WELCOME BACK
&
THANK YOU

Advance Java
Crash Course

For TD Bank

MEET YOUR CRASH COURSE TEAM

TANGY F.

CEO

WILLIAM D.

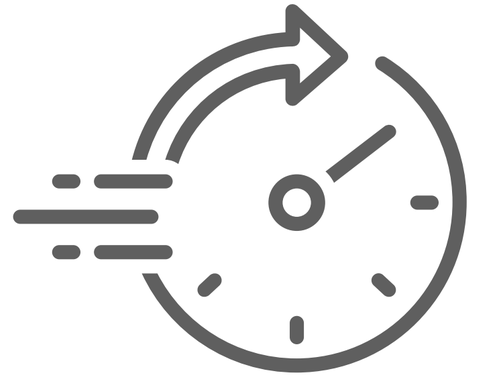
DEVELOPER

TONY J.

T.A * DEVELOPER

Rapid

TRIVIA

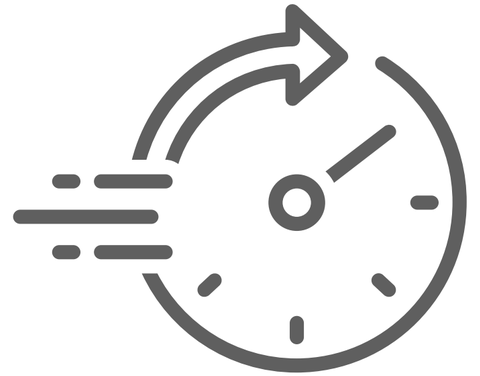


Rapid Trivia

What agency developed foundation of the internet?

Rapid

TRIVIA



Rapid Trivia

Packet switching, the foundation of the internet was developed by the United States Department of Defense's Advanced Research Projects Agency, currently known as (DARPA)

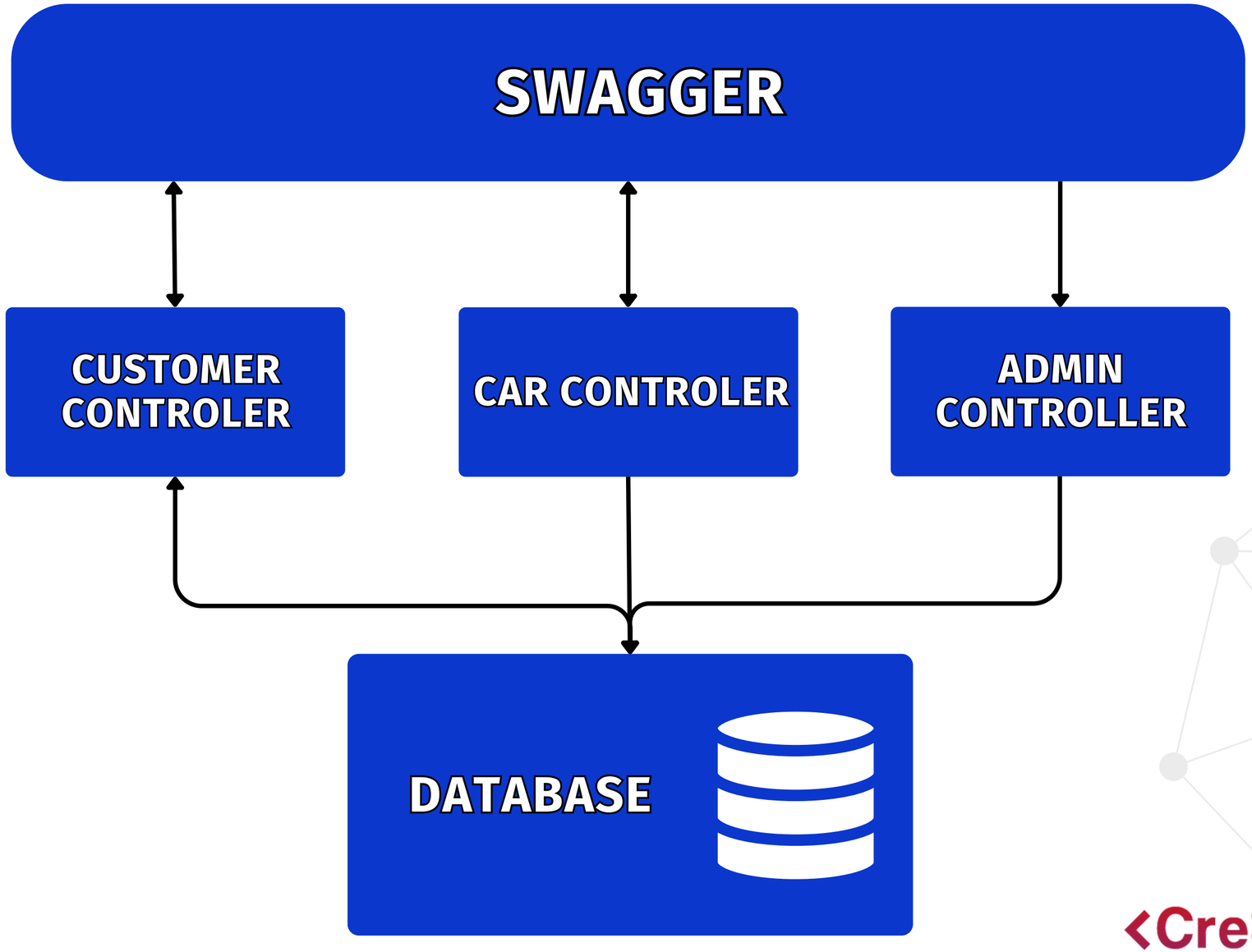
TRAINING DAY 6

TODAY'S AGENDA

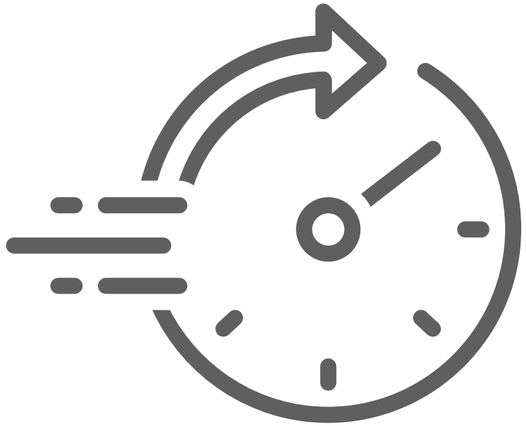
1. Yesterday's Coding Exercise to Go Rapid Review
2. Parallel Streams
3. Flat Maps
4. Connection Pooling

BITE SIZE
CAR RENTAL APP

WE ARE BUILDING



Rapid Review
Lesson 1



Rapid Review

Adding a Lambda expression to CarService.getAllCars()

YERSTERDAYS CODING EXERCISE TO GO

In our **CarService.getAllCars()** we use stream to filter out booked car from the final list (see bellow) refactor this code to use lambda expression. The result will be a one line of code.

```
cars = cars.stream()
    .filter(new Predicate<CarResponseDto>() {
        @Override
        public boolean test(CarResponseDto carResponseDto) {
            return !carResponseDto.getIsBooked();
        }
    }).collect(Collectors.toList());

return cars;
```

YESTERDAY'S CODING TO GO REVIEW

Before

```
cars = cars.stream()
    .filter(new Predicate<CarResponseDto>() {
        @Override
        public boolean test(CarResponseDto carResponseDto) {
            return !carResponseDto.getIsBooked();
        }
    }).collect(Collectors.toList());

return cars;
```

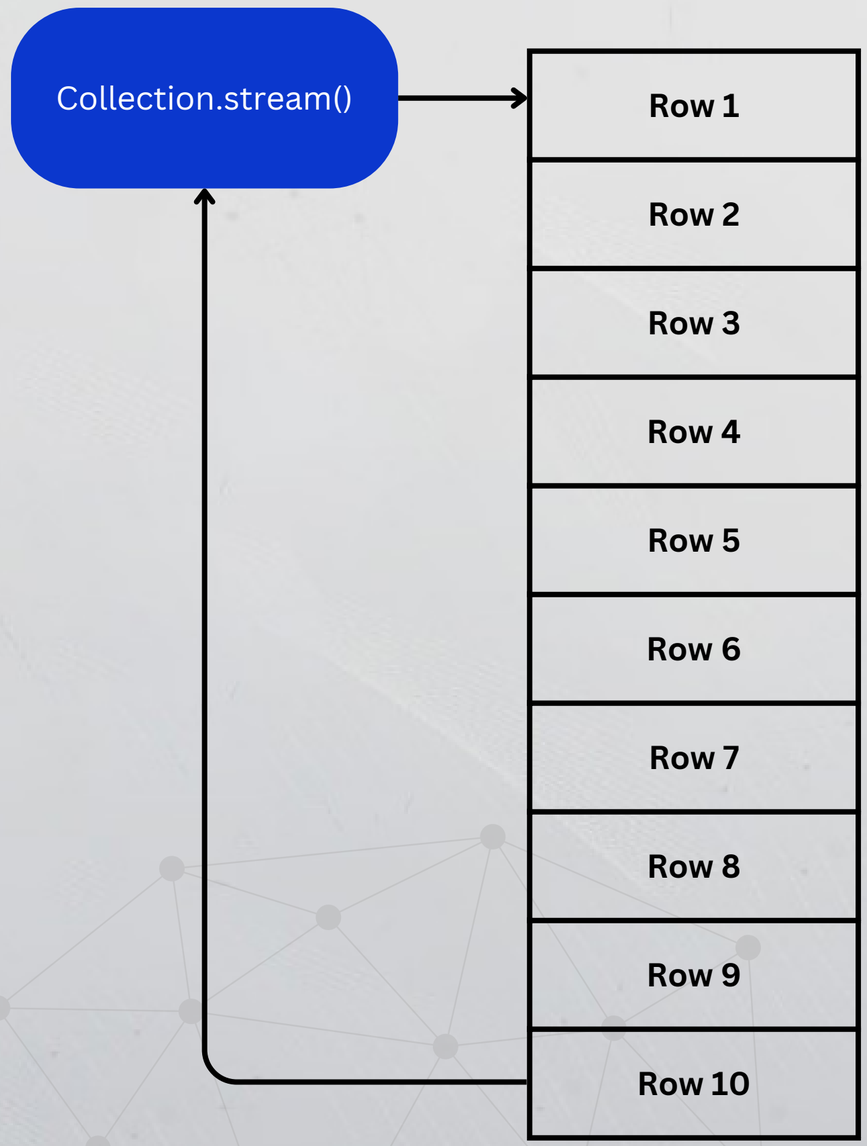
After

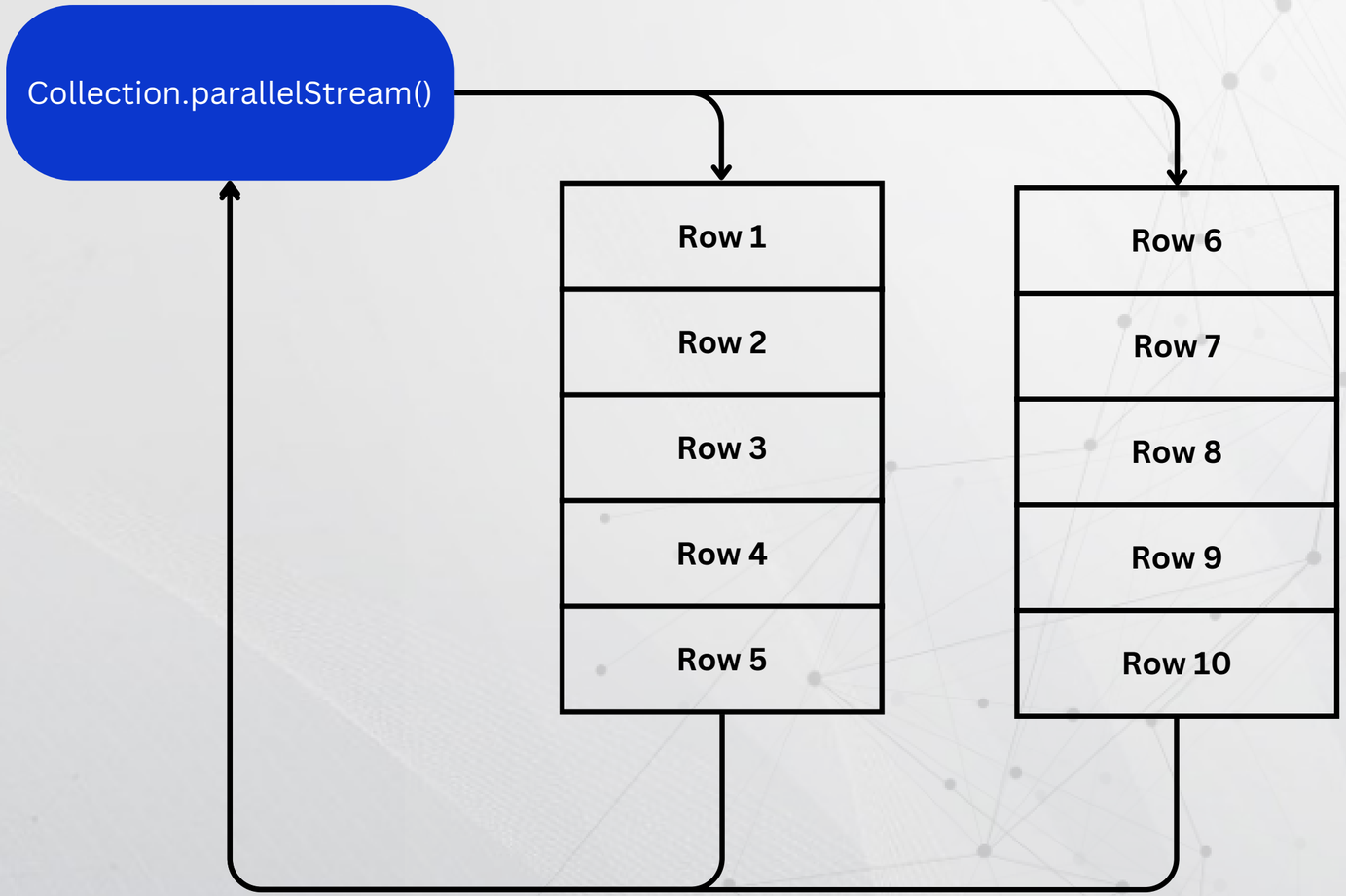
```
cars = cars.stream().filter(carResponseDto -> carResponseDto.getIsBooked() == false).collect(Collectors.toList());
```

DAY 6
LESSON 2

Parallel Streams

PARALLEL STREAMS





PARALLEL STREAMS

Using streams

```
customer.getCarList().stream()
        .forEach(car -> totalPrice.add(car.getPrice()));
```

Using parallel streams

```
customer.getCarList().parallelStream()
        .forEach(car -> totalPrice.add(car.getPrice()));
```

Let's Take a Look.

DAY 6
LESSON 3

FlatMaps

FLATMAP

Collection 1	Collection 2	Collection 3
Element 1	Element 4	Element 7
Element 2	Element 5	Element 8
Element 3	Element 6	Element 9

After Flatmap
Element 1
Element 2
Element 3
Element 4
Element 5
Element 6
Element 7
Element 8
Element 9

FLATMAP

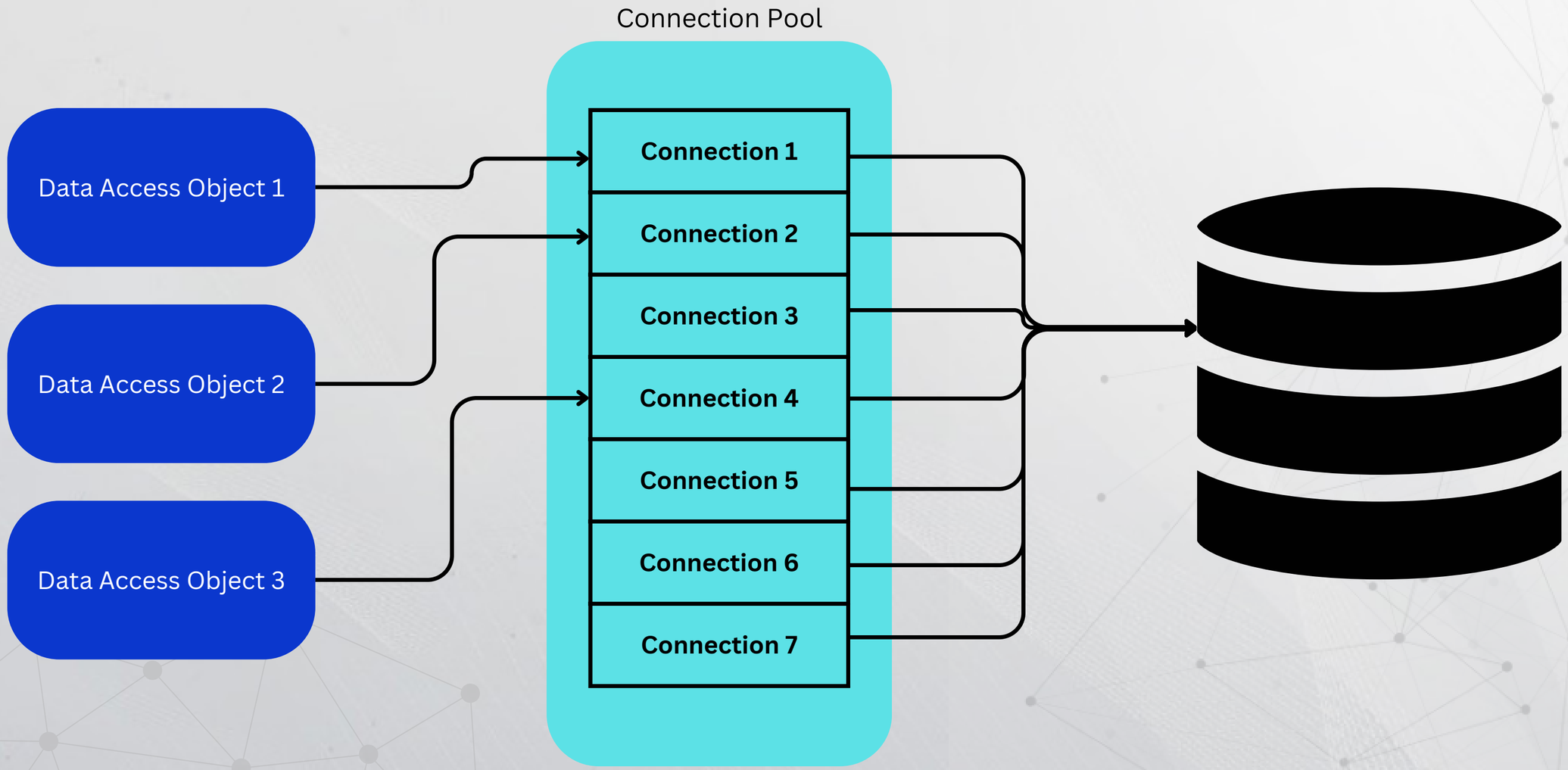
```java
List<List<String>> listOfLists = Arrays.asList(
    Arrays.asList("Element 1", "Element 2", "Element 3"),
    Arrays.asList("Element 4", "Element 5", "Element 6"),
    Arrays.asList("Element 7", "Element 8", "Element 9")
);

// Using flatMap to transform and flatten the stream
List<String> flattenedList = listOfLists.stream()
    .flatMap(list -> list.stream()) // Apply a function to each list and flatten
    .collect(Collectors.toList());

System.out.println(flattenedList);
```

Let's Take a Look.

DAY 6
LESSON 4

Connection Pooling

CONNECTION POOLING



CONNECTION POOLING

```
implementation 'com.zaxxer:HikariCP:4.0.3'
```

```
import com.zaxxer.hikari.HikariConfig;
import com.zaxxer.hikari.HikariDataSource;
```

```
HikariConfig config = new HikariConfig();
config.setJdbcUrl(jdbcUrl:"jdbc:mysql://localhost:3306/autohire");
config.setUsername(username:"autohire");
config.setPassword(password:"autohire");

// Create a data source using HikariCP
HikariDataSource dataSource = new HikariDataSource(config);

try (Connection connection = dataSource.getConnection()) {
    // Perform database operations using the connection
    String sql = "SELECT * FROM customer";
    try (PreparedStatement preparedStatement = connection.prepareStatement(sql);
         ResultSet resultSet = preparedStatement.executeQuery()) {
        while (resultSet.next()) {
            int customerId = resultSet.getInt("id");
            String name = resultSet.getString("first_name");
            String lastName = resultSet.getString("last_name");
            System.out.println("Customer ID: " + customerId + ", Customer Name: " + name + " " + lastName);
        }
    }
} catch (SQLException e) {
    e.printStackTrace();
} finally {
    // Close the data source to release resources
    dataSource.close();
}
```

Let's Take a Look.

CODING TRIVA QUESTION

The code bellow will compile and execute but will throw and error in some condition. Which condition will throw this error and how can we avoid it.

```
private void setDomestic1(List <Domestic> domesticList) // Generics
{
    if (domesticList.size() >= 0) {
        Cat firstCat = (Cat) domesticList.get(0);
        System.out.println("Cats do " + firstCat.doMeaw());
    }
}
```

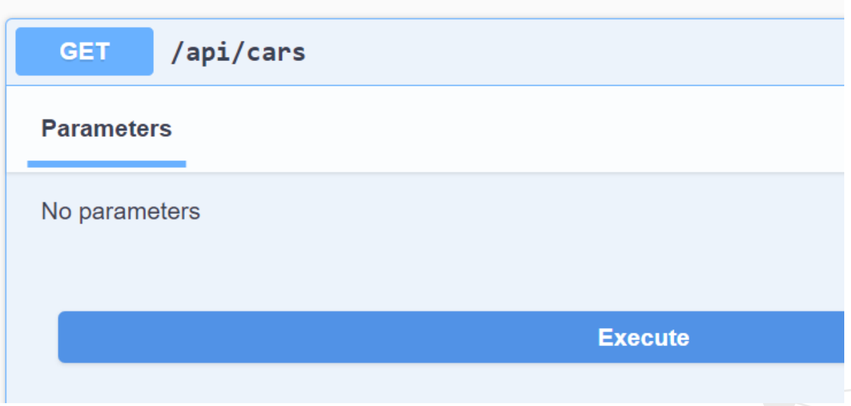
CODING TRIVA ANSWER

```java
private void setDomestic1(List <Domestic> domesticList) // Generics
{
    if (domesticList.size() >= 0 && domesticList.get(0) instanceof Cat) {
        Cat firstCat = (Cat) domesticList.get(0);
        System.out.println("Cats do " + firstCat.doMeaw());
    }
}
```

CODING EXERCISE TO GO

We have seen connection pooling in the DBAccessCP main class. Lets now add it to our autohire app by going to the **DatabaseConfing** class, add a new method **getConnectionPool()** that implements a connection access using connection pool via Hikari. Then ensure that when you run **/api/cars**, the execution uses **getConnectionPool()** instead of the existing **getConnection().** Use the following from **DBAccessCP.java** class as guide

```
HikariConfig config = new HikariConfig();
config.setJdbcUrl(jdbcUrl:"jdbc:mysql://localhost:3306/autohire");
config.setUsername(username:"autohire");
config.setPassword(password:"autohire");

// Create a data source using HikariCP
HikariDataSource dataSource = new HikariDataSource(config);

try (Connection connection = dataSource.getConnection()) {
    // Perform database operations using the connection
    String sql = "SELECT * FROM customer";
    try (PreparedStatement preparedStatement = connection.prepareStatement(sql);
         ResultSet resultSet = preparedStatement.executeQuery()) {
        while (resultSet.next()) {
            int customerId = resultSet.getInt("id");
            String name = resultSet.getString("first_name");
            String lastName = resultSet.getString("last_name");
            System.out.println("Customer ID: " + customerId + ", Customer Name: " + name + " " + lastName);
        }
    }
} catch (SQLException e) {
    e.printStackTrace();
} finally {
    // Close the data source to release resources
    dataSource.close();
}
```



THANK YOU

Crash Course

We will see you Monday