# Auto Manager

**Course name:** *420-SF2-RE Data Structures and Object-Oriented Programming*

**Student name:** Antonia Stoleru, ID: 2484936

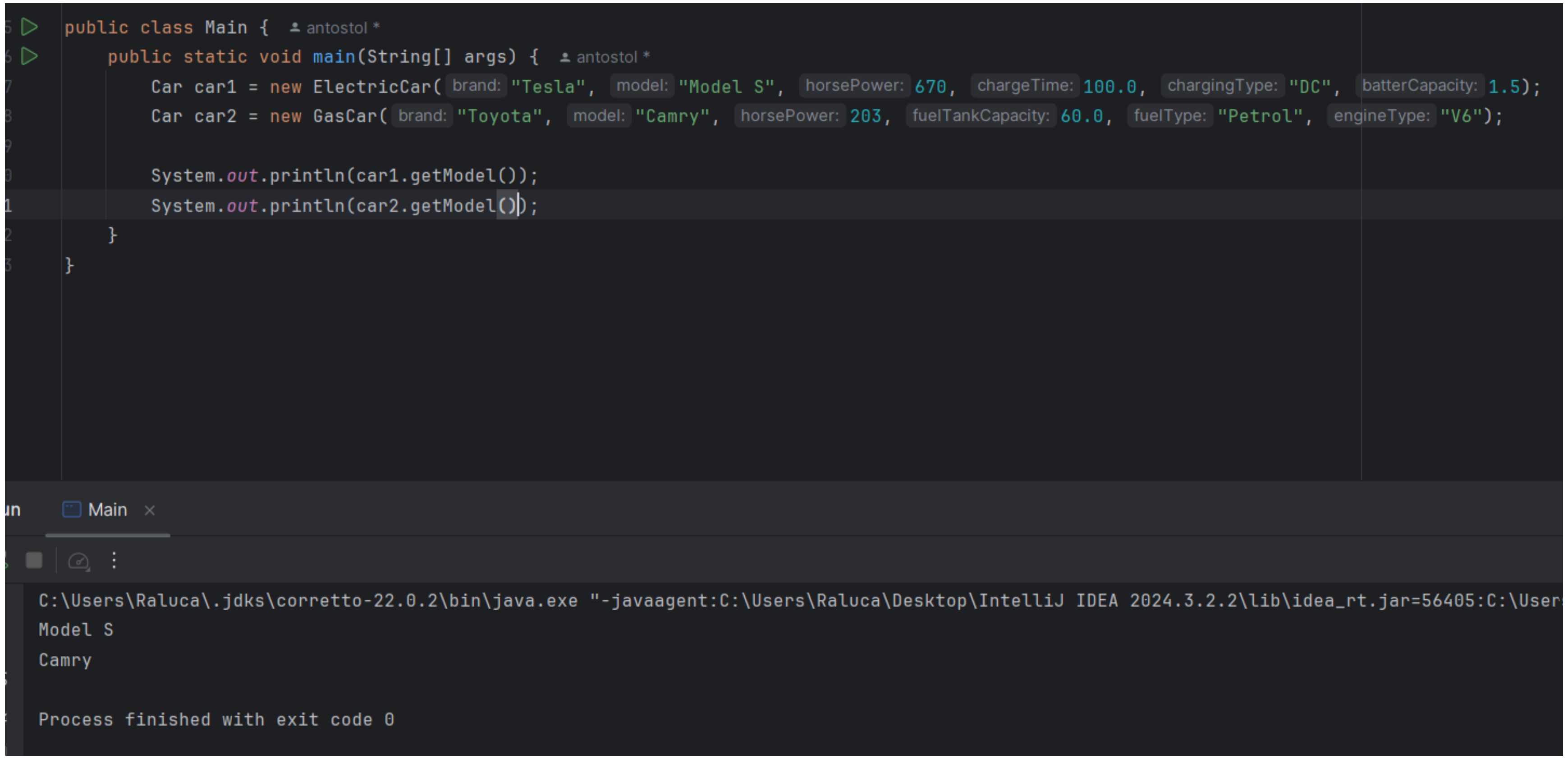
## Table of contents

## Project description

- The project is a simulation of a car dealership. The user can reach out to different dealerships (online or in person) to see various cars (gas or electric). The user can search, view, and compare different vehicles as well as different dealerships. This program also allows for the loading and storing of information from files regarding dealerships.

# Program Features & Screenshots

- Abstract Class & Inheritance (Output & Execution examples)



```
public class Main {  antostol *
    public static void main(String[] args) {  antostol *
        Car car1 = new ElectricCar( brand: "Tesla",  model: "Model S",  horsePower: 670,  chargeTime: 100.0,  chargingType: "DC",  batterCapacity: 1.5);
        Car car2 = new GasCar( brand: "Toyota",  model: "Camry",  horsePower: 203,  fuelTankCapacity: 60.0,  fuelType: "Petrol",  engineType: "V6");

        System.out.println(car1.getModel());
        System.out.println(car2.getModel());
    }
}
```

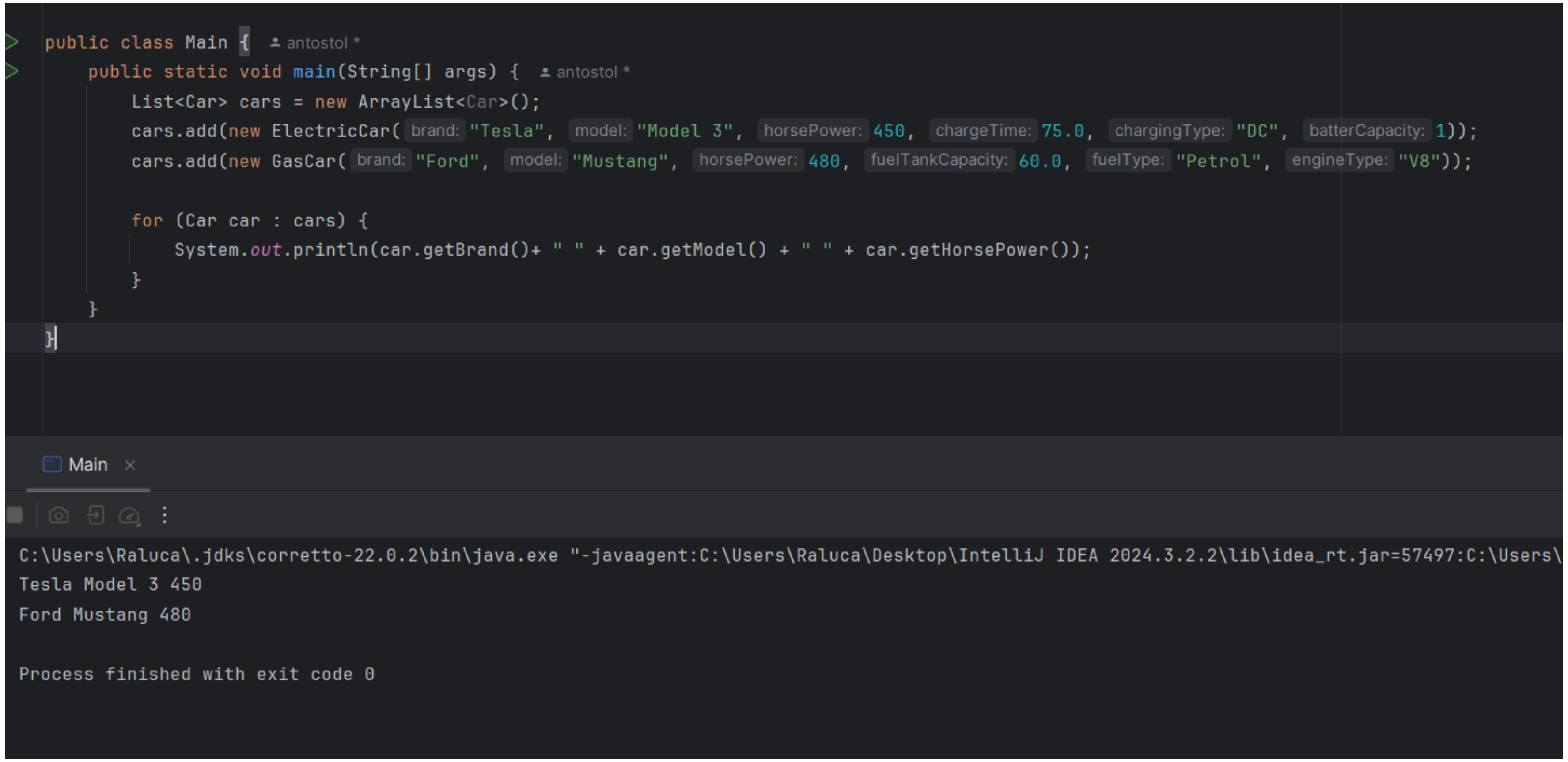
```
C:\Users\Raluca\.jdks\corretto-22.0.2\bin\java.exe "-javaagent:C:\Users\Raluca\Desktop\IntelliJ IDEA 2024.3.2.2\lib\idea_rt.jar=56405:C:\User
Model S
Camry

Process finished with exit code 0
```

## Program Features & Screenshots

- Polymorphism:
  - Collections List<Car>, List<Dealership> store different subclass objects
  - Methods writeToFile() & readFromFile() use polymorphism with Car objects
  - Methods isHighPerfomance() & compareTo are overriden

## Program Features & Screenshots

- Polymorphism: (Output & Execution examples)

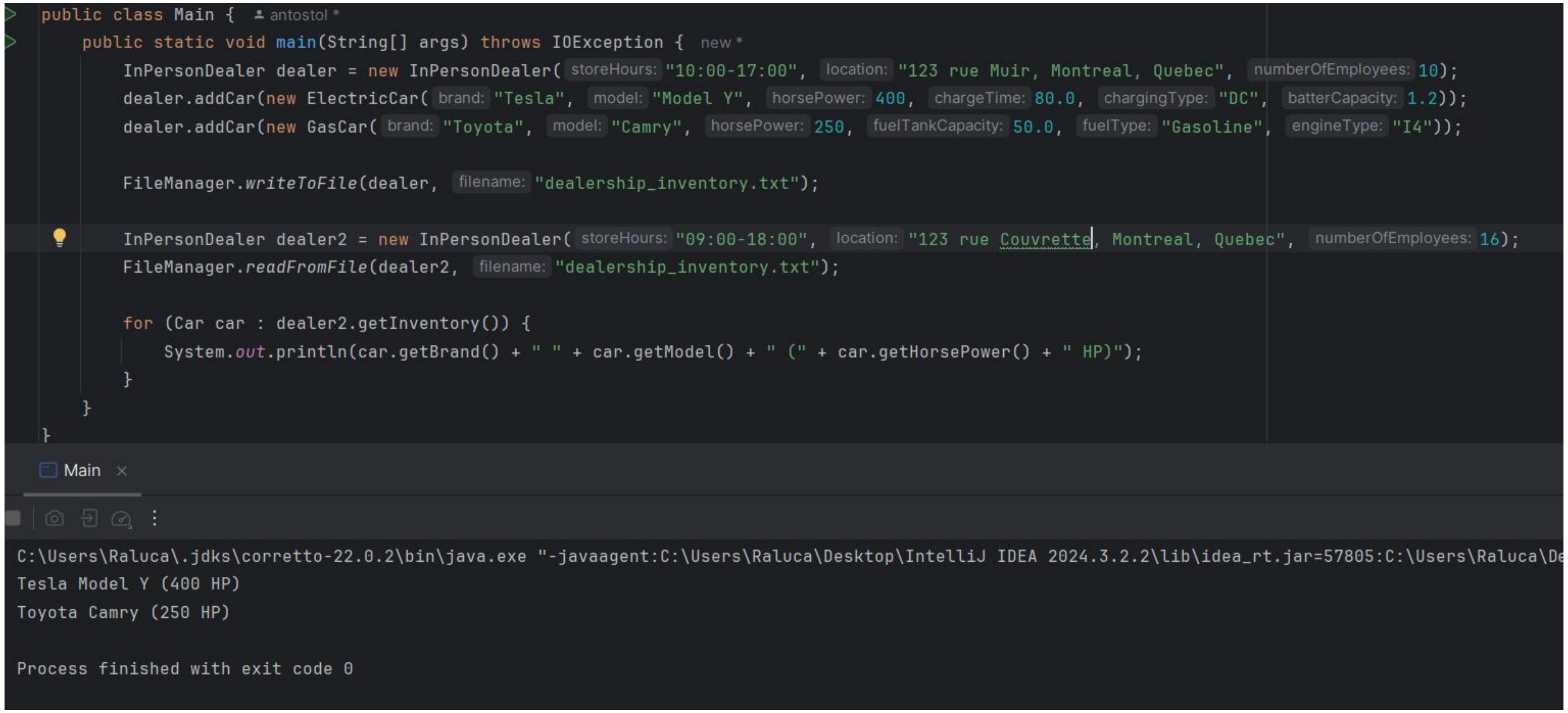


```
public class Main {  antostol *
    public static void main(String[] args) {  antostol *
        List<Car> cars = new ArrayList<Car>();
        cars.add(new ElectricCar( brand: "Tesla",  model: "Model 3",  horsePower: 450,  chargeTime: 75.0,  chargingType: "DC",  batterCapacity: 1));
        cars.add(new GasCar( brand: "Ford",  model: "Mustang",  horsePower: 480,  fuelTankCapacity: 60.0,  fuelType: "Petrol",  engineType: "V8"));

        for (Car car : cars) {
            System.out.println(car.getBrand()+ " " + car.getModel() + " " + car.getHorsePower());
        }
    }
}
```

```
C:\Users\Raluca\.jdks\corretto-22.0.2\bin\java.exe "-javaagent:C:\Users\Raluca\Desktop\IntelliJ IDEA 2024.3.2.2\lib\idea_rt.jar=57497:C:\Users\
Tesla Model 3 450
Ford Mustang 480

Process finished with exit code 0
```

# Program Features & Screenshots

- Polymorphism: (Output & Execution examples)



```
public class Main {  antostol *
    public static void main(String[] args) throws IOException {  new *
        InPersonDealer dealer = new InPersonDealer( storeHours: "10:00-17:00",  location: "123 rue Muir, Montreal, Quebec",  numberOfEmployees: 10);
        dealer.addCar(new ElectricCar( brand: "Tesla",  model: "Model Y",  horsePower: 400,  chargeTime: 80.0,  chargingType: "DC",  batterCapacity: 1.2));
        dealer.addCar(new GasCar( brand: "Toyota",  model: "Camry",  horsePower: 250,  fuelTankCapacity: 50.0,  fuelType: "Gasoline",  engineType: "I4"));

        FileManager.writeToFile(dealer,  filename: "dealership_inventory.txt");

        InPersonDealer dealer2 = new InPersonDealer( storeHours: "09:00-18:00",  location: "123 rue Couvrette, Montreal, Quebec",  numberOfEmployees: 16);
        FileManager.readFromFile(dealer2,  filename: "dealership_inventory.txt");

        for (Car car : dealer2.getInventory()) {
            System.out.println(car.getBrand() + " " + car.getModel() + " (" + car.getHorsePower() + " HP)");
        }
    }
}
```

Main

```
C:\Users\Raluca\.jdks\corretto-22.0.2\bin\java.exe "-javaagent:C:\Users\Raluca\Desktop\IntelliJ IDEA 2024.3.2.2\lib\idea_rt.jar=57805:C:\Users\Raluca\De
Tesla Model Y (400 HP)
Toyota Camry (250 HP)

Process finished with exit code 0
```

## Program Features & Screenshots

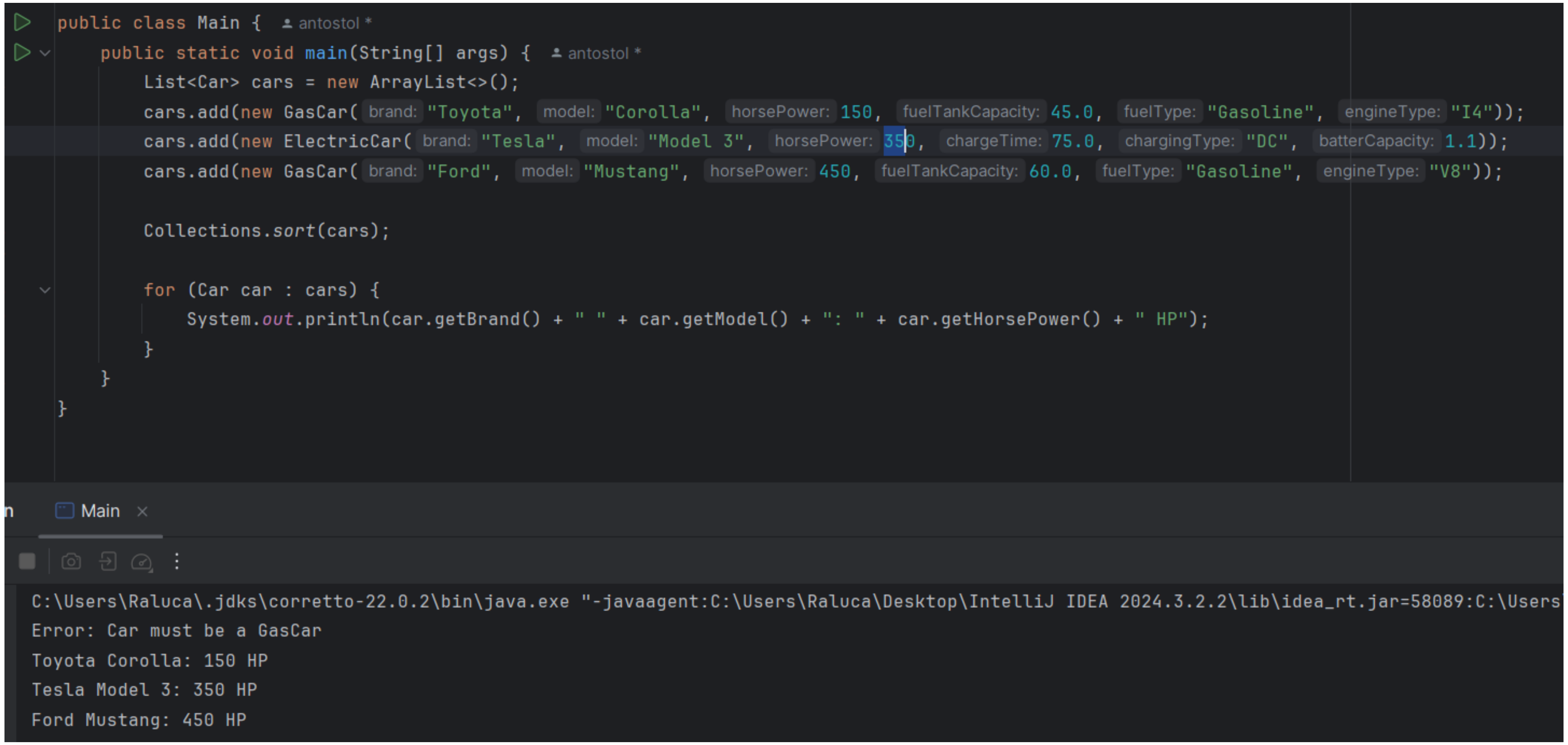
- Interface:
  - Searchable => findByModel(), findByBrand()

## Program Features & Screenshots

- Comparable & Comparator:
  - Car implements Comparable<Car> (horsePower)
  - Comparator classes are used to compare:
    - ElectricCar (chargingTime/batteryCapacity)
    - GasCar (fuelTankCapacity)
    - OnlineDealer (rating)
    - InPersonDealer (numberOfEmployees)
    - Dealership (first by inventory size, then by name)

# Program Features & Screenshots

- Comparable & Comparator: (Output & Execution examples)

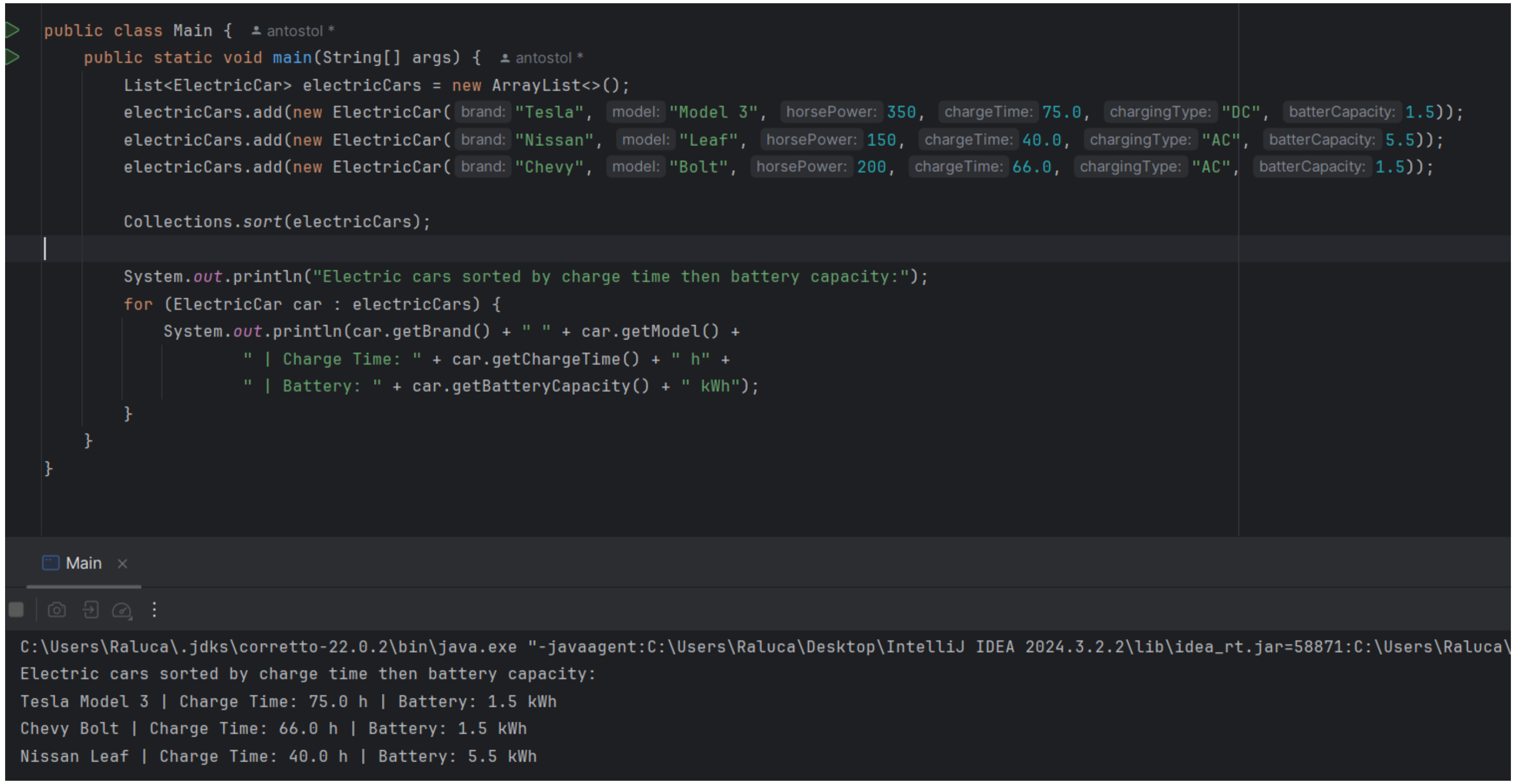


```
public class Main {  antostol *
    public static void main(String[] args) {  antostol *
        List<Car> cars = new ArrayList<>();
        cars.add(new GasCar( brand: "Toyota",  model: "Corolla",  horsePower: 150,  fuelTankCapacity: 45.0,  fuelType: "Gasoline",  engineType: "I4"));
        cars.add(new ElectricCar( brand: "Tesla",  model: "Model 3",  horsePower: 350,  chargeTime: 75.0,  chargingType: "DC",  batterCapacity: 1.1));
        cars.add(new GasCar( brand: "Ford",  model: "Mustang",  horsePower: 450,  fuelTankCapacity: 60.0,  fuelType: "Gasoline",  engineType: "V8"));

        Collections.sort(cars);

        for (Car car : cars) {
            System.out.println(car.getBrand() + " " + car.getModel() + ": " + car.getHorsePower() + " HP");
        }
    }
}
```

```
C:\Users\Raluca\.jdks\corretto-22.0.2\bin\java.exe "-javaagent:C:\Users\Raluca\Desktop\IntelliJ IDEA 2024.3.2.2\lib\idea_rt.jar=58089:C:\Users
Error: Car must be a GasCar
Toyota Corolla: 150 HP
Tesla Model 3: 350 HP
Ford Mustang: 450 HP
```

# Program Features & Screenshots

- Comparable & Comparator: (Output & Execution examples)



```java
public class Main {  antostol *
    public static void main(String[] args) {  antostol *
        List<ElectricCar> electricCars = new ArrayList<>();
        electricCars.add(new ElectricCar( brand: "Tesla",  model: "Model 3",  horsePower: 350,  chargeTime: 75.0,  chargingType: "DC",  batterCapacity: 1.5));
        electricCars.add(new ElectricCar( brand: "Nissan",  model: "Leaf",  horsePower: 150,  chargeTime: 40.0,  chargingType: "AC",  batterCapacity: 5.5));
        electricCars.add(new ElectricCar( brand: "Chevy",  model: "Bolt",  horsePower: 200,  chargeTime: 66.0,  chargingType: "AC",  batterCapacity: 1.5));

        Collections.sort(electricCars);

        System.out.println("Electric cars sorted by charge time then battery capacity:");
        for (ElectricCar car : electricCars) {
            System.out.println(car.getBrand() + " " + car.getModel() +
                    " | Charge Time: " + car.getChargeTime() + " h" +
                    " | Battery: " + car.getBatteryCapacity() + " kWh");
        }
    }
}
```

Main ×

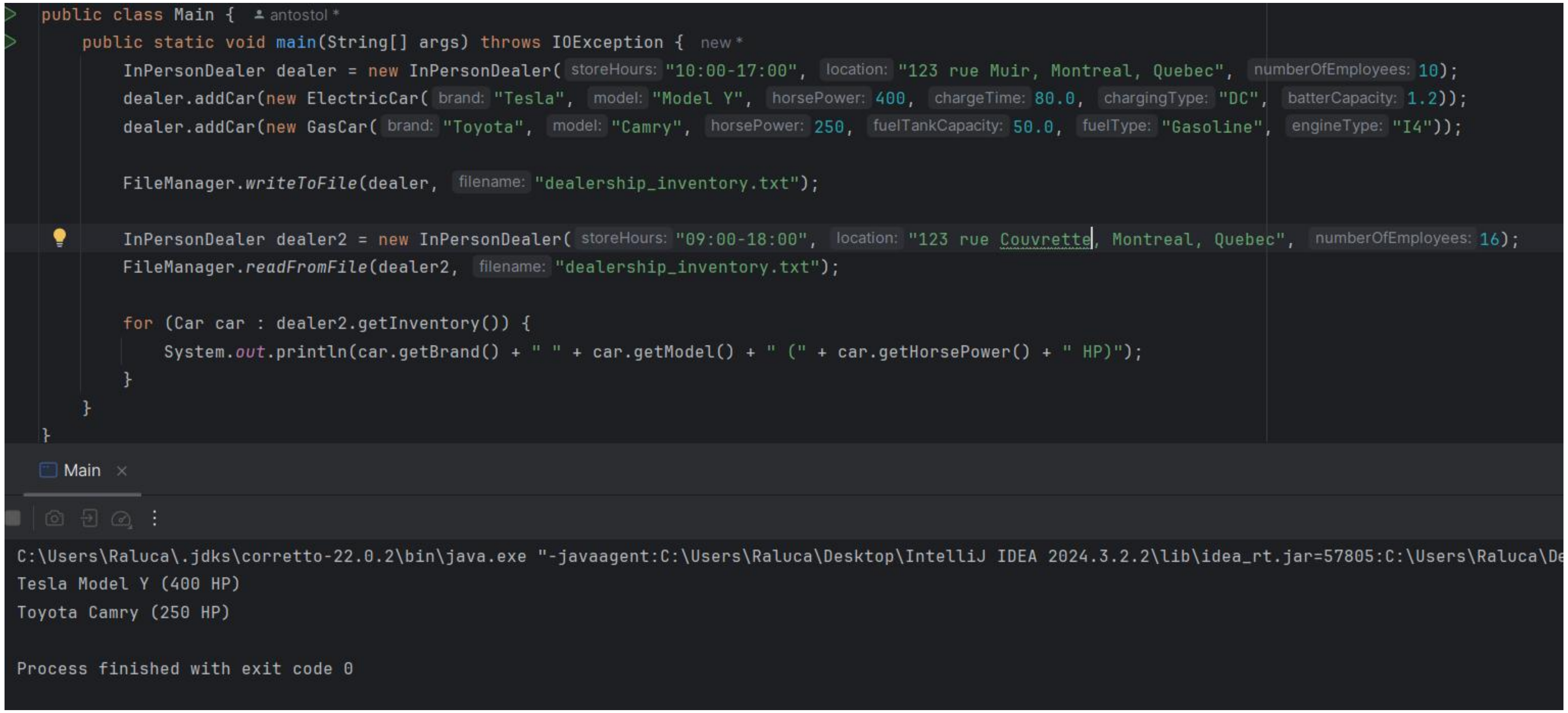
```
C:\Users\Raluca\.jdks\corretto-22.0.2\bin\java.exe "-javaagent:C:\Users\Raluca\Desktop\IntelliJ IDEA 2024.3.2.2\lib\idea_rt.jar=58871:C:\Users\Raluca\
Electric cars sorted by charge time then battery capacity:
Tesla Model 3 | Charge Time: 75.0 h | Battery: 1.5 kWh
Chevy Bolt | Charge Time: 66.0 h | Battery: 1.5 kWh
Nissan Leaf | Charge Time: 40.0 h | Battery: 5.5 kWh
```

# Program Features & Screenshots

- Text IO:
  - FileManager class => contains writeToFile() (store info of inventory) & readFromFile() (load info of inventory) methods

# Program Features & Screenshots

- Text IO (Output & Execution examples):



```
public class Main {  antostol *
    public static void main(String[] args) throws IOException {  new *
        InPersonDealer dealer = new InPersonDealer( storeHours: "10:00-17:00",  location: "123 rue Muir, Montreal, Quebec",  numberOfEmployees: 10);
        dealer.addCar(new ElectricCar( brand: "Tesla",  model: "Model Y",  horsePower: 400,  chargeTime: 80.0,  chargingType: "DC",  batterCapacity: 1.2));
        dealer.addCar(new GasCar( brand: "Toyota",  model: "Camry",  horsePower: 250,  fuelTankCapacity: 50.0,  fuelType: "Gasoline",  engineType: "I4"));

        FileManager.writeToFile(dealer,  filename: "dealership_inventory.txt");

        InPersonDealer dealer2 = new InPersonDealer( storeHours: "09:00-18:00",  location: "123 rue Couvrette, Montreal, Quebec",  numberOfEmployees: 16);
        FileManager.readFromFile(dealer2,  filename: "dealership_inventory.txt");

        for (Car car : dealer2.getInventory()) {
            System.out.println(car.getBrand() + " " + car.getModel() + " (" + car.getHorsePower() + " HP)");
        }
    }
}
```

Main

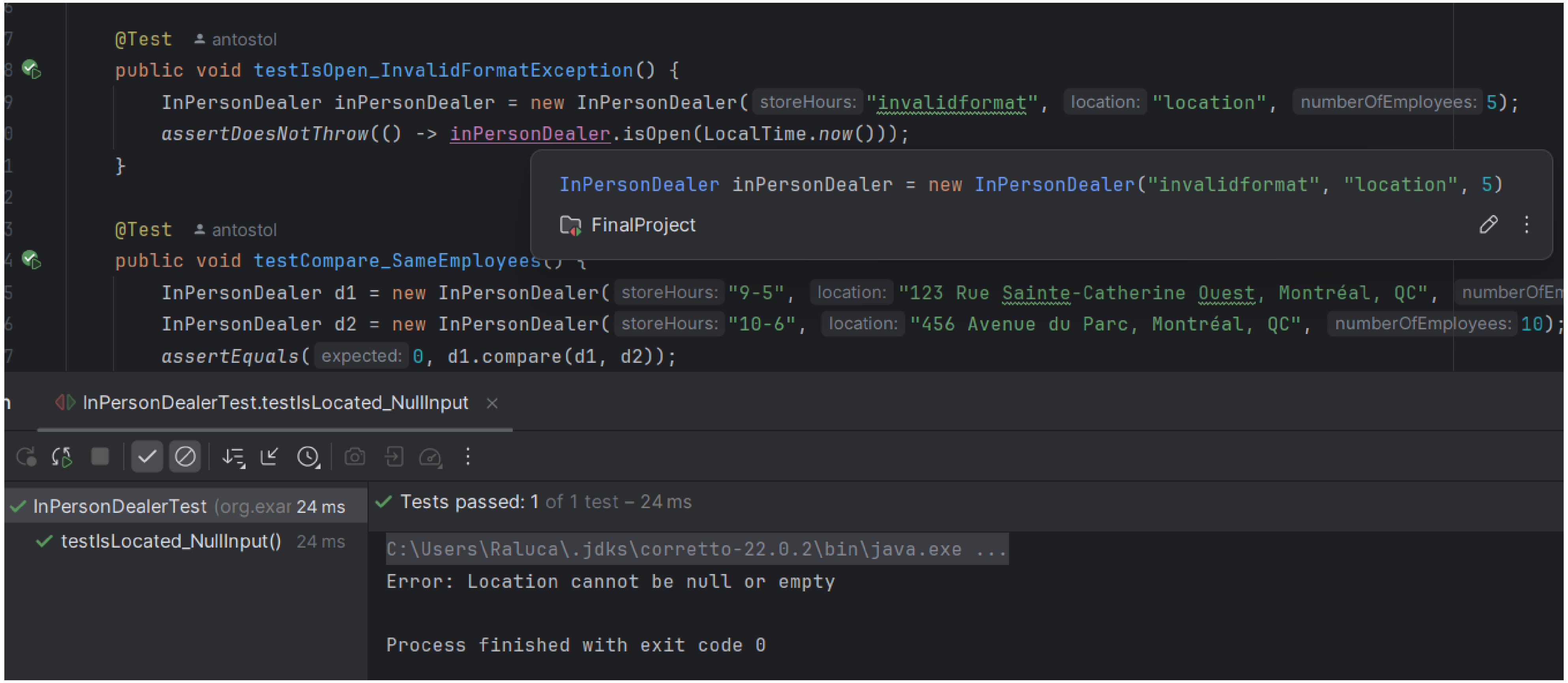
```
C:\Users\Raluca\.jdks\corretto-22.0.2\bin\java.exe "-javaagent:C:\Users\Raluca\Desktop\IntelliJ IDEA 2024.3.2.2\lib\idea_rt.jar=57805:C:\Users\Raluca\De
Tesla Model Y (400 HP)
Toyota Camry (250 HP)

Process finished with exit code 0
```

## Program Features & Screenshots

- Exception handling
  - Almost all methods (besides generic ones) have exception handling
    - IOException, ArrayOutOfBoundException, IllegalArgumentException, IllegalState Exception, Exception, etc.

# Program Features & Screenshots

- Exception handling (Output & Execution examples):



```
@Test  antostol
public void testIsOpen_InvalidFormatException() {
    InPersonDealer inPersonDealer = new InPersonDealer( storeHours: "invalidformat",  location: "location",  numberOfEmployees: 5);
    assertDoesNotThrow(() -> inPersonDealer.isOpen(LocalTime.now()));
}

@Test  antostol
public void testCompare_SameEmployees() {
    InPersonDealer d1 = new InPersonDealer( storeHours: "9-5",  location: "123 Rue Sainte-Catherine Ouest, Montréal, QC",  numberOfEm
    InPersonDealer d2 = new InPersonDealer( storeHours: "10-6",  location: "456 Avenue du Parc, Montréal, QC",  numberOfEmployees: 10);
    assertEquals( expected: 0, d1.compare(d1, d2));
```

```
InPersonDealer inPersonDealer = new InPersonDealer("invalidformat", "location", 5)
FinalProject
```

InPersonDealerTest.testIsLocated_NullInput

✓ InPersonDealerTest (org.exar 24 ms
✓ testIsLocated_NullInput() 24 ms

✓ Tests passed: 1 of 1 test – 24 ms

```
C:\Users\Raluca\.jdks\corretto-22.0.2\bin\java.exe ...
Error: Location cannot be null or empty

Process finished with exit code 0
```

## Program Features & Screenshots

- Stream processing & Lambda expressions/method referencing
  - Filtering cars by model/brand with filterByBrand() & filterByModel()

## Program Features & Screenshots

- Stream processing & Lambda expressions/method referencing (Output & Execution examples):
- Here I was supposed to show you how the output of filtering cars would look, but my program crashed…

## Program Features & Screenshots

- Junit & testCases
  - Each user-defined method has at least 4 test cases to test various scenarios

## Program Features & Screenshots

- Documentation:
    - Every method has documentation – test cases do not, simply because the information about them is written in their name

## Challenges

- The biggest struggle in my case was time management. With a better approach at it, I’d be able to make sure that every single test case run through correctly. I’d also be able to make sure that everything is formatted even more neatly, as well as more efficiently. Not to mention that I’d be able to make modifications after running the test cases.

## Learning Outcomes

- From this project, I learned how amazing programming can be. At first, it seemed like a completely different language from the ones I speak, but once I got used to it and saw how it worked, I realized that it’s like any other language. And I have to admit, I do need more practice (PRACTICE MAKES PERFECT), but from whatever I have learned now, I find it truly amazing what people have come to create. I also learned there are many different shortcuts in IntelliJ, AND INTELLIJ itself (thank you, sir, for introducing us to this program), to make my life easier. I also learned some new things not included in your Syllabus, such as the LocalTime object, different syntaxes with the new Java, etc. And many, many, many other things that I can’t think of right now…

## THANK YOU!!! Enjoy your summer sir!