer {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Long id;
    private String firstName;
    private String lastName;

    protected Customer() {}

    public Customer(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Override
    public String toString() {
        return String.format(
            "Customer[id=%d, firstName='%s', lastName='%s']",
            id, firstName, lastName);
    }

    public Long getId() {
        return id;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }
}
```



## Create Repository Query Interface: CustomerRepository

### Create Simple Queries

“Spring Data JPA focuses on using JPA to store data in a relational database. Its most compelling feature is the ability to create repository implementations automatically, at runtime, from a repository interface.” -- [Accessing Data with JPA](#)

Right click on package “com.example.accessingdatajpa”

Right highlight “New” → select “Interface”

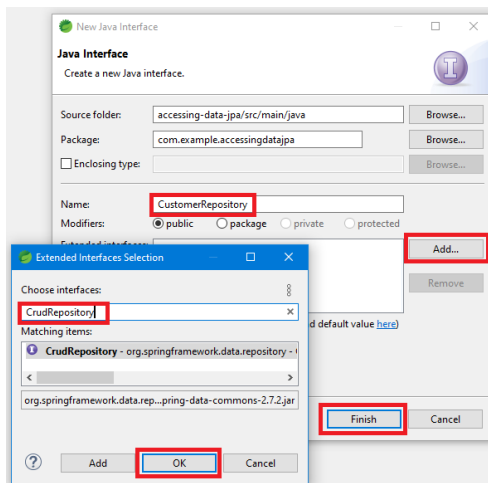
Enter interface “Name:” “CustomerRepository”

Click “Extended interfaces:” “Add...” button

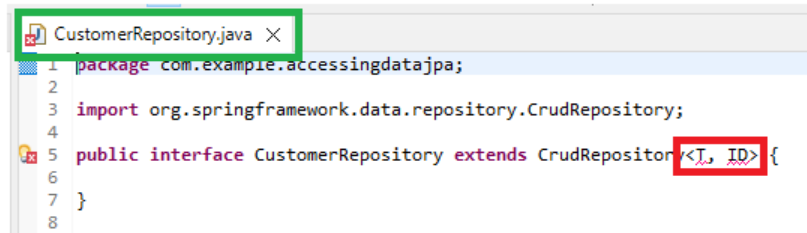
Enter “Chooses interface”: “CrudRepository”

Click “OK”

Click “Finish”



The interface opens automatically in an edit window with errors.



## Update repository interface: CusterRepository

If you are experienced with JPA concepts and correcting errors using the IDE you can skip to the section “[Copy and paste source code for class: CustomerRepository](#)”.

Fix the initial error, update the types in the extends class <diamond>. This update is based on the parameters the class intends to pass in the interface. In this case the Customer object and an identifier of object Long.

```
<Customer, Long>
```

```

CustomerRepository.java X
1 package com.example.accessingdatajpa;
2
3 import org.springframework.data.repository.CrudRepository;
4
5 public interface CustomerRepository extends CrudRepository<Customer, Long> {
6
7 }

```

Adding the inherit class “CrudRepository” using the interface Wizard automatically adds the correct import.

Add a class methods to return a List of Customer and find Customer by record id.

```

List<Customer> findByLastName(String lastName);

Customer findById(long id);

```

```

5 public interface CustomerRepository extends CrudRepository<Customer, Long> {
6
7     List<Customer> findByLastName(String lastName);
8
9     Customer findById(long id);
10 }

```

Resolve the error.

```

7     List<Customer> findByLa
8
9
10 }
11

```

List cannot be resolved to  
32 quick fixes available:  
Import 'List' (java.util)

The completed interface.

```

CustomerRepository.java X
1 package com.example.accessingdatajpa;
2
3 import java.util.List;
4
5 import org.springframework.data.repository.CrudRepository;
6
7 public interface CustomerRepository extends CrudRepository<Customer, Long> {
8
9     List<Customer> findByLastName(String lastName);
10
11     Customer findById(long id);
12 }

```

Copy and paste source code for class: CustomerRepository

As an alternative you can copy the code below and replace all code in the class if you used the same package name.

```

package com.example.accessingdatajpa;

import java.util.List;

import org.springframework.data.repository.CrudRepository;

public interface CustomerRepository extends CrudRepository<Customer, Long> {

    List<Customer> findByLastName(String lastName);

    Customer findById(long id);

}

```

## Update Main Class: AccessingDataJpaApplication

Like mentioned earlier the AccessingDataJpaApplication class created when using the Spring Initializr is used with modification. This section walks through the modification of this class.

“`@SpringBootApplication` is a convenience annotation that adds all of the following:

- `@Configuration`: Tags the class as a source of bean definitions for the application context.
- `@EnableAutoConfiguration`: Tells Spring Boot to start adding beans based on classpath settings, other beans, and various property settings. For example, if `spring-webmvc` is on the classpath, this annotation flags the application as a web application and activates key behaviors, such as setting up a `DispatcherServlet`.
- `@ComponentScan`: Tells Spring to look for other components, configurations, and services in the `com/example` package, letting it find the controllers.

The `main()` method uses Spring Boot’s `SpringApplication.run()` method to launch an application. Did you notice that there was not a single line of XML? There is no `web.xml` file, either. This web application is 100% pure Java and you did not have to deal with configuring any plumbing or infrastructure.

Now you need to modify the simple class that the Initializr created for you. To get output (to the console, in this example), you need to set up a logger.”

-- [Accessing Data with JPA](#)

If you are experienced with the concepts above and correcting errors using the IDE you can skip to the section “[Copy and paste source code for class: AccessingDataJpaApplication](#)”.

The class Wizard generated the following basic main class.

```
AccessingDataJpaApplication.java X
1 package com.example.accessingdatajpa;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5
6 @SpringBootApplication
7 public class AccessingDataJpaApplication {
8
9     public static void main(String[] args) {
10         SpringApplication.run(AccessingDataJpaApplication.class, args);
11     }
12 }
13 }
```

Add a class logging variable.

```
private static final Logger log = LoggerFactory.getLogger(ConsumingRestApplication.class);
```

```

7 public class AccessingDataInaApplication {
8     private static final Logger log = LoggerFactory.getLogger(AccessingDataJpaApplication.class);
9
10    public static void main(String[] args) {

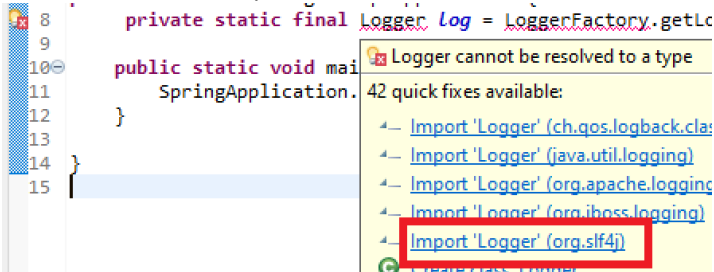
```

### Correct the import errors:

Use the IDE Wizard to help correct the errors.

Hover the mouse over the error.

Select the import link shown below.



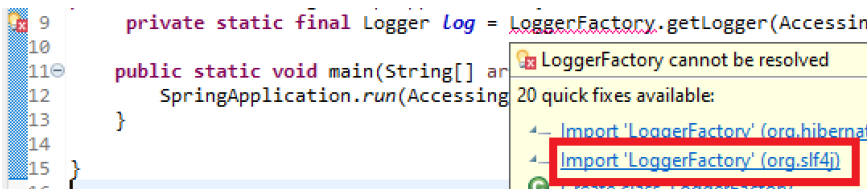
```

8     private static final Logger log = LoggerFactory.getLogger(AccessingDataJpaApplication.class);
9
10    public static void main(String[] args) {
11        SpringApplication.run(AccessingDataJpaApplication.class, args);
12    }
13
14 }
15

```

Logger cannot be resolved to a type  
42 quick fixes available:

- Import 'Logger' (ch.qos.logback.classic.Logger)
- Import 'Logger' (java.util.logging.Logger)
- Import 'Logger' (org.apache.logging.log4j.Logger)
- Import 'Logger' (org.jboss.logging.Logger)
- Import 'Logger' (org.slf4j.Logger)**



```

9     private static final Logger log = LoggerFactory.getLogger(AccessingDataJpaApplication.class);
10
11    public static void main(String[] args) {
12        SpringApplication.run(AccessingDataJpaApplication.class, args);
13    }
14
15 }
16

```

LoggerFactory cannot be resolved  
20 quick fixes available:

- Import 'LoggerFactory' (org.hibernate.boot.internal)
- Import 'LoggerFactory' (org.slf4j.LoggerFactory)**

Add a method with @Bean annotation.

The annotation is applied on a method to specify that it returns a bean. The bean is managed by Spring context. The method is responsible for creating the instance.

```

@Bean
public CommandLineRunner demo(CustomerRepository repository) {
    return (args) -> {
        // save a few customers
        repository.save(new Customer("Jack", "Bauer"));
        repository.save(new Customer("Chloe", "O'Brian"));
        repository.save(new Customer("Kim", "Bauer"));
        repository.save(new Customer("David", "Palmer"));
        repository.save(new Customer("Michelle", "Dessler"));

        // fetch all customers
        log.info("Customers found with findAll():");
        log.info("-----");
        for (Customer customer : repository.findAll()) {
            log.info(customer.toString());
        }
        log.info("");

        // fetch an individual customer by ID
        Customer customer = repository.findById(1L);
        log.info("Customer found with findById(1L):");
        log.info("-----");
        log.info(customer.toString());
        log.info("");

        // fetch customers by last name
        log.info("Customer found with findByLastName('Bauer'):");
        log.info("-----");
        repository.findByLastName("Bauer").forEach(bauer -> {
            log.info(bauer.toString());
        });
        // for (Customer bauer : repository.findByLastName("Bauer")) {
        //     log.info(bauer.toString());
        // }
        log.info("");
    };
}

```

```

16 @Bean
17 public CommandLineRunner demo(CustomerRepository repository) {
18     return (args) -> {
19         // save a few customers
20         repository.save(new Customer("Jack", "Bauer"));
21         repository.save(new Customer("Chloe", "O'Brian"));
22         repository.save(new Customer("Kim", "Bauer"));
23         repository.save(new Customer("David", "Palmer"));
24         repository.save(new Customer("Michelle", "Dessler"));
25
26         // fetch all customers
27         log.info("Customers found with findAll():");
28         log.info("-----");
29         for (Customer customer : repository.findAll()) {
30             log.info(customer.toString());
31         }
32         log.info("");
33
34         // fetch an individual customer by ID
35         Customer customer = repository.findById(1L);
36         log.info("Customer found with findById(1L):");
37         log.info("-----");
38         log.info(customer.toString());
39         log.info("");
40
41         // fetch customers by last name
42         log.info("Customer found with findByName('Bauer')");
43         log.info("-----");
44         repository.findByName("Bauer").forEach(bauer -> {
45             log.info(bauer.toString());
46         });
47         // for (Customer bauer : repository.findByName("Bauer")) {
48         //     log.info(bauer.toString());
49         // }
50         log.info("");
51     };
52 }

```

Use the IDE Wizard to help correct the errors.  
 Hover the mouse over the error.  
 Select the import link shown below.

The first screenshot shows an error on line 16: '@Bean'. The error message is 'Bean cannot be resolved to a type'. Below the error, it says '4 quick fixes available:'. The first quick fix is 'Import 'Bean' (org.springframework.context.annotation)', which is highlighted with a red box.

The second screenshot shows an error on line 18: 'public CommandLineRunner demo(...)'. The error message is 'CommandLineRunner cannot be resolved to a type'. Below the error, it says '7 quick fixes available:'. The first quick fix is 'Import 'CommandLineRunner' (org.springframework.boot)', which is highlighted with a red box.

The completed class code.

```
AccessingDataJpaApplication.java X
4
3 import org.slf4j.Logger;
4 import org.slf4j.LoggerFactory;
5 import org.springframework.boot.CommandLineRunner;
6 import org.springframework.boot.SpringApplication;
7 import org.springframework.boot.autoconfigure.SpringBootApplication;
8 import org.springframework.context.annotation.Bean;
9
10 @SpringBootApplication
11 public class AccessingDataJpaApplication {
12     private static final Logger log = LoggerFactory.getLogger(AccessingDataJpaApplication.class);
13
14     public static void main(String[] args) {
15         SpringApplication.run(AccessingDataJpaApplication.class, args);
16     }
17
18     @Bean
19     public CommandLineRunner demo(CustomerRepository repository) {
20         return (args) -> {
21             // save a few customers
22             repository.save(new Customer("Jack", "Bauer"));
23             repository.save(new Customer("Chloe", "O'Brian"));
24             repository.save(new Customer("Kim", "Bauer"));
25             repository.save(new Customer("David", "Palmer"));
26             repository.save(new Customer("Michelle", "Dessler"));
27
28             // fetch all customers
29             log.info("Customers found with findAll():");
30             log.info("-----");
31             for (Customer customer : repository.findAll()) {
32                 log.info(customer.toString());
33             }
34             log.info("");
35
36             // fetch an individual customer by ID
37             Customer customer = repository.findById(1L);
38             log.info("Customer found with findById(1L):");
39             log.info("-----");
40             log.info(customer.toString());
41             log.info("");
42
43             // fetch customers by last name
44             log.info("Customer found with findByLastName('Bauer'):");
45             log.info("-----");
46             repository.findByLastName("Bauer").forEach(bauer -> {
47                 log.info(bauer.toString());
48             });
49             // for (Customer bauer : repository.findByLastName("Bauer")) {
50             //     log.info(bauer.toString());
51             // }
52             log.info("");
53         };
54     }
55 }
```

Copy and paste source code for class: AccessingDataJpaApplication

As an alternative you can copy the code below and replace all code in the class if you used the same package name.

```
package com.example.accessingdatajpa;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
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 AccessingDataJpaApplication {

    private static final Logger log = LoggerFactory.getLogger(AccessingDataJpaApplication.class);

    public static void main(String[] args) {
        SpringApplication.run(AccessingDataJpaApplication.class, args);
    }

    @Bean
    public CommandLineRunner demo(CustomerRepository repository) {
        return (args) -> {
            // save a few customers
            repository.save(new Customer("Jack", "Bauer"));
            repository.save(new Customer("Chloe", "O'Brian"));
            repository.save(new Customer("Kim", "Bauer"));
            repository.save(new Customer("David", "Palmer"));
            repository.save(new Customer("Michelle", "Dessler"));

            // fetch all customers
            log.info("Customers found with findAll():");
            log.info("-----");
            for (Customer customer : repository.findAll()) {
                log.info(customer.toString());
            }
        }
    }
}
```

```
log.info("");

// fetch an individual customer by ID
Customer customer = repository.findById(1L);
log.info("Customer found with findById(1L):");
log.info("-----");
log.info(customer.toString());
log.info("");

// fetch customers by last name
log.info("Customer found with findByLastName('Bauer'):");
log.info("-----");
repository.findByLastName("Bauer").forEach(bauer -> {
    log.info(bauer.toString());
});
// for (Customer bauer : repository.findByLastName("Bauer")) {
//     log.info(bauer.toString());
// }
log.info("");
};
}
```

## Build and Run the Project

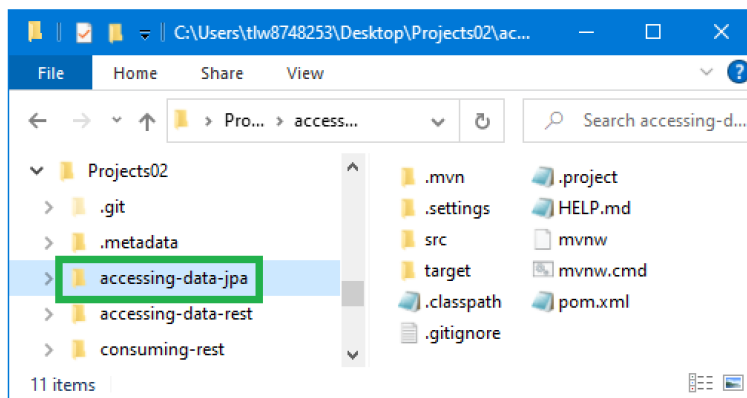
The website “[Accessing Data with JPA](#)” this project is based on has testing uses an executable Java Archive (JAR) file as the way to run and test the program.

Alternatively testing can be performed using the IDE running the main class “AccessingDataJpaApplication” as “Sprint Boot App” inside the IDE.

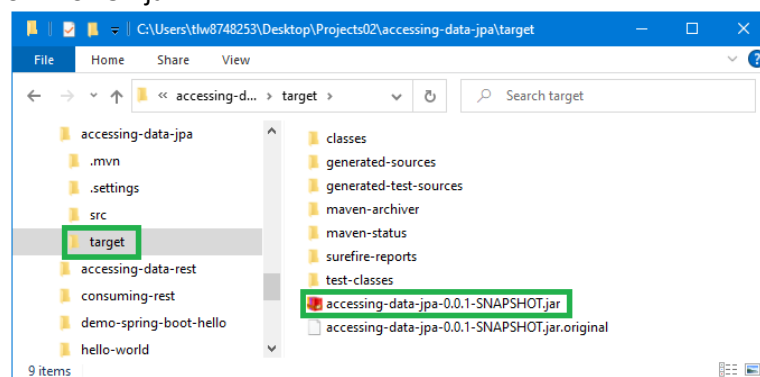
The subsection here discuss testing the program using an executable JAR and using the IDE. Since the program is using a memory database the data will not be preserved. Each run of the application will create the data and display the same results.

### Use Executable JAR File to Test Program

In the “Appendix 04 Run Spring Initializr Project” document follow the instructions in section title “Executable JAR File Lifecycle”. Follow the instructions in sub sections title “Build the Executable JAR” and “Run the Executable JAR File” and stop there. Makes sure to build in this project’s folder.



After the build the executable JAR file is found in the target folder “accessing-data-jpa-0.0.1-SNAPSHOT.jar”.





After starting the project's executable JAR file test the application. The program's main class `AccessingDataJpaApplication` contains code to create data and display the data using methods defined in the repository class. The results from running the executable JAR file are shown below.

```
MINGW64/c:/Users/tlw8748253/Desktop/Projects02/accessing-data-jpa
C:\Users\tlw8748253\Desktop\Projects02>java -jar target/accessing-data-jpa-0.0.1-SNAPSHOT.jar

:: Spring Boot ::
(v2.7.3)

2022-08-25 12:14:45.554 INFO 9368 --- [main] c.e.a.AccessingDataJpaApplication : Starting AccessingDataJpaApplication v0.0.1-SNAPSHOT using Java 1.8.0_291 on TLW8748253
2022-08-25 12:14:45.558 INFO 9368 --- [main] c.e.a.AccessingDataJpaApplication : No active profile set, falling back to 1 default profile: "default"
2022-08-25 12:14:46.204 INFO 9368 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2022-08-25 12:14:46.267 INFO 9368 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 49 ms. Found 1 JPA repository.
2022-08-25 12:14:47.253 INFO 9368 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2022-08-25 12:14:47.552 INFO 9368 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2022-08-25 12:14:47.644 INFO 9368 --- [main] org.hibernate.Version : HHN000204: Processing PersistenceUnitInfo [name: default]
2022-08-25 12:14:47.749 INFO 9368 --- [main] org.hibernate.Version : HHN000412: Hibernate ORM core version 5.6.10.Final
2022-08-25 12:14:48.035 INFO 9368 --- [main] org.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2022-08-25 12:14:48.216 INFO 9368 --- [main] org.hibernate.dialect.Dialect : HHN000400: using dialect: org.hibernate.dialect.H2Dialect
2022-08-25 12:14:49.067 INFO 9368 --- [main] org.h.e.t.j.p.i.JtaPlatformInitiator : HHN000490: using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2022-08-25 12:14:49.075 INFO 9368 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2022-08-25 12:14:49.700 INFO 9368 --- [main] c.e.a.AccessingDataJpaApplication : Started AccessingDataJpaApplication in 4.821 seconds (JVM running for 10.01s)
2022-08-25 12:14:49.843 INFO 9368 --- [main] c.e.a.AccessingDataJpaApplication :
-----
Customers found with findAll():
-----
Customer[id=1, firstName='Jack', lastName='Bauer']
Customer[id=2, firstName='Chloe', lastName='O'Brian']
Customer[id=3, firstName='Kim', lastName='Bauer']
Customer[id=4, firstName='David', lastName='Palmer']
Customer[id=5, firstName='Michelle', lastName='Dessler']
-----
2022-08-25 12:14:50.007 INFO 9368 --- [main] c.e.a.AccessingDataJpaApplication :
Customer found with findById(1L):
-----
Customer[id=1, firstName='Jack', lastName='Bauer']
-----
2022-08-25 12:14:50.007 INFO 9368 --- [main] c.e.a.AccessingDataJpaApplication :
Customer found with findByLastName('Bauer'):
-----
Customer[id=1, firstName='Jack', lastName='Bauer']
Customer[id=3, firstName='Kim', lastName='Bauer']
-----
2022-08-25 12:14:50.023 INFO 9368 --- [main] c.e.a.AccessingDataJpaApplication :
-----
2022-08-25 12:14:50.023 INFO 9368 --- [main] c.e.a.AccessingDataJpaApplication :
-----
2022-08-25 12:14:50.023 INFO 9368 --- [main] c.e.a.AccessingDataJpaApplication :
-----
2022-08-25 12:14:50.106 INFO 9368 --- [main] c.e.a.AccessingDataJpaApplication :
-----
2022-08-25 12:14:50.106 INFO 9368 --- [main] c.e.a.AccessingDataJpaApplication :
-----
2022-08-25 12:14:50.106 INFO 9368 --- [main] c.e.a.AccessingDataJpaApplication :
-----
2022-08-25 12:14:50.112 INFO 9368 --- [main] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
2022-08-25 12:14:50.112 INFO 9368 --- [main] SchemaDropperImpl$DelayedDropActionImpl : HHN000477: Starting delayed evictData of schema as part of SessionFactory shutdown
2022-08-25 12:14:50.122 INFO 9368 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2022-08-25 12:14:50.127 INFO 9368 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.
```

## Use IDE to Test Program

Right click inside the main class “`AccessingDataRestApplication`” select “Run As” → “Sprint Boot App”. Expand the “Console” by double clicking the tab. The results are the same as running as an executable JAR file.

```
Outline Problems Javadoc Declaration Console X
<terminated> accessing-data-jpa - AccessingDataJpaApplication (Spring Boot App) C:\Program Files\Java\jdk-1.8.0_291\bin\java.exe (Aug 25, 2022, 11:43:26 AM - 11:43:33 AM) [pid: 1424]

:: Spring Boot ::
(v2.7.3)

2022-08-25 11:43:28.375 INFO 1424 --- [main] c.e.a.AccessingDataJpaApplication : Starting AccessingDataJpaApplication using Java 1.8.0_291 on TLW8748253
2022-08-25 11:43:28.379 INFO 1424 --- [main] c.e.a.AccessingDataJpaApplication : No active profile set, falling back to 1 default profile: "default"
2022-08-25 11:43:29.186 INFO 1424 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2022-08-25 11:43:29.188 INFO 1424 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 57 ms. Found 1 JPA repository.
2022-08-25 11:43:29.677 INFO 1424 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2022-08-25 11:43:29.679 INFO 1424 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2022-08-25 11:43:30.186 INFO 1424 --- [main] org.hibernate.Version : HHN000204: Processing PersistenceUnitInfo [name: default]
2022-08-25 11:43:30.194 INFO 1424 --- [main] org.hibernate.Version : HHN000412: Hibernate ORM core version 5.6.10.Final
2022-08-25 11:43:30.441 INFO 1424 --- [main] org.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2022-08-25 11:43:30.642 INFO 1424 --- [main] org.hibernate.dialect.Dialect : HHN000400: using dialect: org.hibernate.dialect.H2Dialect
2022-08-25 11:43:31.489 INFO 1424 --- [main] org.h.e.t.j.p.i.JtaPlatformInitiator : HHN000490: using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
2022-08-25 11:43:31.582 INFO 1424 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2022-08-25 11:43:32.036 INFO 1424 --- [main] c.e.a.AccessingDataJpaApplication : Started AccessingDataJpaApplication in 4.815 seconds (JVM running for 10.01s)
2022-08-25 11:43:32.145 INFO 1424 --- [main] c.e.a.AccessingDataJpaApplication :
-----
Customers found with findAll():
-----
Customer[id=1, firstName='Jack', lastName='Bauer']
Customer[id=2, firstName='Chloe', lastName='O'Brian']
Customer[id=3, firstName='Kim', lastName='Bauer']
Customer[id=4, firstName='David', lastName='Palmer']
Customer[id=5, firstName='Michelle', lastName='Dessler']
-----
2022-08-25 11:43:32.310 INFO 1424 --- [main] c.e.a.AccessingDataJpaApplication :
Customer found with findById(1L):
-----
Customer[id=1, firstName='Jack', lastName='Bauer']
-----
2022-08-25 11:43:32.322 INFO 1424 --- [main] c.e.a.AccessingDataJpaApplication :
Customer found with findByLastName('Bauer'):
-----
Customer[id=1, firstName='Jack', lastName='Bauer']
Customer[id=3, firstName='Kim', lastName='Bauer']
-----
2022-08-25 11:43:32.322 INFO 1424 --- [main] c.e.a.AccessingDataJpaApplication :
-----
2022-08-25 11:43:32.322 INFO 1424 --- [main] c.e.a.AccessingDataJpaApplication :
-----
2022-08-25 11:43:32.322 INFO 1424 --- [main] c.e.a.AccessingDataJpaApplication :
-----
2022-08-25 11:43:32.371 INFO 1424 --- [main] c.e.a.AccessingDataJpaApplication :
-----
2022-08-25 11:43:32.371 INFO 1424 --- [main] c.e.a.AccessingDataJpaApplication :
-----
2022-08-25 11:43:32.371 INFO 1424 --- [main] c.e.a.AccessingDataJpaApplication :
-----
2022-08-25 11:43:32.458 INFO 1424 --- [main] j.LocalContainerEntityManagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
2022-08-25 11:43:32.458 INFO 1424 --- [main] SchemaDropperImpl$DelayedDropActionImpl : HHN000477: Starting delayed evictData of schema as part of SessionFactory shutdown
2022-08-25 11:43:32.458 INFO 1424 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown initiated...
2022-08-25 11:43:32.472 INFO 1424 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Shutdown completed.
```