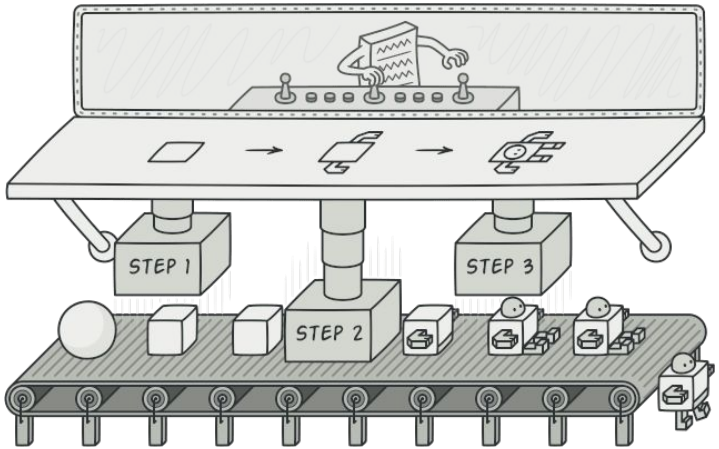
CarBuilder

Eihab Syed (100707448) & Preet Panchal (100707094)
Ontario Tech University
CSCI 2020U - R. Weagant
February 24, 2022

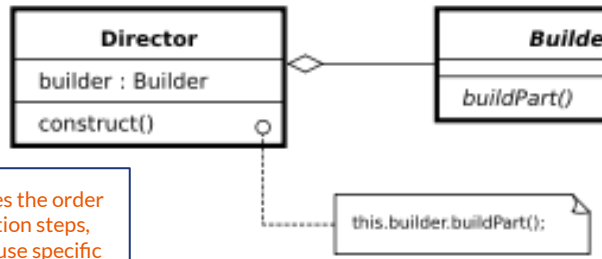
The Builder Pattern

What is the Builder Pattern?

- Helps solve object creation problems in OOP
- Uses a step-by-step approach
- Separates the construction of a complex object from its representation
- Allows you to produce different types and representations of an object using the same construction code



The Builder Pattern Structure



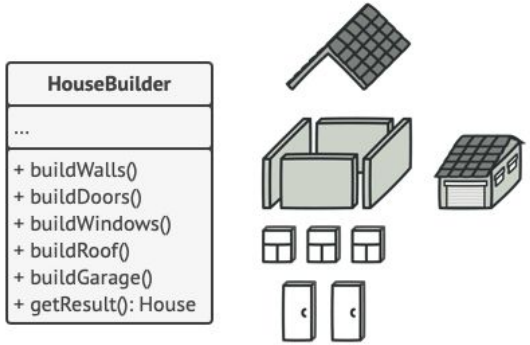
The Builder interface declares product construction steps that are common to all types of builders.

Concrete Builders provide different implementations of the construction steps. Concrete builders may produce products that don't follow the common interface.

The Director class defines the order in which to call construction steps, so you can create and reuse specific configurations of products.

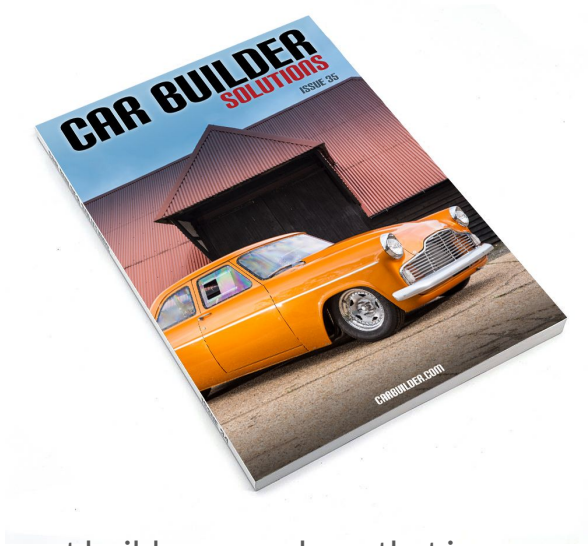
Products are resulting objects. Products constructed by different builders don't have to belong to the same class hierarchy or interface.

Examples of the Builder Pattern



- Building a house which requires four walls and a floor, install a door, fit a pair of windows, and a roof
  - Extend the “base” House class and create a set of subclasses to cover all combinations of the parameters.
- A fast-food restaurant where you have to build a typical meal with specific burgers and drinks
  - Create an Item interface representing food items such as burgers and cold drinks and concrete classes implementing the Item interface and a Packing interface representing packaging of food items.

Our Implementation



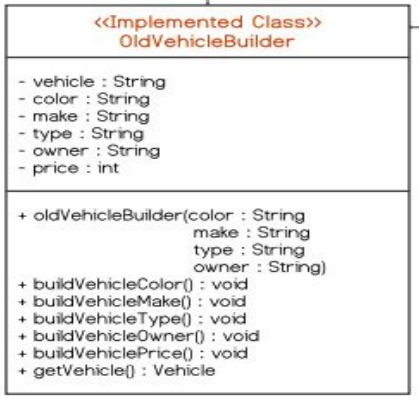
Project:
CarBuilder

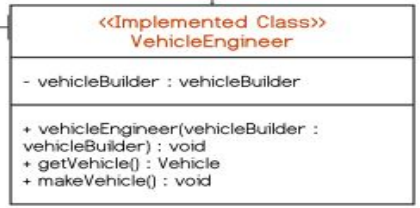
Problem:
If a client is looking to buy a car, to help them find their ideal car, we must build a car package that is tailored to our client. We know there are different car types (ie. Sedan, SUV, Minivan), but what if the client wants a specific make or colour for their car?

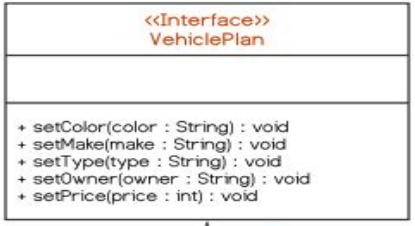
CarBuilder UML Class Diagram

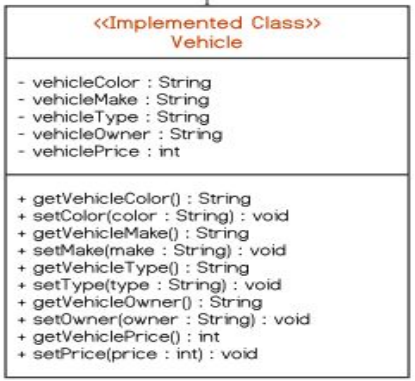
CarBuilder:
Builder Design Pattern Structure

Client: "Build my car!"

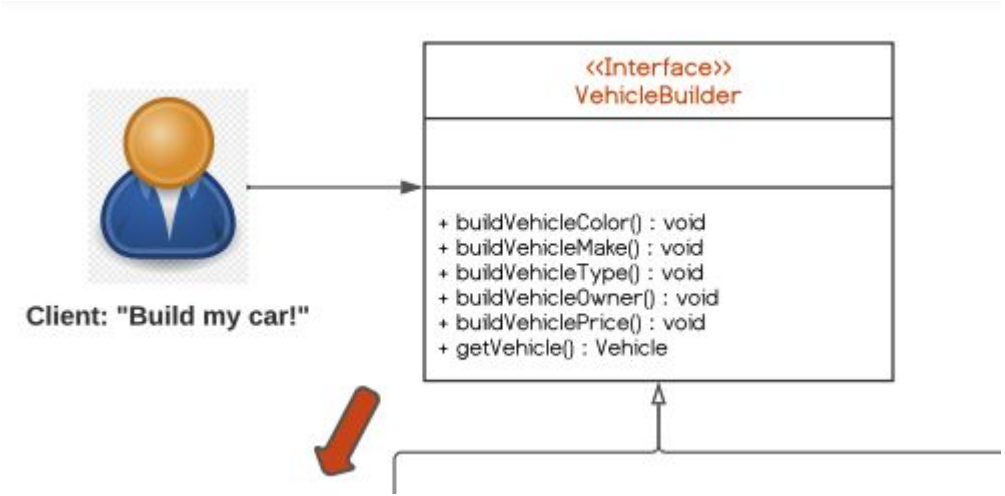






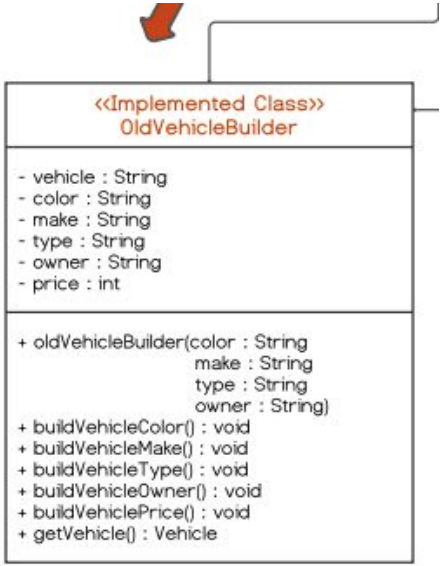


The Builder:




```java
public interface VehicleBuilder {

    // builds vehicle color
    public void buildVehicleColor();

    // builds vehicle make
    public void buildVehicleMake();

    // builds vehicle type
    public void buildVehicleType();

    // builds vehicle owner
    public void buildVehicleOwner();

    // builds vehicle price
    public void buildVehiclePrice();

    // retrieves newly built vehicle
    public Vehicle getVehicle();
}
```

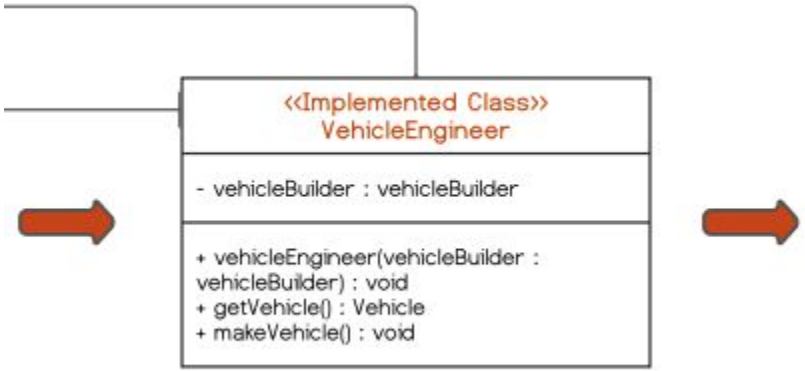
Concrete Builders:




```java
// vehicle object constructor
public OldVehicleBuilder(String color, String make, String type, String owner) {
    this.vehicle = new Vehicle();
    this.color = color;
    this.make = make;
    this.type = type;
    this.owner = owner;

    // Check vehicle type and make to change price accordingly
    if (Objects.equals(type, b: "Sedan")) {
        if (Objects.equals(make, b: "Toyota")){
            price += 20000;
        } else if (Objects.equals(make, b: "Honda")) {
            price += 18000;
        } else {
            price += 15000;
        }
    }
    else if (Objects.equals(type, b: "SUV")) {
        if (Objects.equals(make, b: "Toyota")){
            price += 28000;
        } else if (Objects.equals(make, b: "Honda")) {
            price += 26000;
        } else {
            price += 24000;
        }
    } else {
        if (Objects.equals(make, b: "Toyota")){
            price += 36000;
        } else if (Objects.equals(make, b: "Honda")) {
            price += 34000;
        } else {
            price += 31000;
        }
    }
}
```

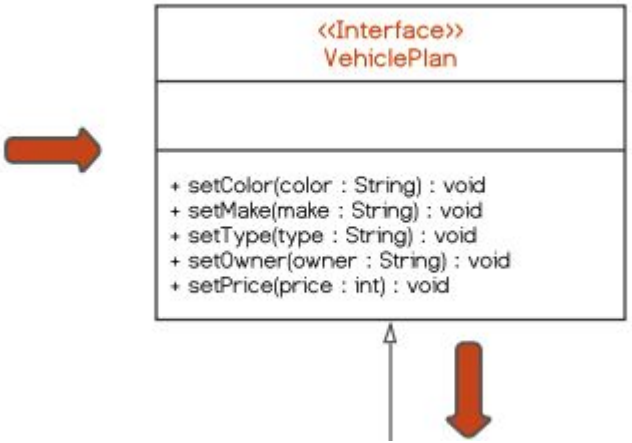
```java
// builds vehicle color
@Override
public void buildVehicleColor() { vehicle.setColor(color); }

// builds vehicle make
@Override
public void buildVehicleMake() { vehicle.setMake(make); }

// builds vehicle type
@Override
public void buildVehicleType() { vehicle.setType(type); }

// builds vehicle owner
@Override
public void buildVehicleOwner() { vehicle.setOwner(owner); }

// builds vehicle price
@Override
public void buildVehiclePrice() { vehicle.setPrice(price); }

// gets newly built vehicle
public Vehicle getVehicle() { return this.vehicle; }
```

The Director:



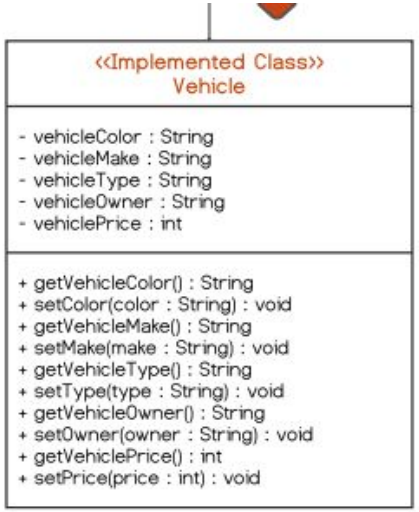

```java
public class VehicleEngineer {

    private VehicleBuilder vehicleBuilder;

    public VehicleEngineer(VehicleBuilder vehicleBuilder) {
        this.vehicleBuilder = vehicleBuilder;
    }

    // gets newly built vehicle
    public Vehicle getVehicle() { return this.vehicleBuilder.getVehicle(); }

    // builds vehicle
    public void makeVehicle() {
        this.vehicleBuilder.buildVehicleColor();
        this.vehicleBuilder.buildVehicleMake();
        this.vehicleBuilder.buildVehicleType();
        this.vehicleBuilder.buildVehicleOwner();
        this.vehicleBuilder.buildVehiclePrice();
    }

}
```

Secondary Builder:




```java
public interface VehiclePlan {

    // sets vehicle color
    public void setColor(String color);

    // sets vehicle make
    public void setMake(String make);

    // sets vehicle type
    public void setType(String type);

    // sets vehicle owner
    public void setOwner(String owner);

    // sets vehicle price
    public void setPrice(int price);

}
```

The Product:




```java
public class Vehicle implements VehiclePlan{

    private String vehicleColor;
    private String vehicleMake;
    private String vehicleType;
    private String vehicleOwner;
    private int vehiclePrice;

    // sets vehicle color
    @Override
    public void setColor(String color) { vehicleColor = color; }

    // gets vehicle color
    public String getVehicleColor() { return vehicleColor; }

    // sets vehicle make
    @Override
    public void setMake(String make) { vehicleMake = make; }

    // gets vehicle make
    public String getVehicleMake() { return vehicleMake; }

    // sets vehicle type
    @Override
    public void setType(String type) { vehicleType = type; }

    // gets vehicle type
    public String getVehicleType() { return vehicleType; }

    // sets vehicle owner
    @Override
    public void setOwner(String owner) { vehicleOwner = owner; }

    // gets vehicle owner
    public String getVehicleOwner() { return vehicleOwner; }
```

Sample Application Run:

References

Builder design pattern. GeeksforGeeks. (2021, October 18). Retrieved February 24, 2022, from https://www.geeksforgeeks.org/builder-design-pattern/

Builder design pattern. HowToDoInJava. (2022, January 2). Retrieved February 24, 2022, from https://howtodoinjava.com/design-patterns/creational/builder-pattern-in-java/

Builder. Refactoring.Guru. (n.d.). Retrieved February 24, 2022, from https://refactoring.guru/design-patterns/builder

Design patterns - builder pattern. (n.d.). Retrieved February 24, 2022, from https://www.tutorialspoint.com/design_pattern/builder_pattern.htm

Poyias, A. (2019, February 8). Design patterns - a quick guide to builder pattern. Medium. Retrieved February 24, 2022, from https://medium.com/@andreaspoyias/design-patterns-a-quick-guide-to-builder-pattern-a834d7cacead