

# TUTORIAL JDBC PARTE 1

<b>Passo 0 - ACESSANDO A BASE DE DADOS.....</b>	<b>3</b>
<b>Passo 1 - CRIAR A CLASSE DE REPRESENTAÇÃO DOS DADOS NA APLICAÇÃO JAVA 8</b>	
<b>Passo 2 - CRIAR A CONEXÃO COM O BANCO DE DADOS.....</b>	<b>11</b>
<b>Passo 3 - CRIAR A ABSTRAÇÃO DO REPOSITÓRIO DE DADOS.....</b>	<b>15</b>
<b>Passo 04 - CRIAR A CLASSE DE TESTE.....</b>	<b>20</b>

# Tutorial JDBC

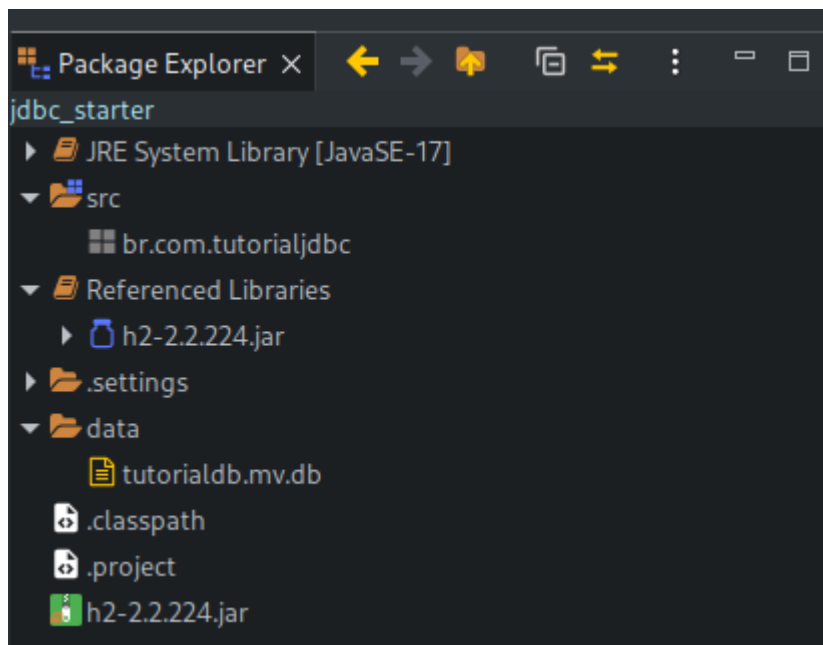
## Objetivo

Realizar a consulta das informações de um banco de dados através de uma aplicação Java.

## Avisos:

Para realizar este tutorial você precisará baixar o projeto “jdbc\_starter” que está disponível no ambiente AVA. O projeto possui as dependências necessárias para executar este tutorial. Descompacte o projeto e realize a importação dele na IDE Eclipse.

Ao abrir o projeto no eclipse, a configuração do projeto deve ser parecida com a imagem abaixo:



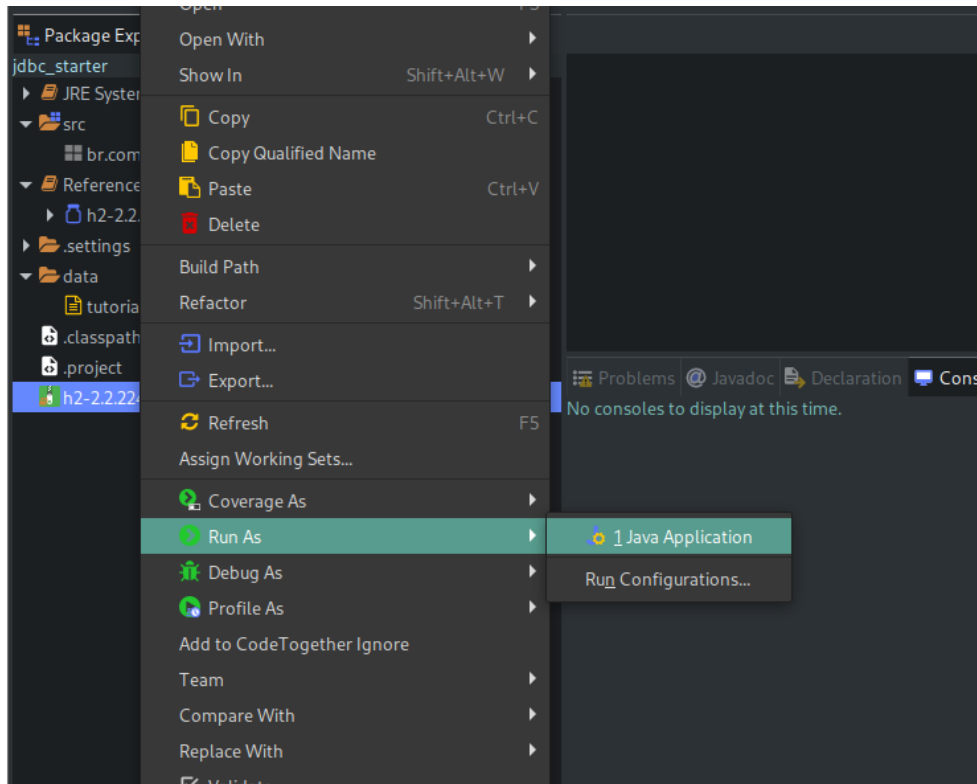
## Passo 0 - ACESSANDO A BASE DE DADOS

Este projeto está configurado para utilizar o banco de dados H2.

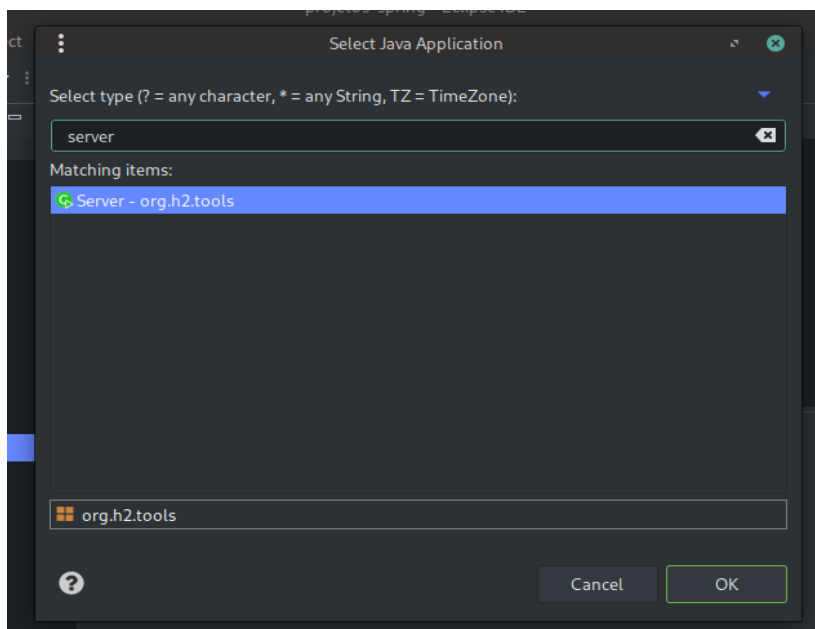
O H2 é um banco de dados que pode ser embarcado em aplicações e pode ser utilizado com dados em memória ou através de uma base de dados de arquivo que é o que utilizaremos neste tutorial. Ele possui uma interface web que nos permite executar comandos sql e verificar o resultado.

Para acessar a interface web do H2 realize os seguintes passos:

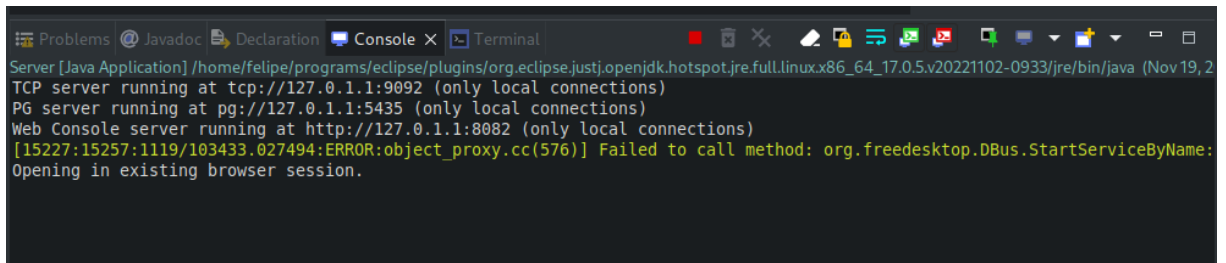
- 1) clique com o botão direito do mouse sobre o arquivo h2-2.224.jar e selecione a opção Run As -> Java Application



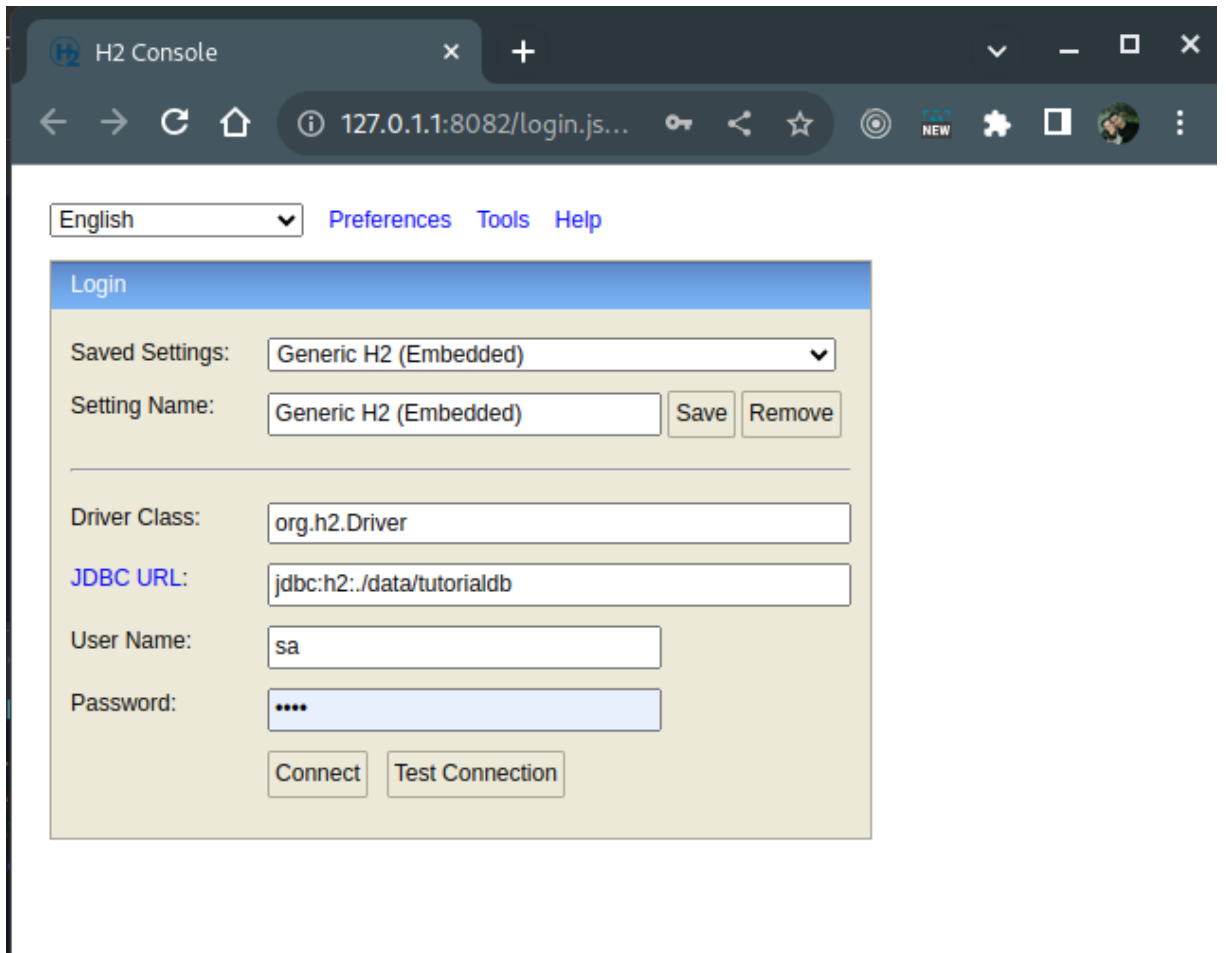
- 2) Em seguida na próxima tela que se abrirá digite no campo de filtro na parte superior da tela a palavra server, selecione a classe Server de acordo com a imagem abaixo e clique no botão ok.



- 3) Você deve ver algumas mensagens no console e em seguida o browser deve abrir na página inicial da interface web do H2.

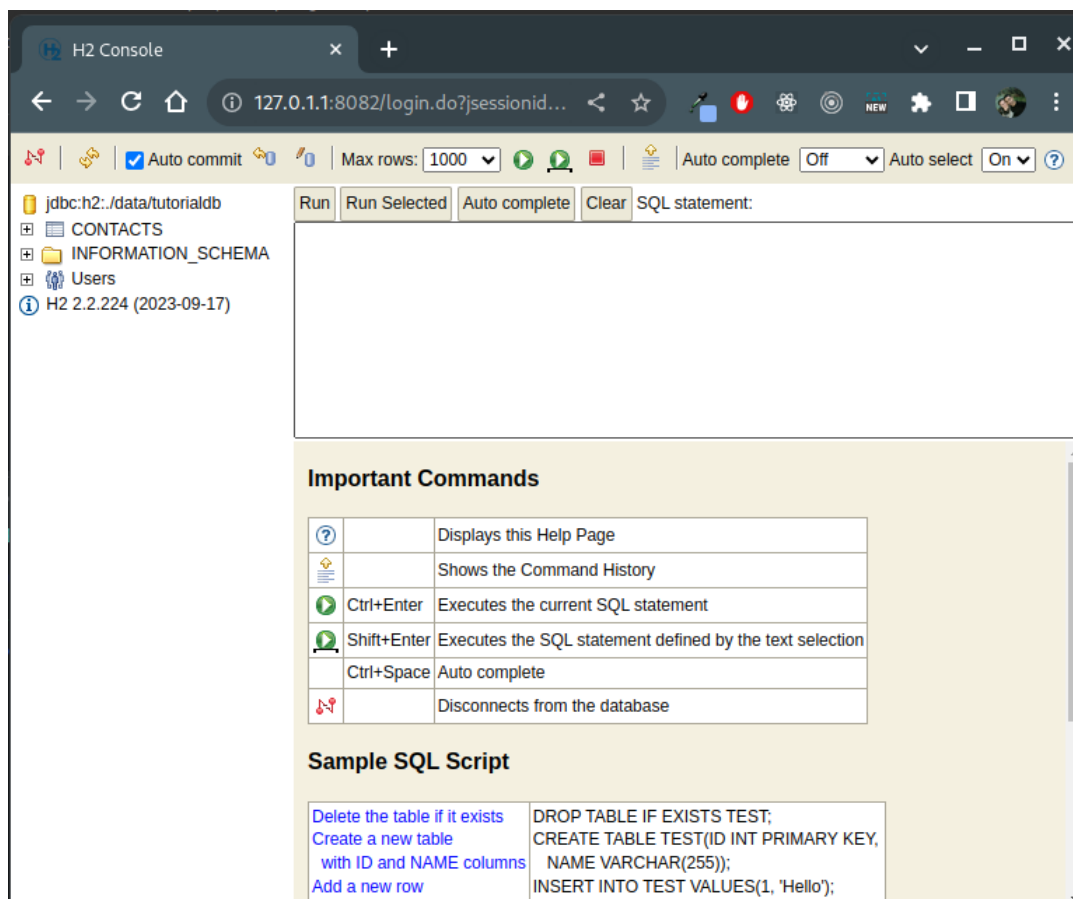


```
Server [Java Application] /home/felipe/programs/eclipse/plugins/org.eclipse.justi.openjdk.hotspot.jre.full.linux.x86_64_17.0.5.v20221102-0933/jre/bin/java (Nov 19, 2022)
TCP server running at tcp://127.0.1.1:9092 (only local connections)
PG server running at pg://127.0.1.1:5435 (only local connections)
Web Console server running at http://127.0.1.1:8082 (only local connections)
[15227:15257:1119/103433.027494:ERROR:object_proxy.cc(576)] Failed to call method: org.freedesktop.DBus.StartServiceByName:
Opening in existing browser session.
```

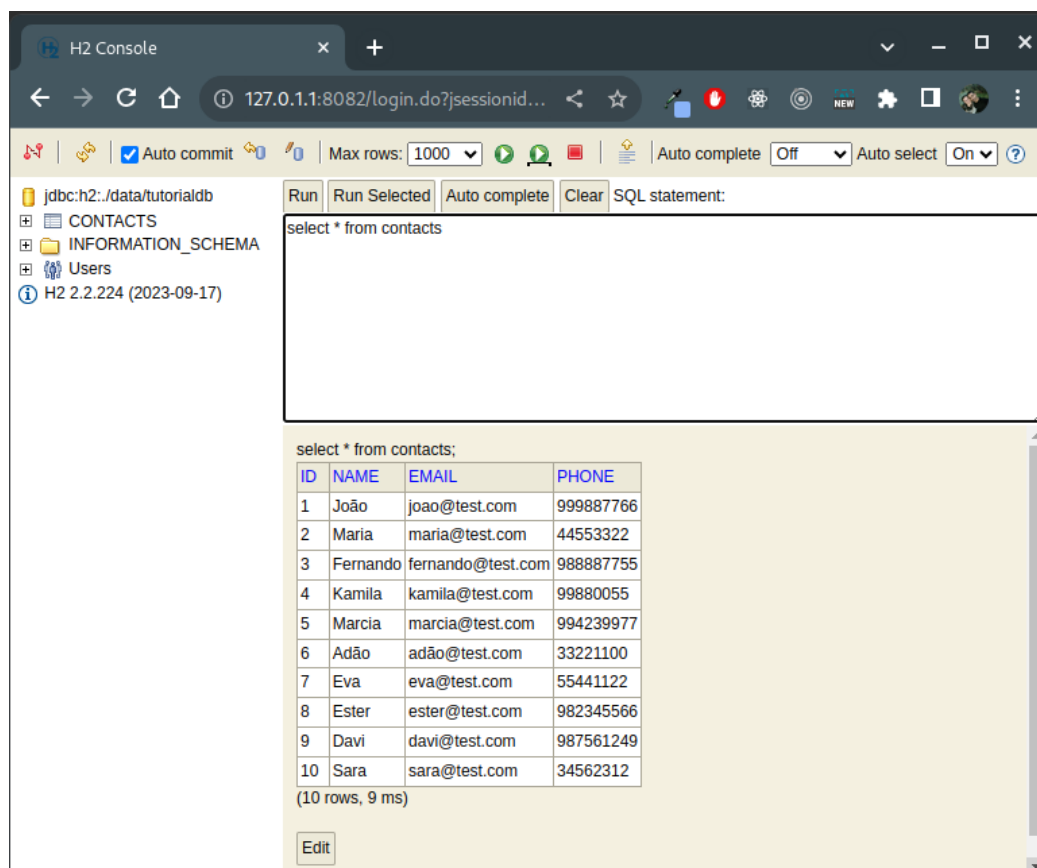


- 4) Digite os seguintes dados para realizar a conexão com o banco H2:
- No campo JDBC URL digite o valor jdbc:h2:./data/tutorialdb;
  - No campo User Name digite sa;
  - E no campo Password digite java;

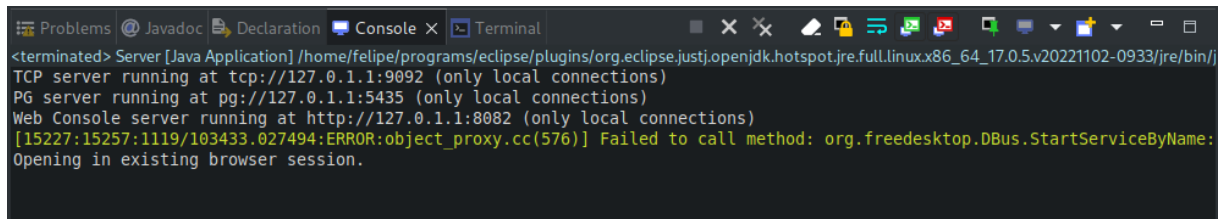
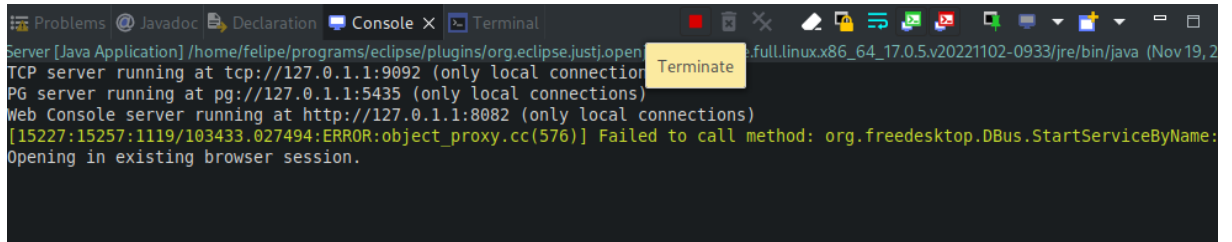
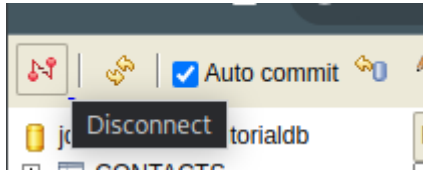
Em seguida clique no botão Connect e se tudo estiver correto você verá a tela de execução de comando da interface web do H2.



- 5) Na caixa de texto superior da página digite o comando sql para verificar os contatos cadastrados na tabela contacts (***select \* from contacts***) e clique no botão “Run” logo acima da caixa de texto; você deverá ver o resultado da operação na parte inferior da tela.



- 6) Após verificar que os dados estão cadastrados no banco clique no botão superior no canto esquerdo para desconectar da base de dados e feche a aplicação do servidor que está rodando no terminal do Eclipse.



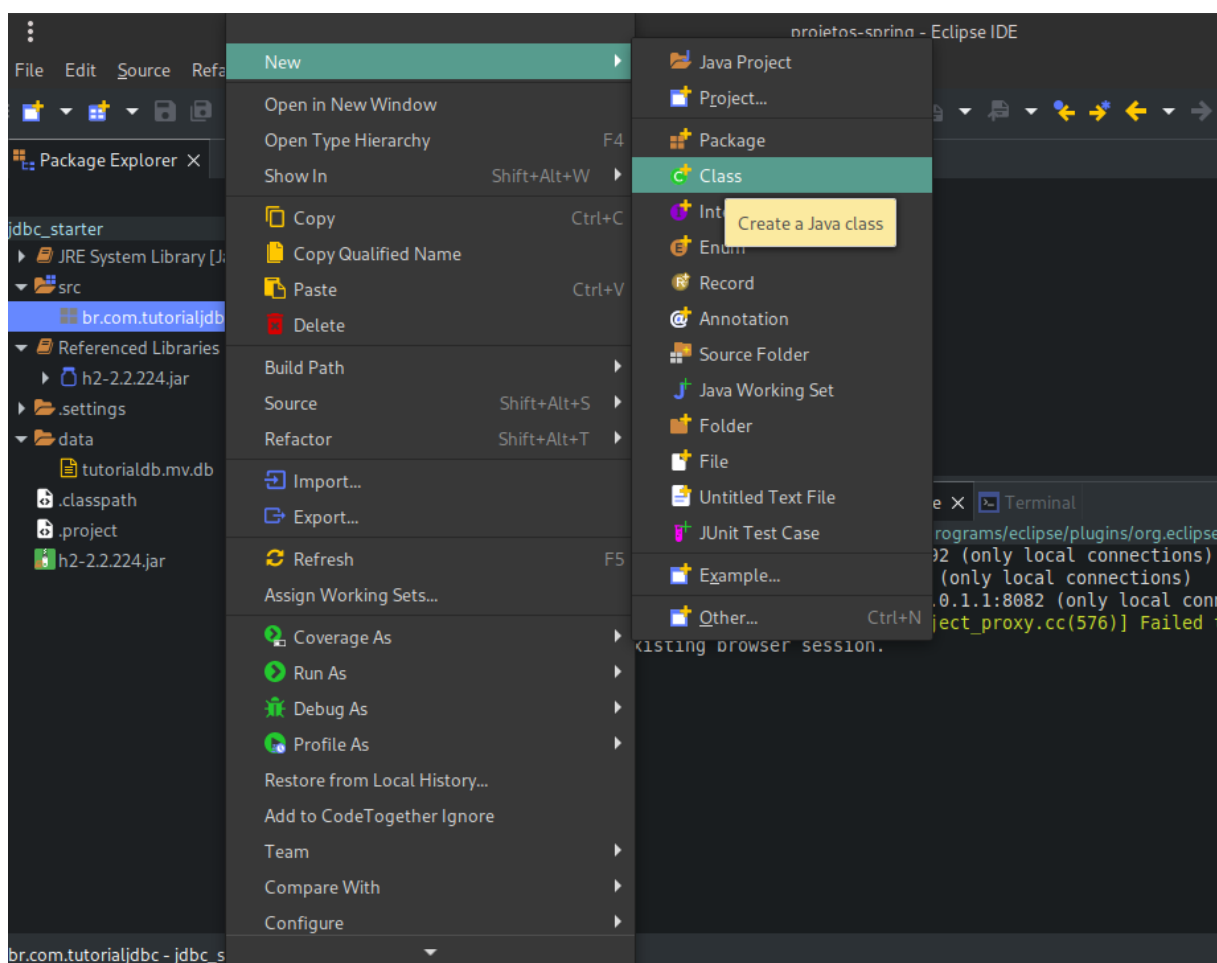


## Passo 1 - CRIAR A CLASSE DE REPRESENTAÇÃO DOS DADOS NA APLICAÇÃO JAVA

Agora que você já teve a oportunidade de consultar os dados no H2 chegou a hora de iniciarmos o nosso processo de leitura dos dados diretamente na aplicação Java.

A primeira coisa que precisamos fazer é criarmos uma classe que represente as informações que serão lidas do banco de dados, por isso iremos criar a classe “Contact” com os atributos id, name, email e phone, que são exatamente os mesmos dados da tabela “contacts” do banco de dados.

Para realizarmos esse mapeamento, clique com o botão direito sobre o nome do pacote “**br.com.tutorialjdb**” e selecione a opção **new -> class**.



Na próxima tela do wizard coloque o nome da classe de **Contact** e clique no botão **finish** na parte inferior direita da tela.

**Java Class**  
Create a new Java class.

Source folder: jdbc\_starter/src Browse...

Package: br.com.tutorialjdbc Browse...

Enclosing type: Browse...

Name: Contact

Modifiers: ☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static  
☒ none ☐ sealed ☐ non-sealed ☐ final

Superclass: java.lang.Object Browse...

Interfaces: Add...  
Remove

Which method stubs would you like to create?  
☐ public static void main(String[] args)  
☐ Constructors from superclass  
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))  
☐ Generate comments

? Cancel Finish

Em seguida preencha os dados da classe de acordo com as informações abaixo:

```
package br.com.tutorialjdbc;
import java.util.Objects;

public class Contact {
    private Integer id;
    private String name;
    private String email;
    private String phone;

    public Contact() {}

    public Contact(Integer id, String name, String email, String phone) {
```

```

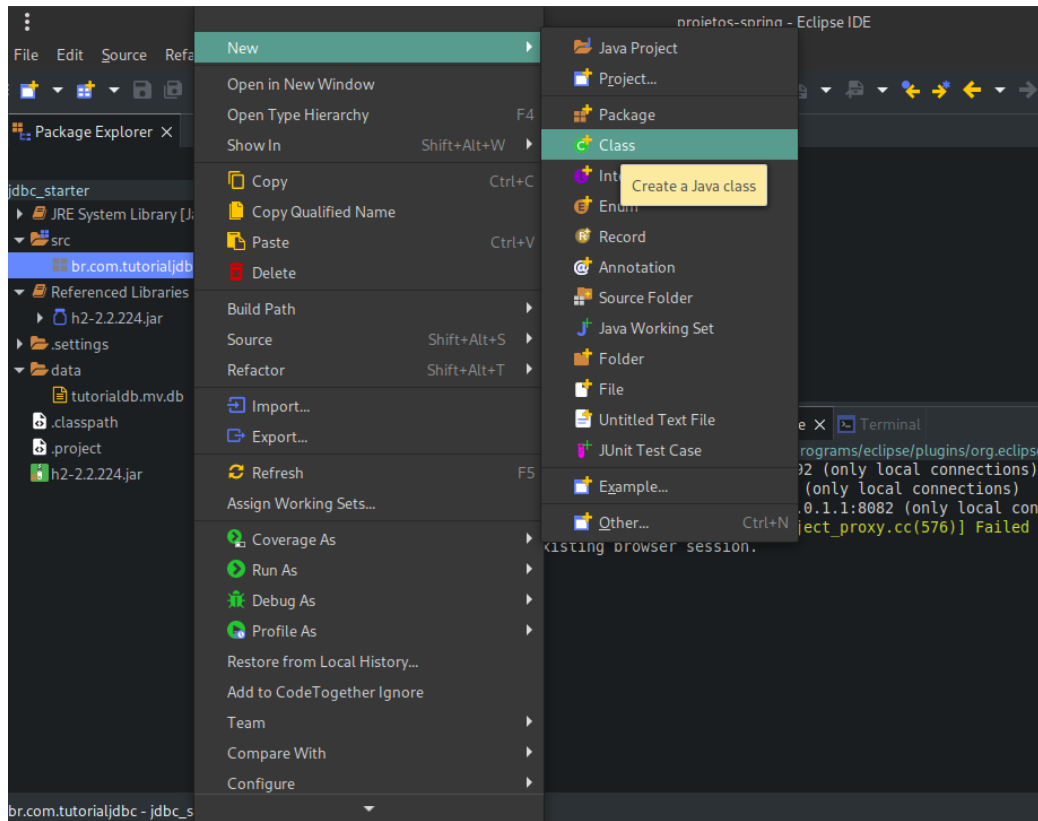
        super();
        this.id = id;
        this.name = name;
        this.email = email;
        this.phone = phone;
    }
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getPhone() {
        return phone;
    }
    public void setPhone(String phone) {
        this.phone = phone;
    }
    @Override
    public int hashCode() {
        return Objects.hash(id);
    }
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Contact other = (Contact) obj;
        return Objects.equals(id, other.id);
    }
    @Override
    public String toString() {
        return "Contact [id=" + id + ", name=" + name + ", email=" + email +
", phone=" + phone + "];"
    }
}

```

## Passo 2 - CRIAR A CONEXÃO COM O BANCO DE DADOS

O próximo passo é criar uma classe que realize o processo de conexão com o banco de dados para que posteriormente possamos executar os comandos necessários diretamente no banco.

Para isso clique com o botão direito sobre o nome do pacote **“br.com.tutorialjdbc”** e selecione a opção **new -> class**.



Na próxima tela do wizard coloque o nome da classe de **Database** e clique no botão **finish** na parte inferior direita da tela.



Em seguida adicione o código abaixo na classe Database:

```
package br.com.tutorialjdbc;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class Database {

    private static Connection connection;
    private static final String URL = "jdbc:h2:./data/tutorialdb";
    private static final String USER = "sa";
    private static final String PASSWORD = "java";

    /*Realiza o carregamento do driver do banco*/

    static {
        try {
            Class.forName("org.h2.Driver");
        } catch (ClassNotFoundException e) {
            System.out.println("Não foi possível carregar o Driver do
banco H2");
        }
    }

    /*
     * Cria uma conexão com o banco de dados
     */
}
```

```

        public static Connection getConnection() throws SQLException {
            try {
                if(connection == null || connection.isClosed()) {
                    //O método getConnection da classe DriverManager
recebe como parâmetros
                    //a URL de conexão do banco de dados, o nome do
usuário e a senha do usuário
                    //e retorna um objeto do tipo Connection que é a
referência da conexão com o banco de dados
                    connection = DriverManager.getConnection(URL, USER,
PASSWORD);
                }
            } catch (SQLException e) {
                System.out.println("Não foi possível realizar a conexão
com o banco de dados");
                throw e;
            }
            return connection;
        }
    }
}

```

Vamos agora analisar o código da classe Database:

Vamos começar a entender as primeiras declarações de variáveis utilizadas na classe.

```

8
9     private static Connection connection;
10    private static final String URL = "jdbc:h2:./data/tutorialdb";
11    private static final String USER = "sa";
12    private static final String PASSWORD = "java";

```

Na Linha 9 declaramos uma variável **connection** que é uma referência do tipo `java.sql.Connection`. Esta variável será utilizada para armazenar as informações da conexão com o banco de dados.

Logo em seguida, na linha 10 declaramos a constante URL, do tipo String, e atribuímos a ela a url de conexão com o banco de dados.

Na linha 11 declaramos a constante USER, do tipo String, e atribuímos a ela o nome do usuário que será utilizado para realizar a conexão com o banco de dados.

Por fim, na linha 12, declaramos a constante PASSWORD, também do tipo String, e armazenamos o valor da senha que será utilizada para realizar a conexão com o banco de dados.

Seguindo nossa análise temos o bloco de código estático a seguir:

```

16 static {
17     try {
18         Class.forName("org.h2.Driver");
19     } catch (ClassNotFoundException e) {
20         System.out.println("Não foi possível carregar o Driver do banco H2");
21     }
22 }
23

```

Este bloco de código, iniciado na linha 16, tenta realizar o carregamento do driver do banco de dados que será utilizado. Este processo é realizado chamando o método `forName` da classe `Class`.

Caso o método `forName` não encontre a classe que do driver (neste caso “org.h2.Driver”) ele lançará uma exceção do tipo `ClassNotFoundException`. Por isso na entre as linhas 19 e 21 colocamos um bloco `catch` para capturar o erro e apresentarmos uma mensagem no console indicando que o driver não foi carregado corretamente.

Por fim temos o método `getConnection()`, que é o responsável por realizar a conexão propriamente dita com o banco de dados.

```

24
25 /*
26  * Cria uma conexão com o banco de dados
27  */
28 public static Connection getConnection() throws SQLException {
29     try {
30         if(connection == null || connection.isClosed()) {
31             //O método getConnection da classe DriverManager recebe como parâmetros
32             //a URL de conexão do banco de dados, o nome do usuário e a senha do usuário
33             //e retorna um objeto do tipo Connection que é a referência da conexão com o banco de dados
34             connection = DriverManager.getConnection(URL, USER, PASSWORD);
35         }
36     } catch (SQLException e) {
37         System.out.println("Não foi possível realizar a conexão com o banco de dados");
38         throw e;
39     }
40     return connection;
41 }
42

```

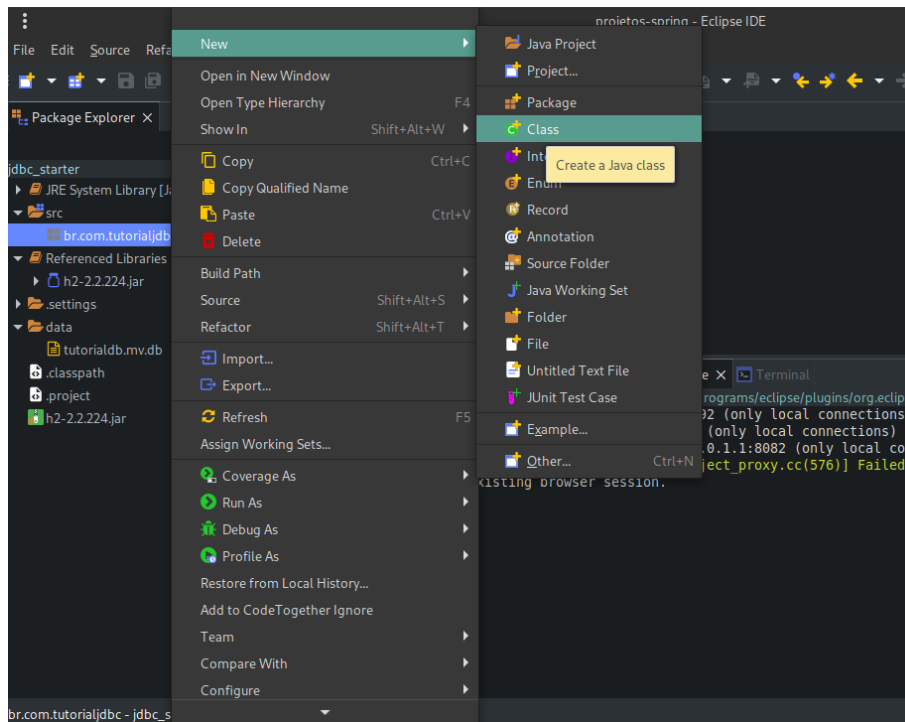
O método `getConnection` na linha 30, verifica se a conexão referenciada na variável `connection` é nula, não foi criada, ou se caso exista uma conexão, se ela está fechada. Caso as duas condições sejam verdadeiras então é solicitada uma nova conexão com o banco de dados através do método `getConnection` da classe `DriverManager`. Este método recebe como parâmetros as informações da URL de conexão, o nome de usuário e a senha necessária para realizar a conexão com o banco de dados e em seguida retorna uma nova conexão com o banco que é armazenada na referência estática `connection` que criamos lá na linha 9.

Por fim na linha 40 o método retorna a referência do objeto `connection` que contém as informações da conexão com o banco de dados.

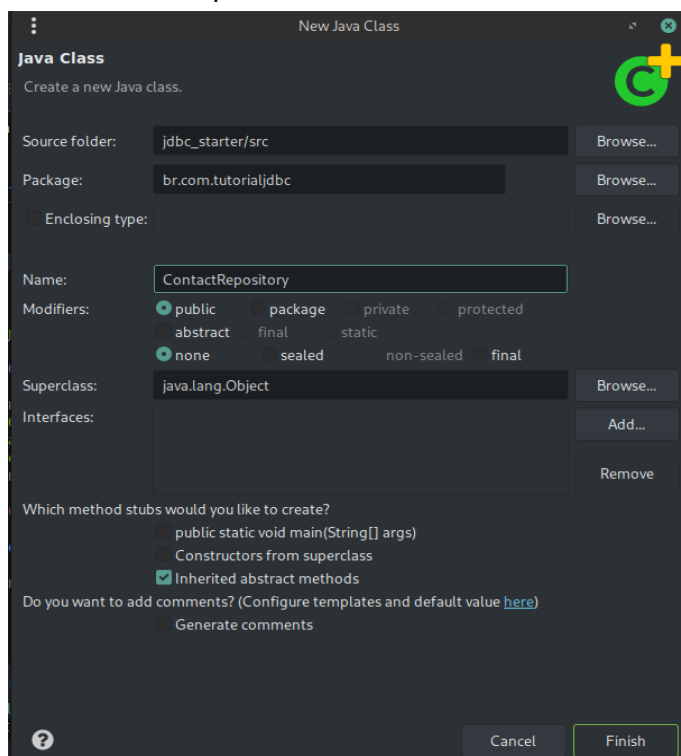
## Passo 3 - CRIAR A ABSTRAÇÃO DO REPOSITÓRIO DE DADOS

Agora que já temos a classe que representa os dados que queremos ler do banco de dados e também temos a classe que realiza a conexão com o banco de dados, vamos criar uma classe para representar a abstração do repositório de dados dentro da aplicação Java. Este tipo de classe é utilizada para representar a fonte de dados dentro da aplicação.

Para isso clique com o botão direito sobre o nome do pacote **“br.com.tutorialjdb”** e selecione a opção **new -> class**.



Na próxima tela do wizard coloque o nome da classe de **ContactRepository** e clique no botão **finish** na parte inferior direita da tela.





Dentro da classe ContactRepository adicione o código abaixo, que será responsável por carregar a lista de contatos do banco de dados para a aplicação Java.

```
package br.com.tutorialjdbc;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class ContactRepository {
    private static final String COL_ID = "id";
    private static final String COL_NAME = "name";
    private static final String COL_EMAIL = "email";
    private static final String COL_PHONE = "phone";

    public List<Contact> findAll(){
        List<Contact> records = new ArrayList<>();
        String sql = "select * from contacts";

        try {
            Connection db = Database.getConnection();
            PreparedStatement preparedStatement = db.prepareStatement(sql);
            ResultSet resultSet = preparedStatement.executeQuery();
            while(resultSet.next()) {
                records.add(fromResultSet(resultSet));
            }
            resultSet.close();
            preparedStatement.close();
            db.close();
        }catch(SQLException e) {
            throw new RuntimeException("Não foi possível executar a
operação no banco de dados");
        }

        return records;
    }

    private Contact fromResultSet(ResultSet resultSet) throws SQLException {
        Integer id = resultSet.getInt(COL_ID);
        String name = resultSet.getString(COL_NAME);
        String email = resultSet.getString(COL_EMAIL);
        String phone = resultSet.getString(COL_PHONE);

        return new Contact(id,name,email,phone);
    }
}
```

Vamos analisar o código da classe ContactRepository:

```
11 private static final String COL_ID = "id";
12 private static final String COL_NAME = "name";
13 private static final String COL_EMAIL = "email";
14 private static final String COL_PHONE = "phone";
```

Nas linhas de 11 a 14 temos constantes utilizadas para mapear o nome das colunas da tabela no banco de dados. COL\_ID faz o mapeamento do nome da coluna "id", COL\_NAME faz o mapeamento da coluna "name", COL\_EMAIL faz o mapeamento do nome da coluna "email" e por fim COL\_PHONE faz o mapeamento para o nome da coluna "phone".

Estes mapeamentos são utilizados posteriormente para facilitar a leitura das informações de cada coluna.

Vamos agora analisar o método findAll:

```
16 public List<Contact> findAll(){
17     List<Contact> records = new ArrayList<>();
18     String sql = "select * from contacts";
19
20     try {
21         Connection db = Database.getConnection();
22         PreparedStatement preparedStatement = db.prepareStatement(sql);
23         ResultSet resultSet = preparedStatement.executeQuery();
24         while(resultSet.next()) {
25             records.add(fromResultSet(resultSet));
26         }
27         resultSet.close();
28         preparedStatement.close();
29         db.close();
30     } catch (SQLException e) {
31         throw new RuntimeException("Não foi possível executar a operação no banco de dados");
32     }
33
34     return records;
35 }
```

O método na findAll é declarado na linha 16 e indica que deve retornar uma lista de contatos (List<Contact> findAll).

Na linha 17 criamos uma variável com o nome records, do tipo List<Contact>, que será responsável por armazenar todos os contatos que serão lidos no banco de dados.

Na linha 18, criamos uma variável chamada sql, do tipo String, e atribuímos a essa String o comando que queremos executar no banco de dados (select \* from contacts).

Na linha 21 criamos uma variável chamada db, do tipo java.sql.Connection e atribuímos a ela o valor do retorno da chamada ao método getConnection da classe Database que criamos no passo 2. Essa referência possui a ligação com o nosso banco de dados.

Na linha 22 criamos uma variável preparedStatement, do tipo java.sql.PreparedStatement. Atribuímos a essa variável o resultado da chamada db.prepareStatement(sql), uma referência do tipo preparedStatement representa um comando preparado para ser executado no banco de dados, este comando preparado é um comando que realiza validações para impedir que comandos com sql injection sejam executados no banco de dados.

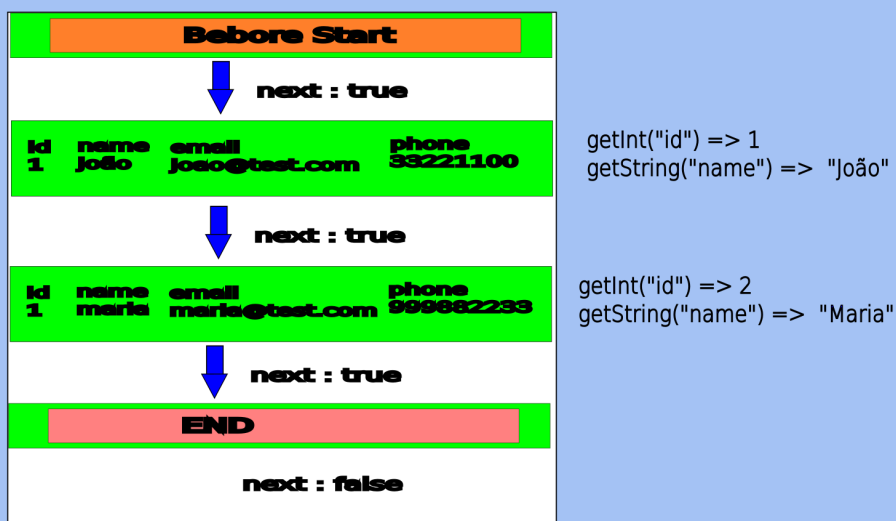
Na linha 23, temos a execução do comando propriamente dito no banco de dados, através da chamada ao método executeQuery(). Este comando retorna um objeto do tipo java.sql.ResultSet que é uma representação de um cursor de banco de dados.

## ATENÇÃO

O Objeto `ResultSet` possui alguns métodos para podermos realizar a leitura das informações que foram retornadas pelo banco de dados, entre eles estão os métodos `next()`, `getInt(columnName)`, `getString(columnName)`, entre outros.

Para entender um pouco melhor esse objeto vamos analisar o comportamento do `ResultSet` a partir da imagem abaixo:

### RESULT SET



O Objeto `ResultSet` possui internamente uma representação de todos os registros retornados na consulta realizada no banco de dados, mas apenas um registro pode ser acessível a cada leitura.

Quando o `ResultSet` é retornado ele vem em uma posição antes do primeiro registro e chamadas aos métodos `getInt`, `getString`, entre outros, não terão resultado.

Para posicionar o cursor interno do `ResultSet` em um registro é necessário chamar o método `next()`, este método retorna `true` indicando que conseguiu posicionar o cursor no próximo registro, e se não houver registros, retornará `false`.

Com o cursor interno do `ResultSet` posicionado em um registro, podemos chamar os métodos, `getInt`, `getString`, etc, passando para eles o nome da coluna que queremos realizar a leitura.

Ao chamarmos novamente o método `next()`, o cursor irá se posicionar no próximo registro.

Nas linhas 24 a 26 temos um laço de repetição que verifica se existem registros no cursos e para cada novo registro no resultset cria um novo objeto do tipo `Contato` e adiciona na lista de registros criada na linha 17. Cada registro é mapeado do `resultSet` para o tipo `Contact` através da chamada da função `fromResultSet`.

Na linha 34, o método `findAll` retorna a lista de registros populada com as informações de cada contato lido na base de dados.

Por fim, vamos analisar o método `fromResultSet`.

```
37 private Contact fromResultSet(ResultSet resultSet) throws SQLException {  
38     Integer id = resultSet.getInt(COL_ID);  
39     String name = resultSet.getString(COL_NAME);  
40     String email = resultSet.getString(COL_EMAIL);  
41     String phone = resultSet.getString(COL_PHONE);  
42  
43     return new Contact(id,name,email,phone);  
44 }  
45
```

O método `fromResultSet` recebe como parâmetro um objeto do tipo `java.sql.ResultSet` e retorna um Objeto do tipo `Contact`.

Neste método podemos ver que nas linhas 38 a 41 realizamos a leitura das informações de cada uma das colunas do registro que está sendo apontado pelo cursor do `ResultSet`.

Na linha 38 realizamos a leitura do valor armazenado na coluna “id” e retornamos esse valor no formato de um inteiro através da chamada da função `getInt` do `resultSet`.

Na linha 39 realizamos a leitura do valor armazenado na coluna “name” e retornamos esse valor no formato de uma `String` através da chamada da função `getString` do `resultSet`.

Na linha 40 realizamos a leitura do valor armazenado na coluna “email”.

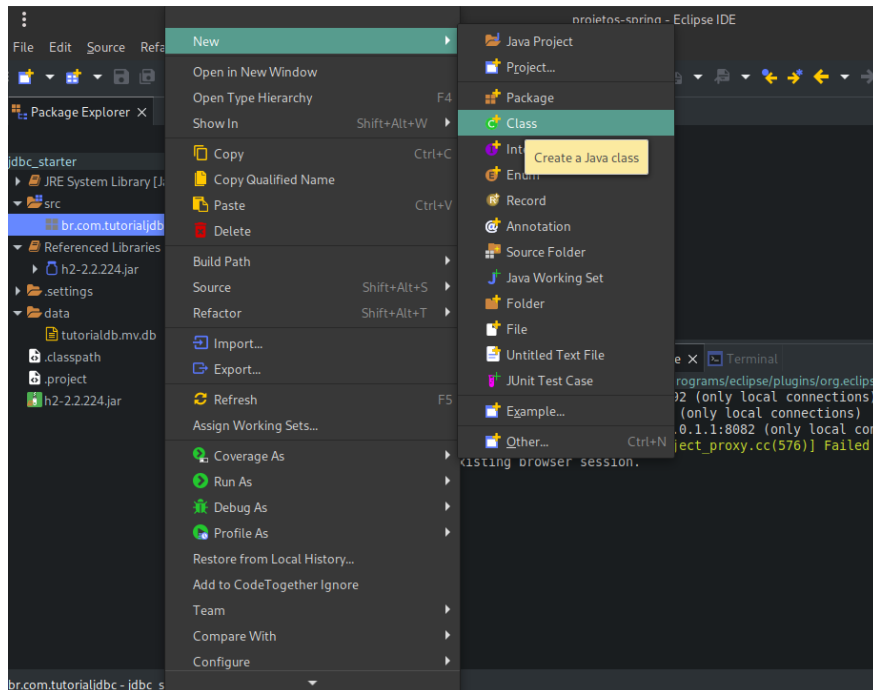
Na linha 41 realizamos a leitura do valor armazenado na coluna “phone”.

Já na linha 43, criamos um novo objeto do tipo `Contact` passando os valores lidos no `resultSet` anteriormente.

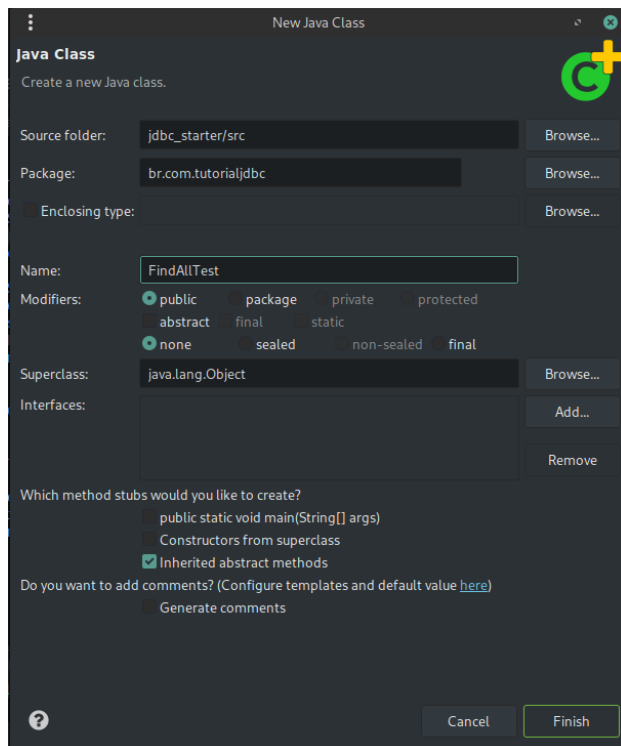
## Passo 04 - CRIAR A CLASSE DE TESTE

Agora que já temos todas as partes do quebra cabeças podemos criar a classe de testes que irá unir todo o código criado e apresentar a lista de contatos diretamente na aplicação Java.

Para isso clique com o botão direito sobre o nome do pacote **“br.com.tutorialjdbc”** e selecione a opção **new -> class**.



Na próxima tela do wizard coloque o nome da classe de **FindAllTest** e clique no botão **finish** na parte inferior direita da tela.



Em seguida adicione o código abaixo na classe de teste.

```

package br.com.tutorialjdbc;

import java.util.List;

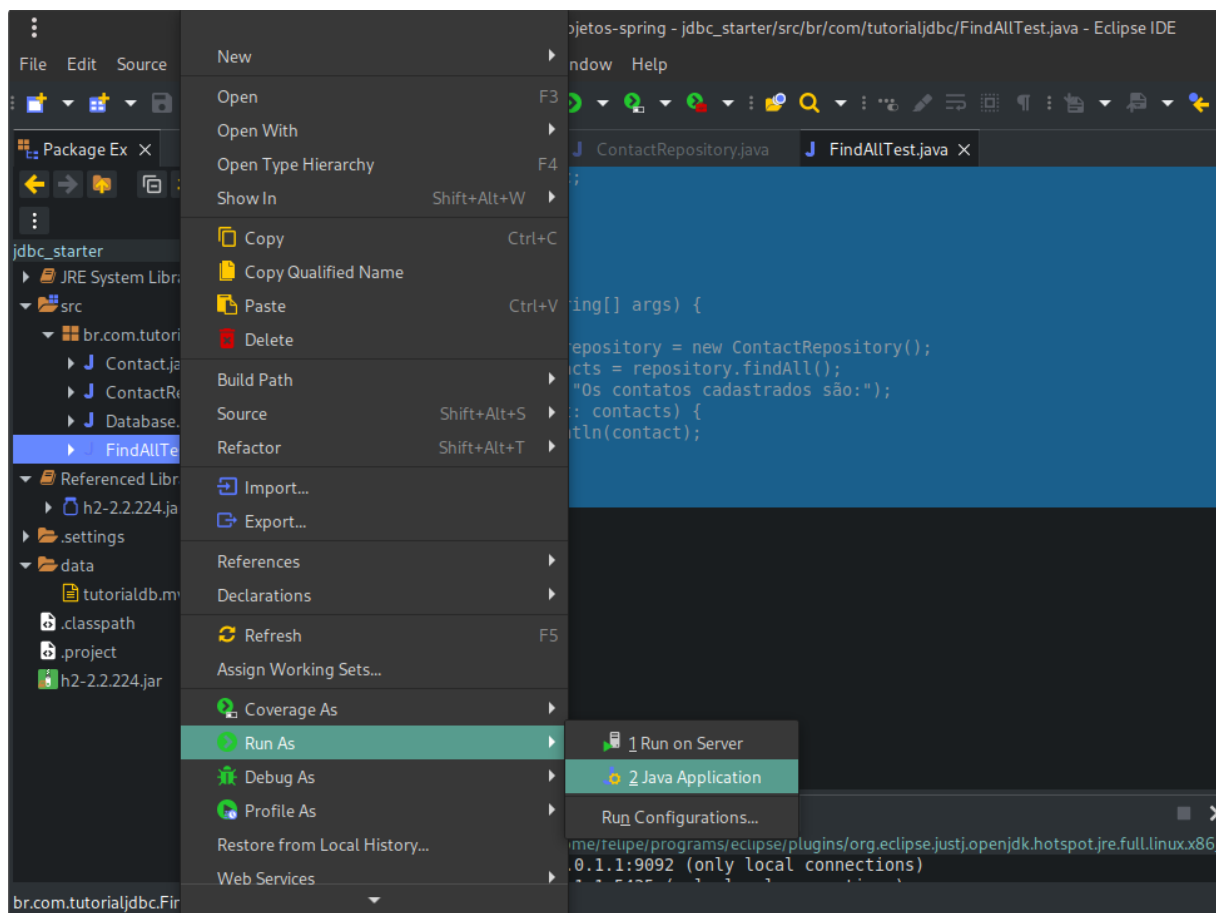
public class FindAllTest {

    public static void main(String[] args) {

        ContactRepository repository = new ContactRepository();
        List<Contact> contacts = repository.findAll();
        System.out.println("Os contatos cadastrados são:");
        for(Contact contact: contacts) {
            System.out.println(contact);
        }
    }
}

```

Com a classe de teste criada, rode o código de teste selecionando com o botão direito no nome da classe FindAllTest e escolha a opção Run As -> Java Application



Como resultado você deve ver no console a lista de contatos cadastrada no banco de dados.

```
Problems Javadoc Declaration Console X Terminal
<terminated> FindAllTest [Java Application] /home/felipe/programs/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.5.v20221102-0933/jre/bin/java (Nov 19, 2023, 3:32:00)
Os contatos cadastrados são:
Contact [id=1, name=João, email=joao@test.com, phone=999887766]
Contact [id=2, name=Maria, email=maria@test.com, phone=44553322]
Contact [id=3, name=Fernando, email=fernando@test.com, phone=988887755]
Contact [id=4, name=Kamila, email=kamila@test.com, phone=998880055]
Contact [id=5, name=Marcia, email=marcia@test.com, phone=994239977]
Contact [id=6, name=Adão, email=adao@test.com, phone=33221100]
Contact [id=7, name=Eva, email=eva@test.com, phone=55441122]
Contact [id=8, name=Ester, email=ester@test.com, phone=982345566]
Contact [id=9, name=Davi, email=davi@test.com, phone=987561249]
Contact [id=10, name=Sara, email=sara@test.com, phone=34562312]
```