WELCOME BACK
&
THANK YOU

Advance Java
Crash Course

For TD Bank

MEET YOUR CRASH COURSE TEAM

TANGY F.

CEO

WILLIAM D.

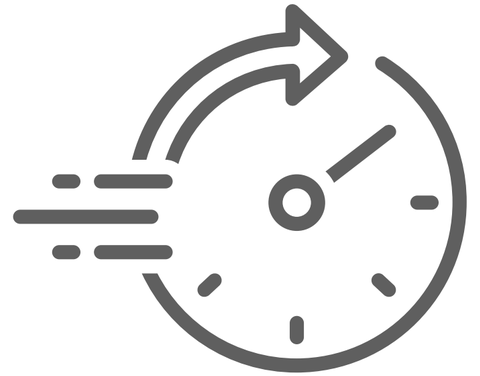
DEVELOPER

TONY J.

T.A * DEVELOPER

Rapid Review

TRIVIA

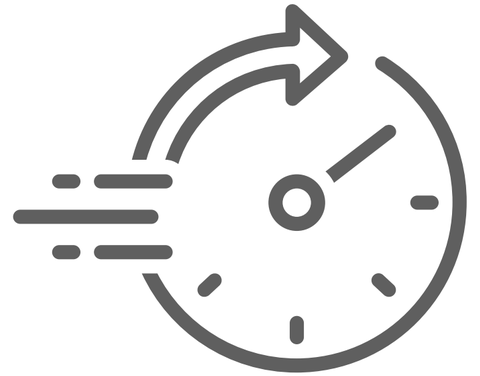


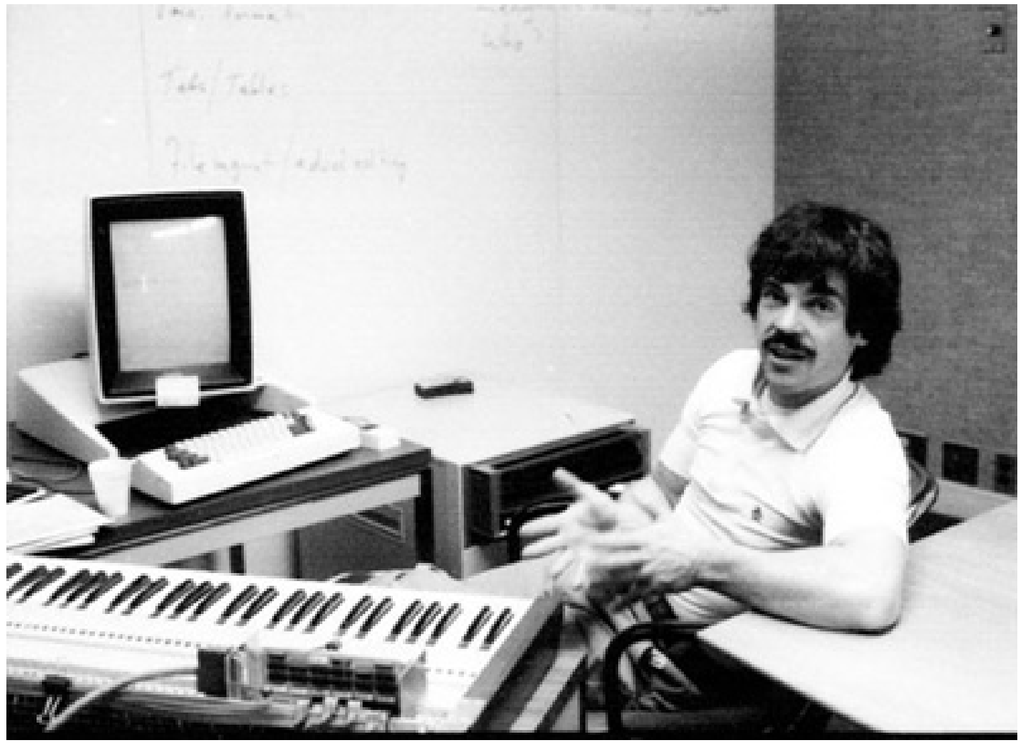
Rapid Trivia

Who developed the first Object Oriented Programming language

Rapid Review

TRIVIA





Alan Kay

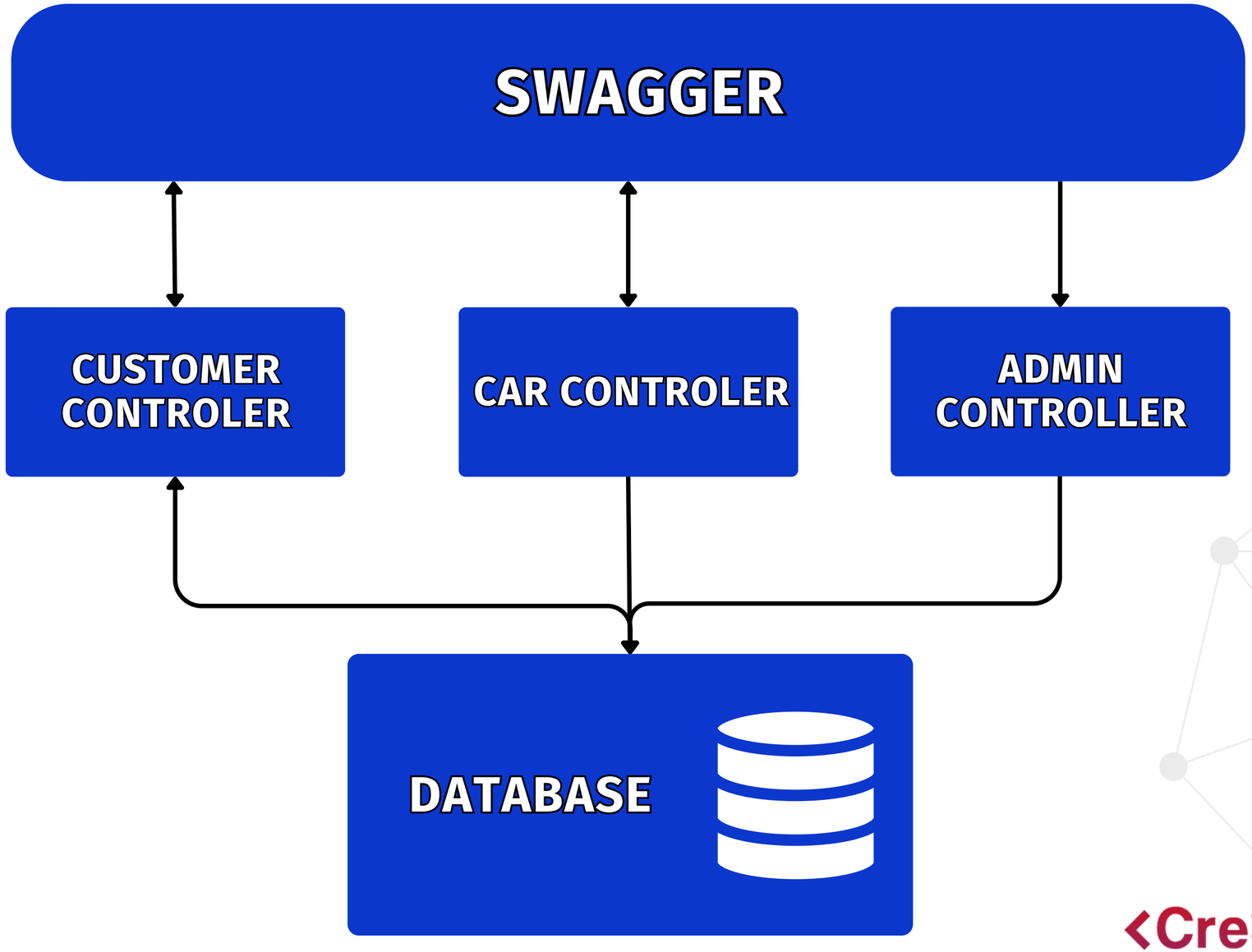
TRAINING DAY 5

TODAY'S AGENDA

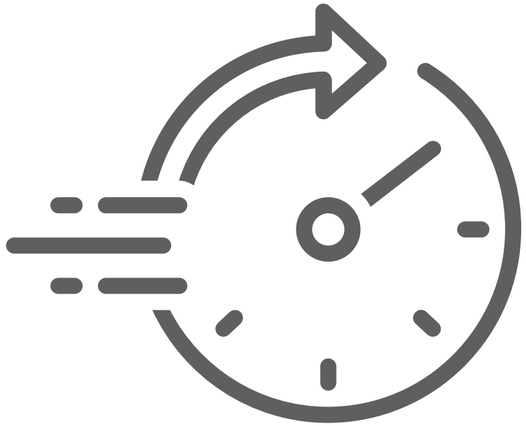
1. Yesterday's Coding Exercise to Go Rapid Review
2. Wildcards
3. Streams
4. Functional Interfaces
5. Lambda Expressions

BITE SIZE
CAR RENTAL APP

WE ARE BUILDING



Rapid Review
Lesson 1

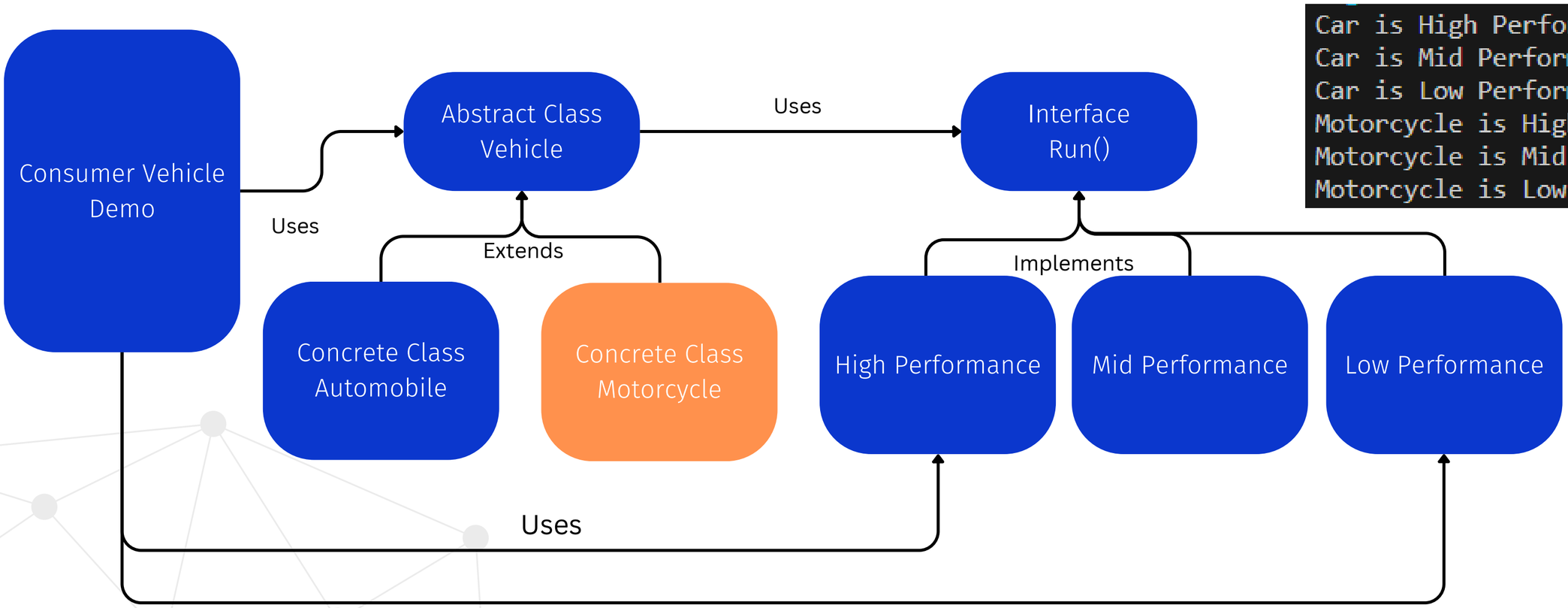


Rapid Review

Adding motorcycle vehicle type to the bridge pattern code sample

YERSTERDAYS CODING EXERCISE TO GO

Remember this, we added mid performance, now add Motorcycle Vehicle type in order to print what's shown on the right.




```
Car is High Performance
Car is Mid Performance
Car is Low Performance
Motorcycle is High Performance
Motorcycle is Mid Performance
Motorcycle is Low Performance
```

YESTERDAY'S CODING TO GO REVIEW

src > main > java > com > mains > BridgePattern > Motorcycle.java > Motorcycle >

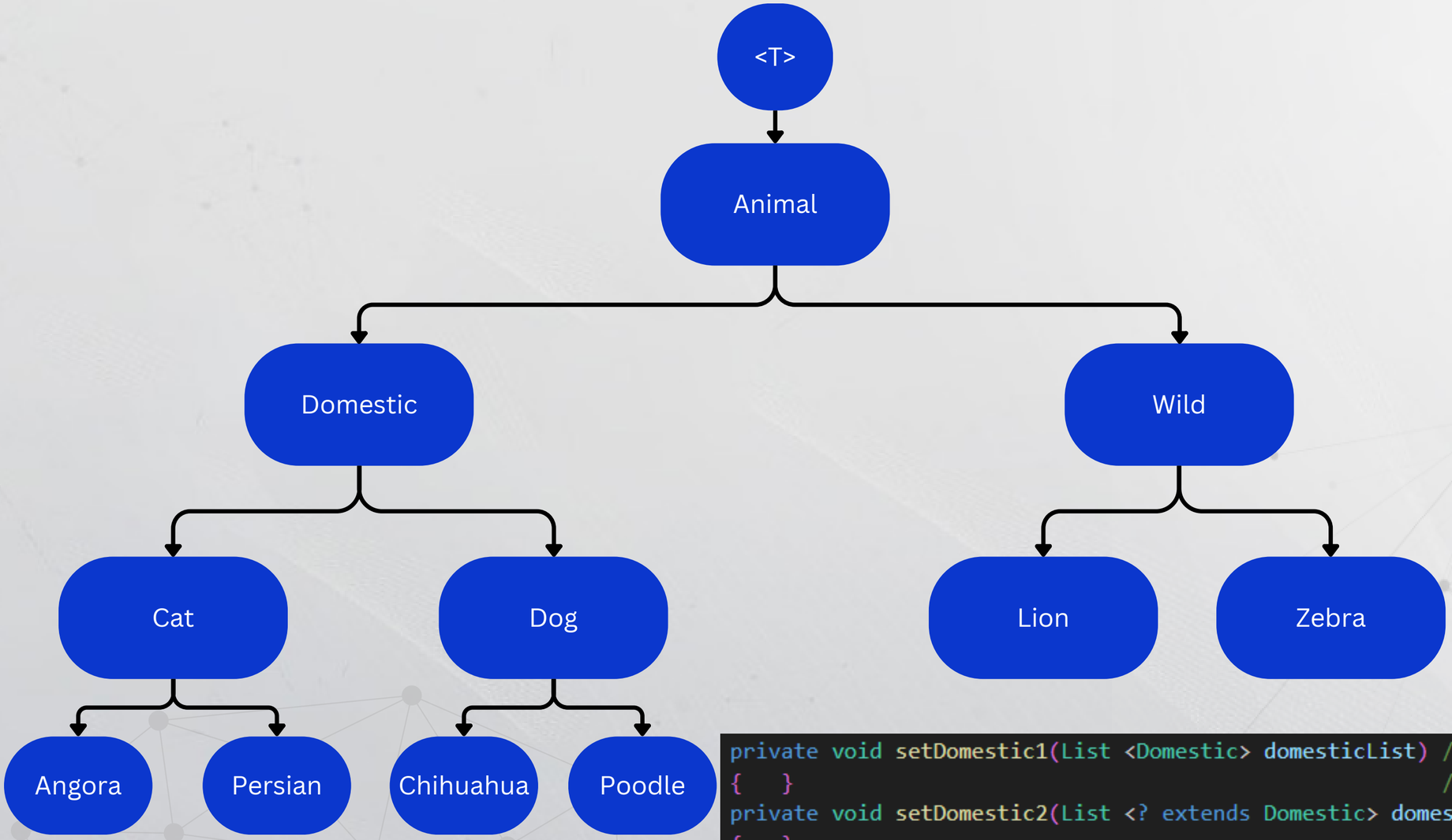
```java
package com.mains.BridgePattern;
public class Motorcycle extends Vehicle {

    public Motorcycle(RunAPI runAPI) {
        super(runAPI);
    }

    public void run() {  // implement high level logic here
        runAPI.runVehicle(vehicleType:"Motorcycle");
    }
}

```

```java
public class VehicleDemo {
    Run | Debug
    public static void main(String[] args) {

        ArrayList <Vehicle> vehicleList = new ArrayList<>();
        vehicleList.add(new Car(new HighPerformance()));
        vehicleList.add(new Car(new MidPerformance()));
        vehicleList.add(new Car(new LowPerformance()));
        vehicleList.add(new Motorcycle(new HighPerformance()));
        vehicleList.add(new Motorcycle(new MidPerformance()));
        vehicleList.add(new Motorcycle(new LowPerformance()));

        for (Vehicle vList : vehicleList) {
            vList.run();
        }
    }
}
```

DAY 5
LESSON 2

Wildcards

GENERICS AND WILDCARDS



Let's see the code.

```
private void setDomestic1(List <Domestic> domesticList) // Generics
{   }                                                   // List with Domestic
private void setDomestic2(List <? extends Domestic> domesticList) // Generecs & lower bound wildcards
{   }                                                             // List with Domestic or lower classes
private void setDomestic3(List <? super Domestic> domesticList) // Generics & upper bound wildcards
{   }                                                           // List with Domestic or upper classes
```

DAY 5
LESSON 3

Streams

STREAMS

Traditional Way

```
List<Client> filteredCli = filterByAge(clients);
```

```
private static List<Client> filterByAge(List<Client> clients) {

    List<Client> filteredClients = new ArrayList<>();
    for (Client client : clients) {
        if (client.getAge() >= 30) {
            filteredClients.add(client);
        }
    }
    return filteredClients;

}
```

Let's Take a Look.

Using Streams

```
List<Client> filteredClients = clients.stream().filter(new java.util.function.Predicate<Client>() {
    @Override
    public boolean test(Client clients) {
        return clients.getAge() >= 30;
    }
}).collect(Collectors.toList());
```

DAY 5
LESSON 4

Functional Interfaces

FUNCTIONAL INTERFACE

```java
@FunctionalInterface
interface FunctionalCalculator {
    int multiply(int number1, int number2);
}
```

```java
package java.lang;

@FunctionalInterface
public interface Runnable {
   void run();
}
```

Let's Take a Look.

Java provides several functional interfaces like:

- Runnable (above)
- Callable
- Comparator
- Predicate
- Consumer

DAY 5
LESSON 5

Lambda Expressions

LAMBDA EXPRESSIONS

From our stream sample before

```
List<Client> filteredClients = clients.stream().filter(new java.util.function.Predicate<Client>() {
    @Override
    public boolean test(Client clients) {
        return clients.getAge() >= 30;
    }
}).collect(Collectors.toList());
```

Now using Lambda

```
List<Client> filteredClients = clients.stream()
    .filter(client -> client.getAge() >= 30)
    .collect(Collectors.toList());
```

Let's Take a Look.

General Format of Lambda Expression;

```
(parameters) -> expression
(parameters) -> { expression }
```

CODING TRIVA QUESTION

As you can see, the compiler is not liking the code bellow, something is missing, what is it?

```
List<CarResponseDto> cars = new ArrayList<>();


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

CODING TRIVA ANSWER

<CarResponseDto> generic type was missing

```
List<CarResponseDto> cars = new ArrayList<>();

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

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;
```

THANK YOU

Crash Course

We will see you on Friday