Summer Semester 2015

Software Engineering Design & Construction

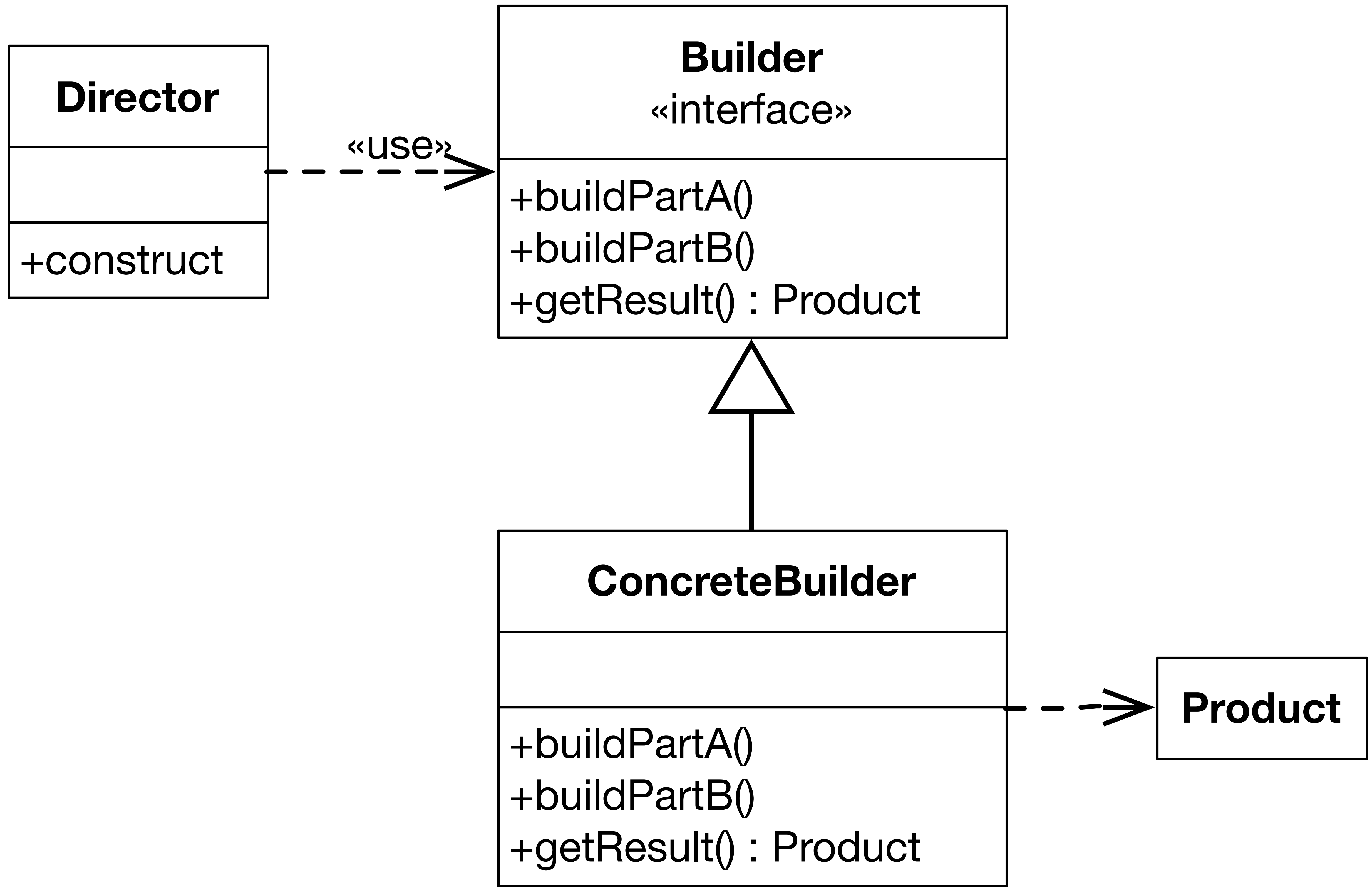
Dr. Michael Eichberg
Fachgebiet Softwaretechnik
Technische Universität Darmstadt

Builder Pattern

The Builder Pattern

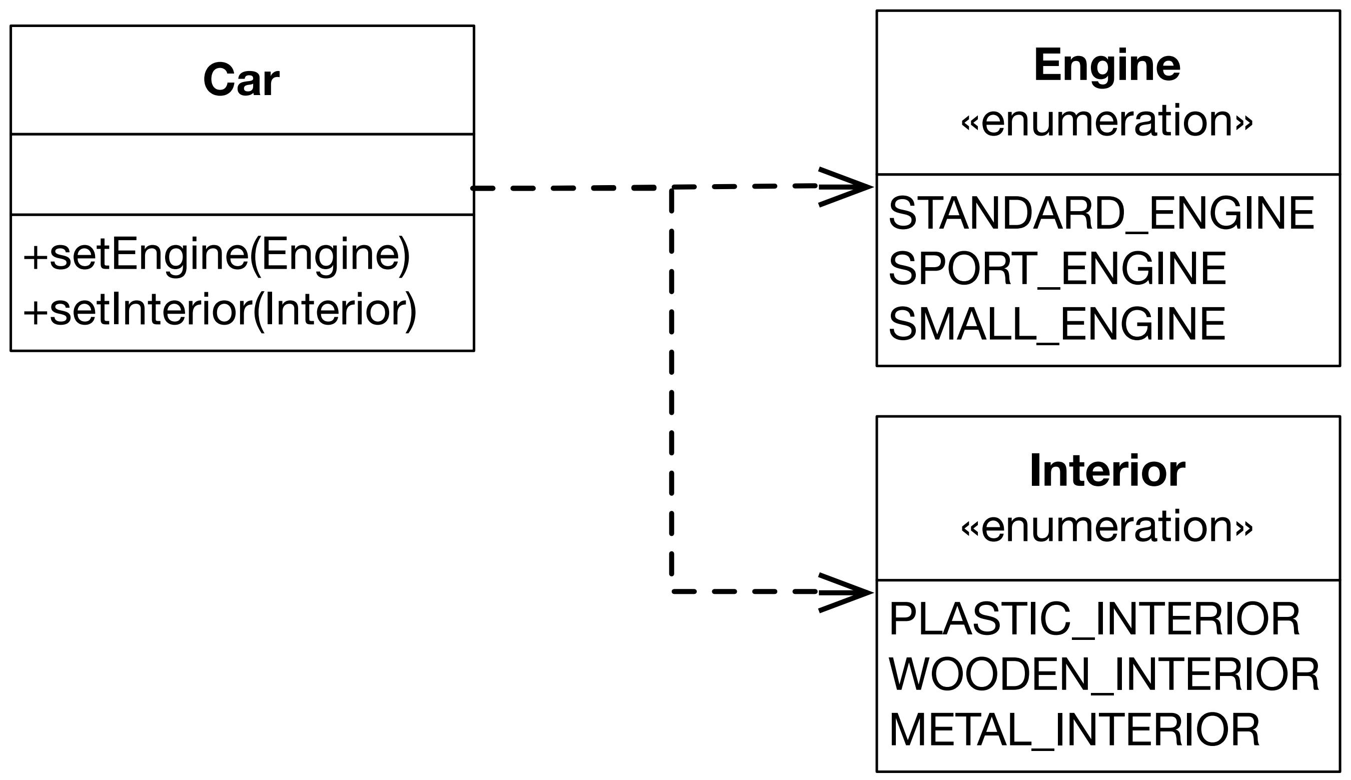
Divide the construction of multi-part objects in different steps, so that different implementations of these steps can construct different representations of object

Builder - Structure

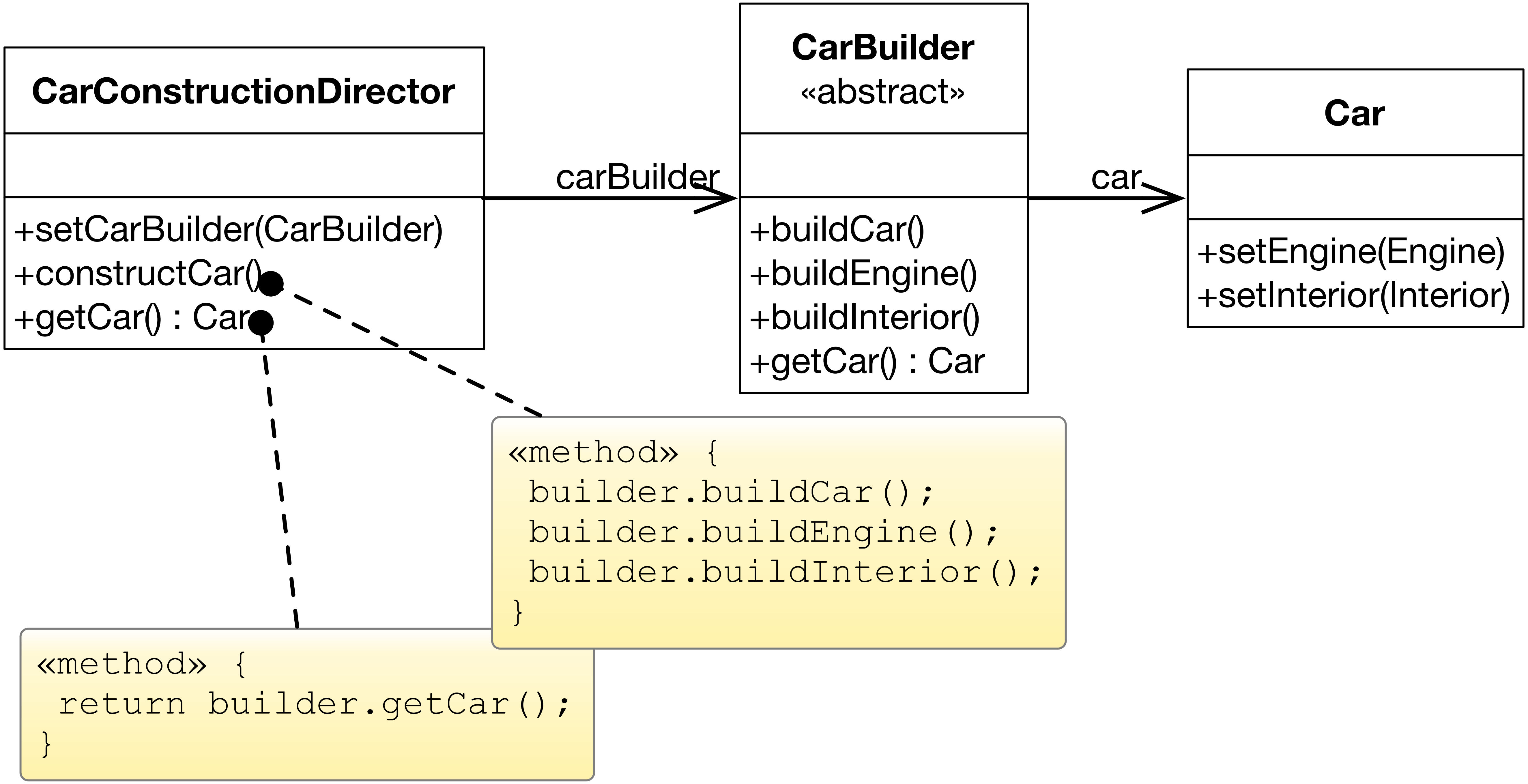


Builder - A Car Builder

- We want to construct different types of cars.
- In this example, cars have an engine and an interior.



Builder - A Car Builder



```
«method» {
 builder.buildCar();
 builder.buildEngine();
 builder.buildInterior();
}
```

```
«method» {
 return builder.getCar();
}
```

Two Possible Car Builders

```
class CheapCarBuilder extends CarBuilder {
  void buildEngine() {
    car.setEngine(Engine.SMALL_ENGINE);
  }

  void buildInterior() {
    car.setInterior(Interior.PLASTIC_INTERIOR);
  }
}

class LuxuryCarBuilder extends CarBuilder {

  void buildEngine() {
    car.setEngine(Engine.SPORT_ENGINE);
  }

  void buildInterior() {
    car.setInterior(Interior.WOODEN_INTERIOR);
  }
}
```

Takeaway

- Use Abstract Factory for creating objects depending on finite numbers of factors you know in advance.
  E.g. if there are only three kinds of cars.

- Use Builder for creating complex objects depending on unbound number of factors that are decided at runtime.
  E.g. if cars can be configured with multiple different parts.