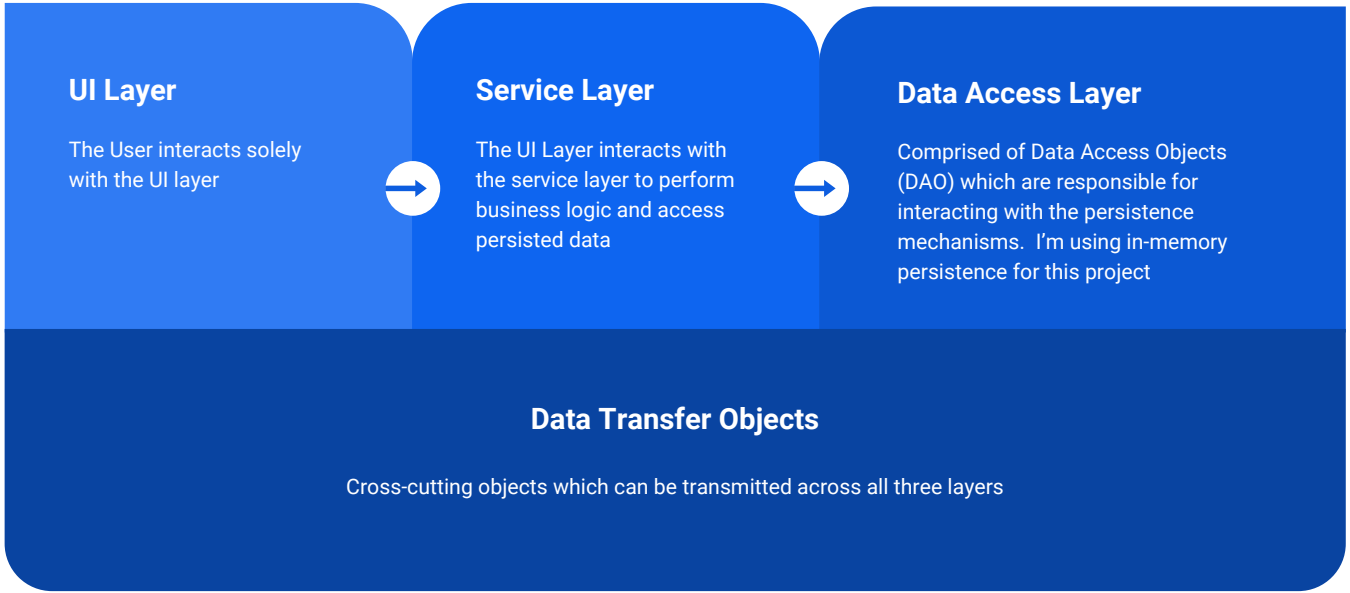
CS 665 Final Project

Car Dealership
12/06/18
Jimmy Goddard

Use Cases

- Car dealership software model which allows the user to
  - See what cars are available
  - Purchase cars
  - Customize cars
  - Schedule test drives

Multi-tier Architecture



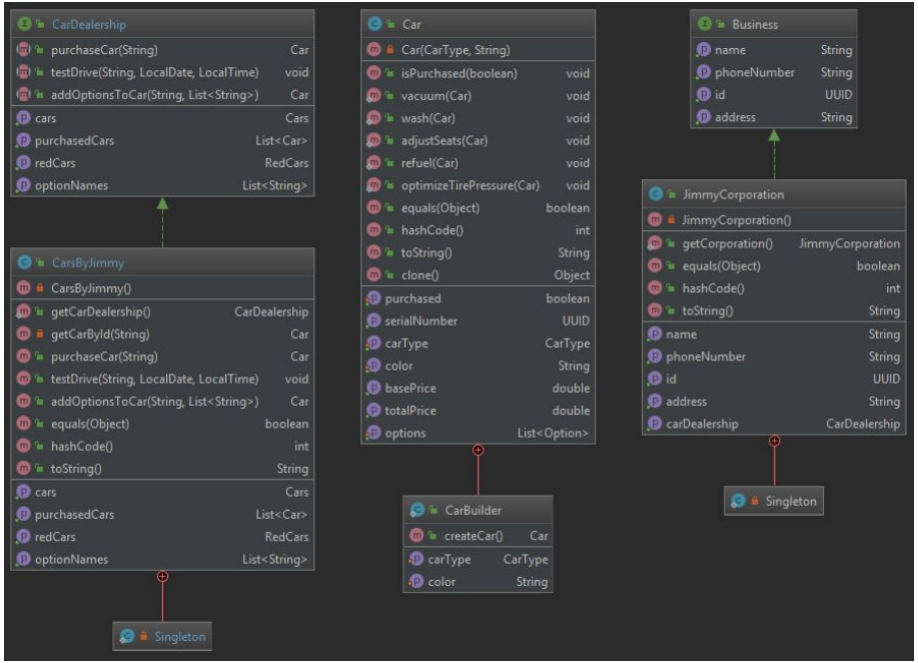
Structure

- A Corporation, Jimmy Corporation, which has a Car Dealership
- The Car Dealership, Cars By Jimmy, has a Garage of Cars
- Garage contains iterable collections of Cars

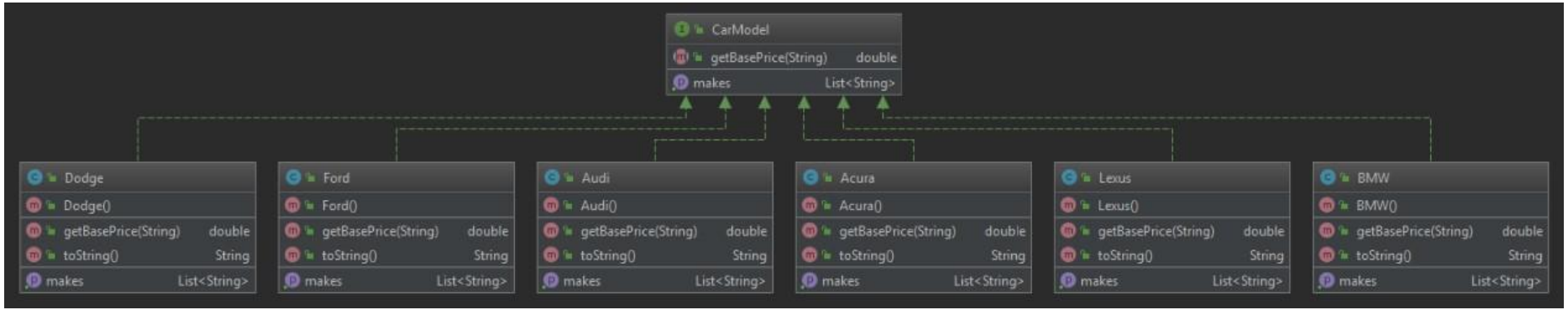
Structure

- Cars have Models and Makes which are encapsulated in a CarType class which enforces the proper combinations of models and makes
- Cars also have Options which can be added and do affect the price of the car

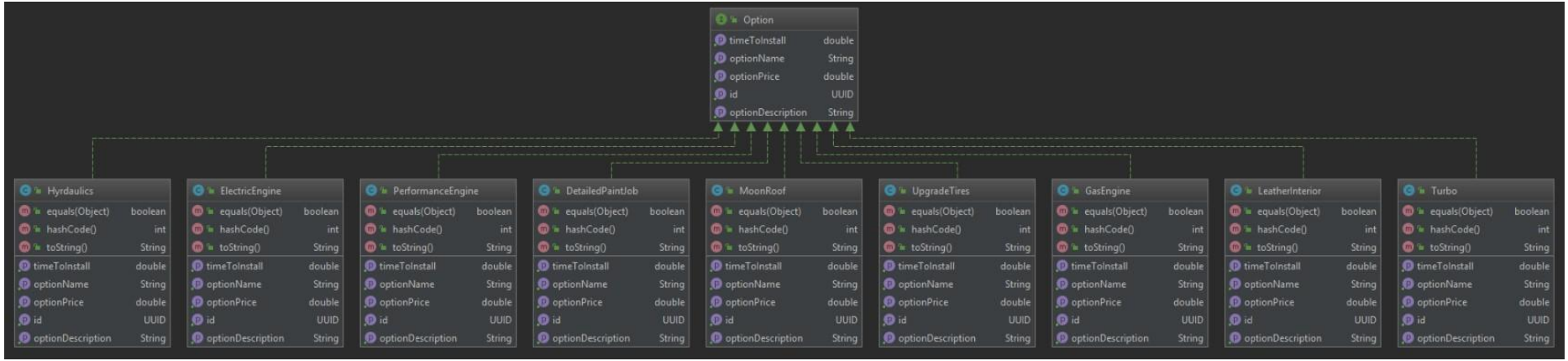
Important Classes



Car Models


CarModel
getBasePrice(String) double
makes List<String>
Dodge
Dodge()
getBasePrice(String) double
toString() String
makes List<String>
Ford
Ford()
getBasePrice(String) double
toString() String
makes List<String>
Audi
Audi()
getBasePrice(String) double
toString() String
makes List<String>
Acura
Acura()
getBasePrice(String) double
toString() String
makes List<String>
Lexus
Lexus()
getBasePrice(String) double
toString() String
makes List<String>
BMW
BMW()
getBasePrice(String) double
toString() String
makes List<String>

Car Options


Option
timeToInstall double
optionName String
optionPrice double
id UUID
optionDescription String
Hyrdaulics
ElectricEngine
PerformanceEngine
DetailedPaintJob
MoonRoof
UpgradeTires
GasEngine
LeatherInterior
Turbo
equals(Object) boolean
hashCode() int
toString() String
timeToInstall double
optionName String
optionPrice double
id UUID
optionDescription String

Design Patterns

- Singleton
- Builder
- Prototype
- Iterator
- Facade
- Mediator

Singleton

```
10  public class JimmyCorporation implements Business {
11
12    private final UUID id = UUID.randomUUID();
13    private final DataFactory dataFactory = new DataFactory();
14    private final String address = dataFactory.getAddress();
15    private final String phoneNumber = dataFactory.getNumberText(10);
16    private final CarDealership carDealership = CarsByJimmy.getCarDealership();
17
18    private JimmyCorporation() {}
19
20    private static class Singleton {
21      private static final JimmyCorporation instance = new JimmyCorporation();
22    }
23
24    public static JimmyCorporation getCorporation() {
25      return Singleton.instance;
26    }
27
```

```
7   private static final CarDealership carDealership =
8       JimmyCorporation.getCorporation().getCarDealership();
9   private static final CarDealershipMenu carDealershipMenu = new CarDealershipMenu(carDealership);
10
```

Builder

```
140   public static class CarBuilder {
141
142     private CarType carType;
143     private String color = "black";
144
145     public CarBuilder setCarType(final CarType carType) {
146       this.carType = carType;
147       return this;
148     }
149
150     public CarBuilder setColor(final String color) {
151       this.color = color;
152       return this;
153     }
154
155     public Car createCar() {
156       return new Car(carType, color);
157     }
158   }
159 }
160
```

```
31   private final void setUpCarStore() {
32     acuras =
33         new Acura()
34             .getMakes()
35             .stream()
36             .map(make -> new CarType(new Acura(), make))
37             .map(
38                 carType ->
39                     new Car.CarBuilder()
40                         .setCarType(carType)
41                         .setColor(colors.get(random.nextInt(colors.size())))
42                         .createCar())
43             .collect(Collectors.toList());
44
```

Prototype

```
11  public class Car implements Cloneable {
12
13    @Override
14    public Object clone() throws CloneNotSupportedException {
15      final Car oldCar = (Car) super.clone();
16      return new Car.CarBuilder()
17          .setCarType((CarType) oldCar.getCarType().clone())
18          .setColor(oldCar.getColor())
19          .createCar();
20    }
21
```

```
117   private Car getRandomCar() throws CloneNotSupportedException {
118     final List<Car> cars = allCars.get(random.nextInt(allCars.size()));
119     return (Car) cars.get(random.nextInt(cars.size())).clone();
120   }
121
```

Iterator

```java
public class Cars implements Iterable<Car> {

  private final List<Car> cars;

  public Cars(final List<Car> cars) {
    this.cars = cars;
  }

  public boolean isEmpty() {
    return cars == null || cars.isEmpty();
  }

  public List<Car> toList() {
    return cars;
  }

  @Override
  public Iterator<Car> iterator() {
    return new CarIterator();
  }

  private class CarIterator implements Iterator<Car> {

    private final int size = cars.size();
    private int currentPosition = 0;

    @Override
    public boolean hasNext() {
      return currentPosition < size;
    }

    @Override
    public Car next() {
      if (!hasNext()) {
        throw new NoSuchElementException();
      }
      return cars.get(currentPosition++);
    }
  }
}

```

```java
public class RedCars implements Iterable<Car> {

  private final List<Car> cars;

  public RedCars(final List<Car> cars) {
    if (isEmpty(cars)) {
      this.cars = Collections.emptyList();
      return;
    }
    this.cars =
        cars.stream()
            .filter(car -> car.getColor().equalsIgnoreCase("red"))
            .collect(Collectors.toList());
  }

  public boolean isEmpty() {
    return isEmpty(cars);
  }

  private static boolean isEmpty(final List<Car> cars) {
    return cars == null || cars.isEmpty();
  }

  @Override
  public Iterator<Car> iterator() {
    return new CarIterator();
  }

  private class CarIterator implements Iterator<Car> {

    private final int size = cars.size();
    private int currentPosition = 0;

    @Override
    public boolean hasNext() {
      return currentPosition < size;
    }

    @Override
    public Car next() {
      if (!hasNext()) {
        throw new NoSuchElementException();
      }
      return cars.get(currentPosition++);
    }
  }
}

```

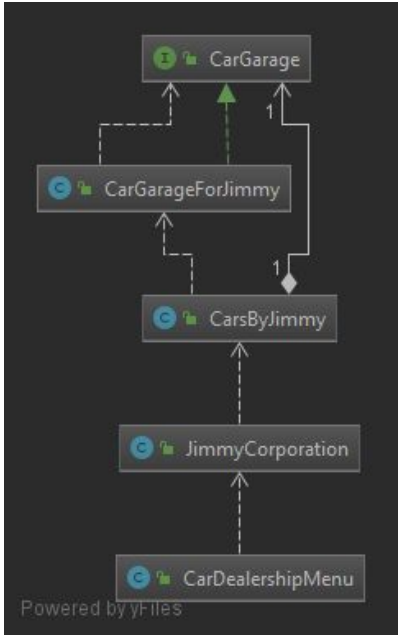
Iterator Usage

```
56          () -> {
57            final RedCars redCars = carDealership.getRedCars();
58            if (redCars.isEmpty()) {
59              System.out.println("No cars to list.  Please create a list of cars");
60              return;
61            }
62            final Iterator<Car> carIterator = redCars.iterator();
63            while (carIterator.hasNext()) {
64              final Car redCar = carIterator.next();
65              System.out.println(redCar);
66            }
67          });
68
```

Facade

```
23    public void beginTestDrive() throws InvalidTestDriveException {
24      if (this.car == null) {
25        throw new InvalidTestDriveException("There must be a car to test drive");
26      }
27      Car.vacuum(car);
28      Car.wash(car);
29      Car.adjustSeats(car);
30      Car.refuel(car);
31      Car.optimizeTirePressure(car);
32    }
33
```

Mediator


CarGarage
1
CarGarageForJimmy
1
CarsByJimmy
JimmyCorporation
CarDealershipMenu
Powered by yFiles