

Design Patterns

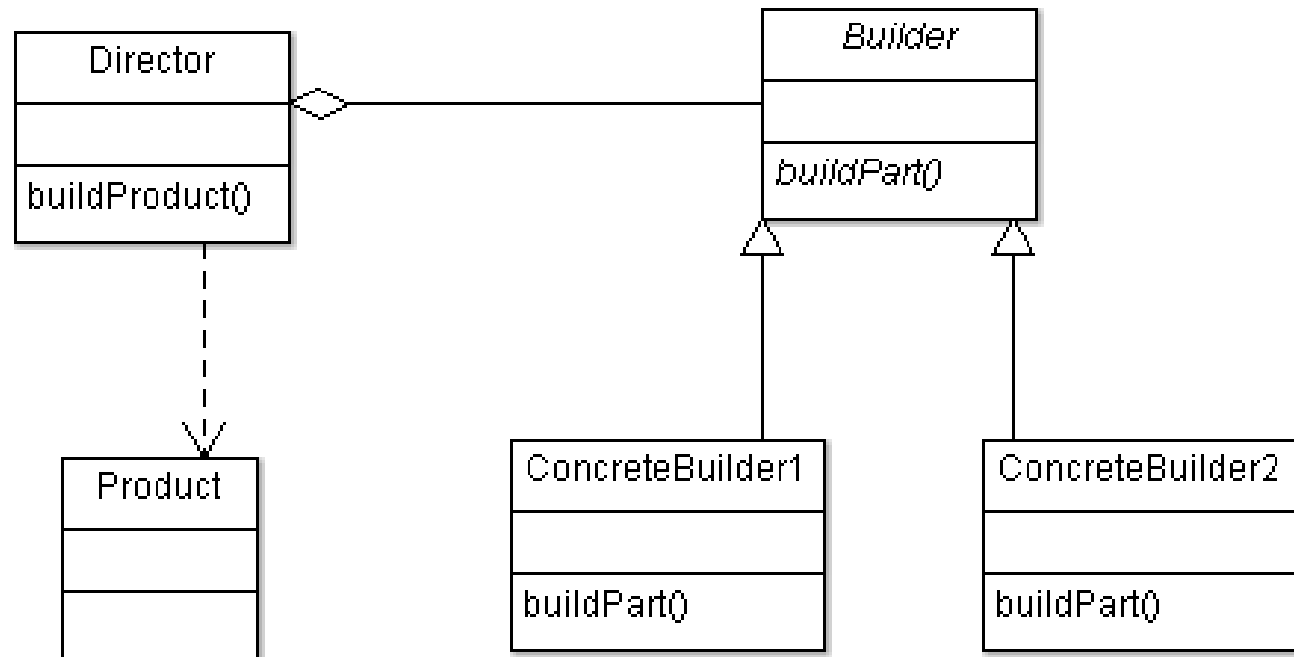
Creational patterns: Builder

Dr. Chad Williams
Central Connecticut State University

Design pattern: Builder

- **Category:** Creational design pattern
- **Intent:**
 - Separate construction of complex objects so same construction process can create complex object from different implementation parts
- **Applicability:**
 - When process for creating object should be independent of parts that make up the object
 - When construction process should allow various implementations of the parts used for construction

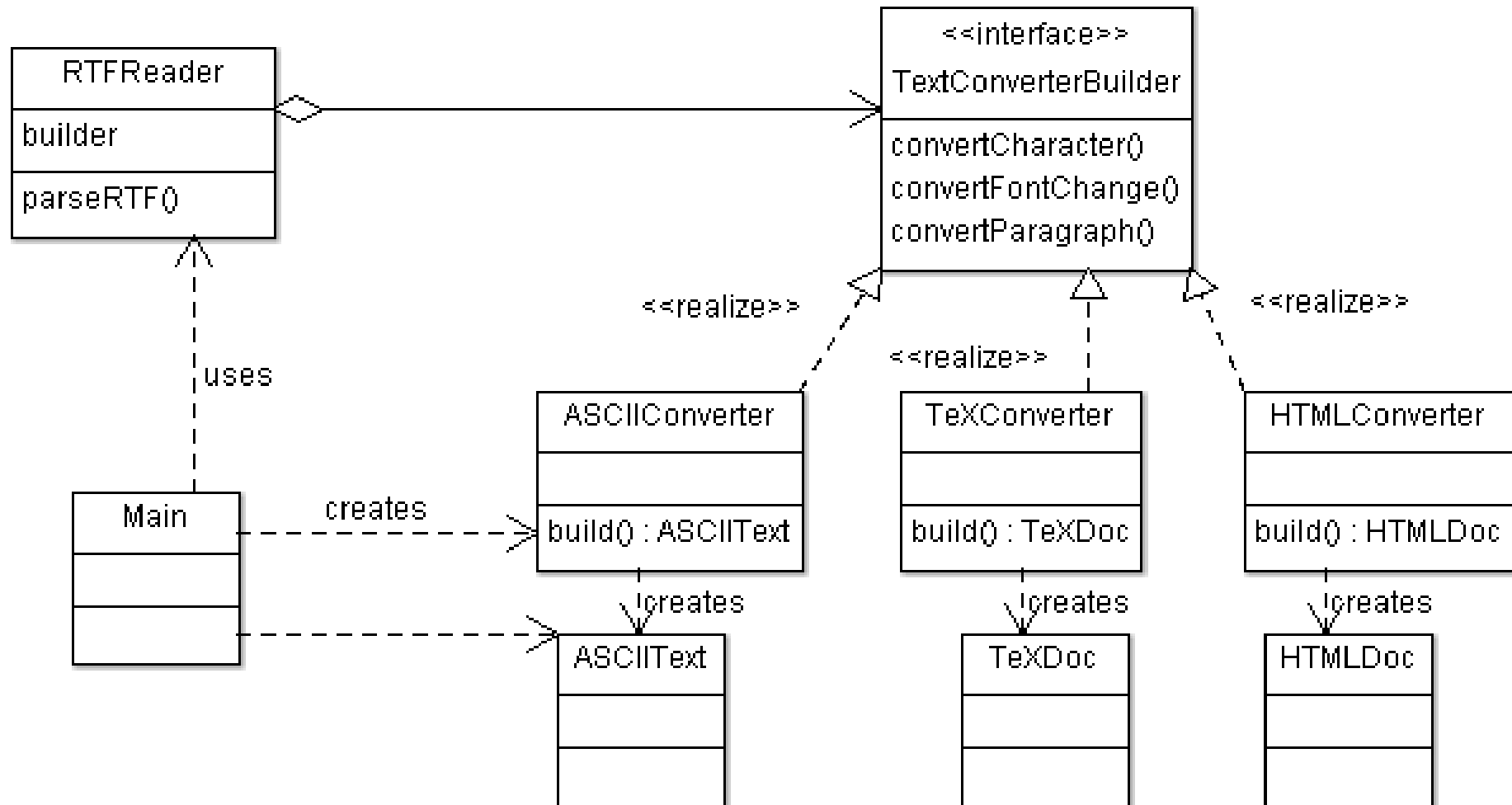
Builder UML



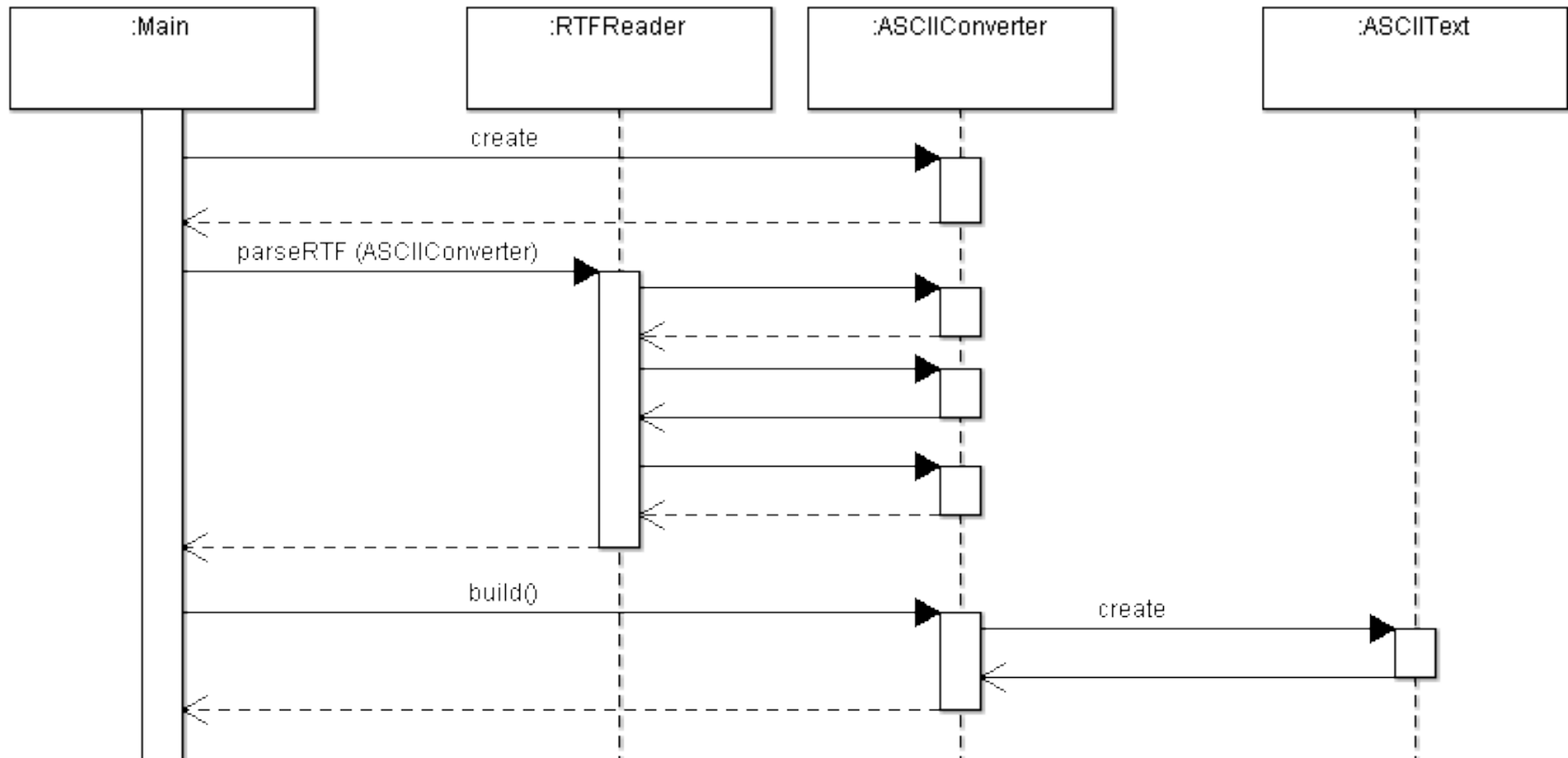
Builder roles

- **Builder** – Defines interface for creating parts of Product object
- **ConcreteBuilder** – Constructs and assembles parts of the product by implementing Builder interface
- **Director** – Constructs a Product using the Builder interface
- **Product** – complex object under construction

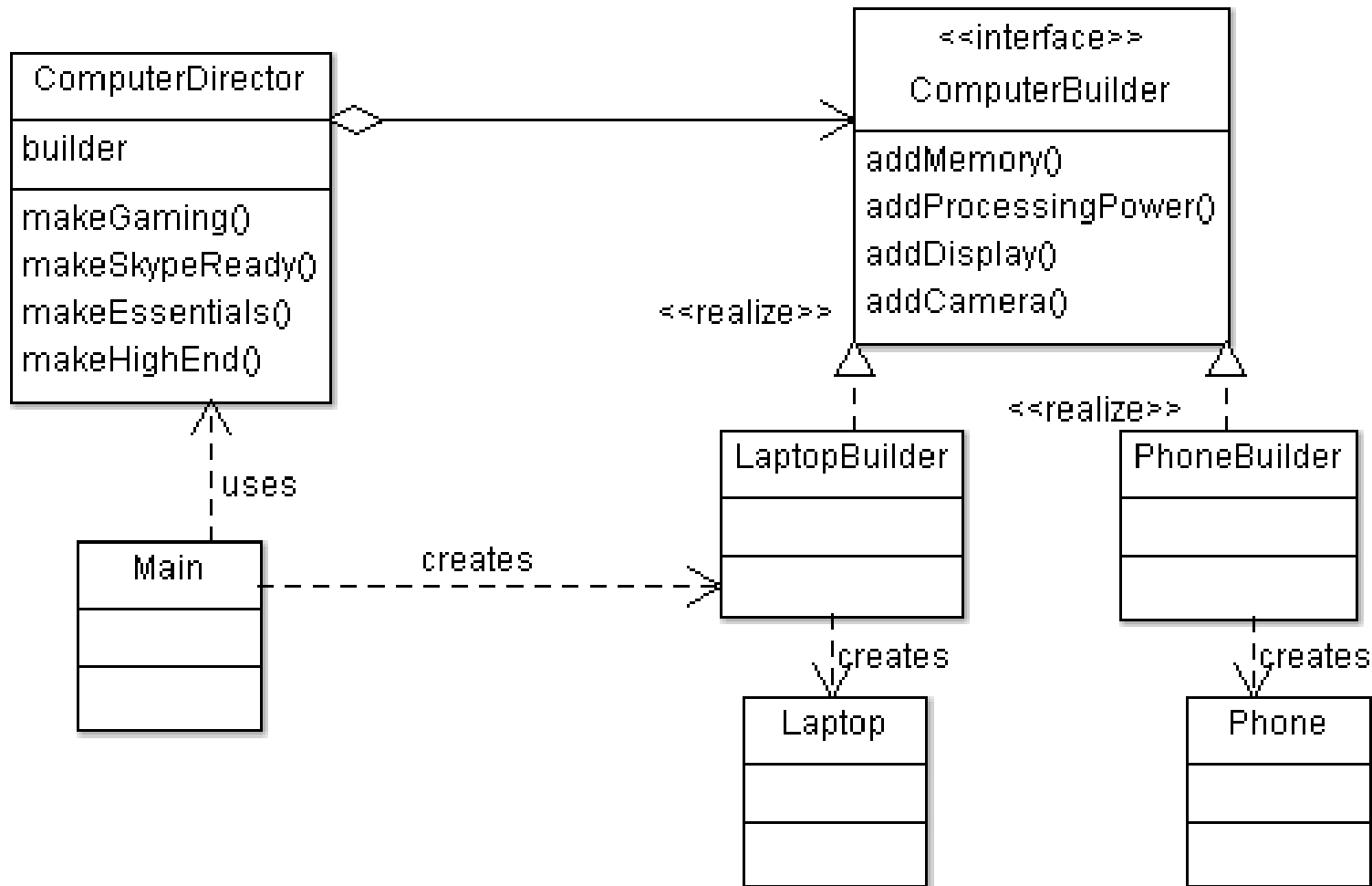
Document converter



Builder sequence



Computer/phone builder

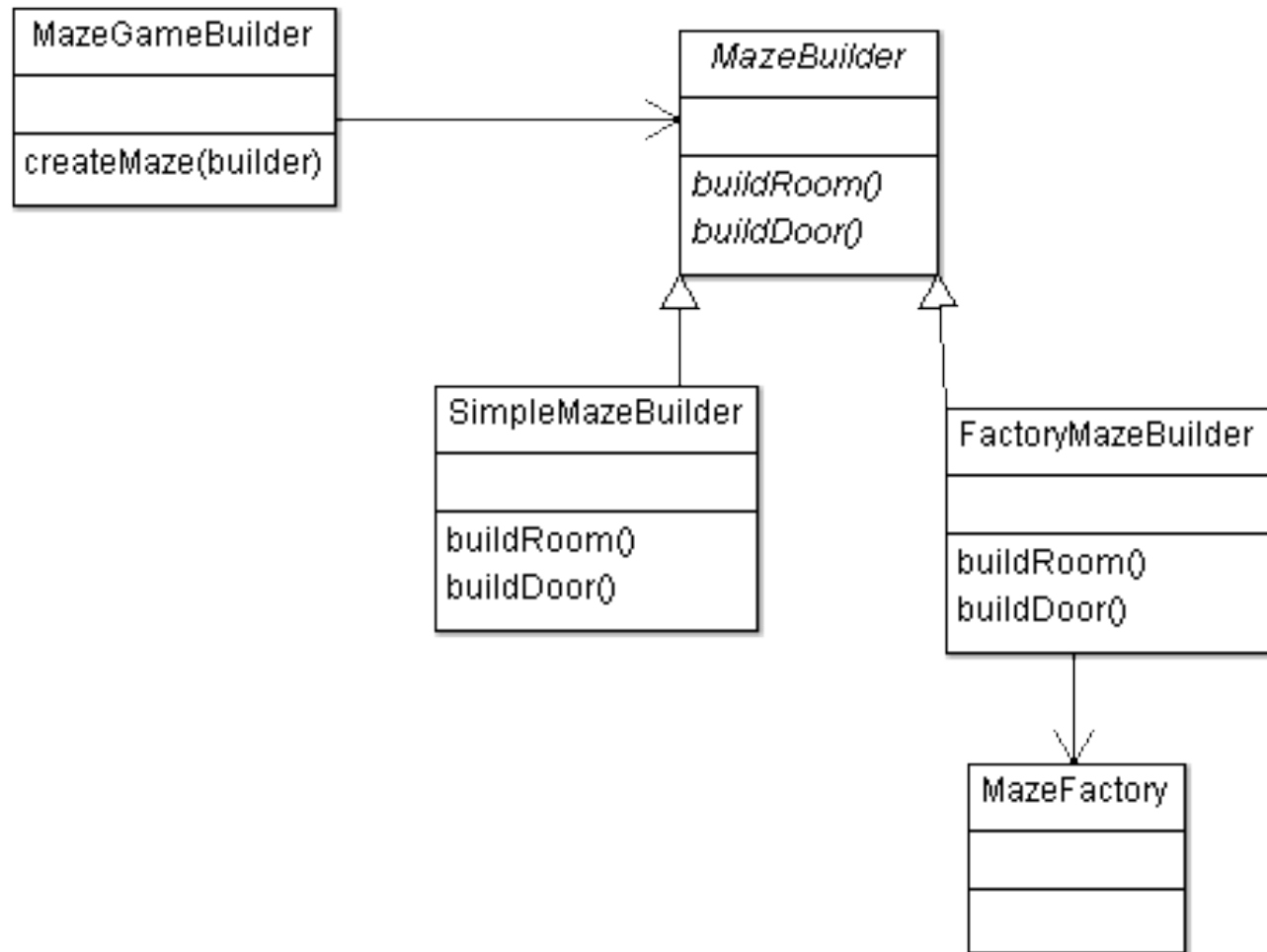


Builder pattern

- Pattern used when constructing objects is complex and repetitive
- Example for maze:

```
Room room1 = factory.makeRoom(1);
Room room2 = factory.makeRoom(2);
Door door1 = factory.makeDoor(room1, room2);
room1.setSide(Direction.NORTH, door1);
room1.setWall(Direction.EAST,
    factory.makeWall());
room1.setWall(Direction.WEST,
    factory.makeWall());
room1.setWall(Direction.SOUTH,
    factory.makeWall());
room2.setSide(Direction.SOUTH, door1);
...
```


Maze builder



Builder pattern evolution

- **Reminder:**
- **Intent**
 - Separate construction of complex objects so same construction process can create complex object from different implementation parts

Pattern has evolved to have additional flavors since original Gang of 4 vision

- Main change accomplish same intent, but less complexity needed for many implementations
- Easy to convert to original model should complexity change to justify it

Two primary new flavors

- Construction of objects with a lot of variety of construction mechanisms - uses similar pattern but for different goals (usually) – (what I will refer to as “simple builder” but not an official term)
- Construction of complex components with many aggregates that can change - fully realizes Gang of 4 goals (what I will refer to as “complex construction builder” but not an official term)
- Both generally implemented with nested static class as the builder with private constructor for object

“Simple builder”

- Intent: For objects that have many different ways of being constructed and potentially dependencies in construction
 - 20 possible attributes, numerous combinations of valid initial state, but “empty” instance would violate idea of valid object
- General form
 - Single class plus potentially enumerations
 - Nested static builder with private outer class constructor
 - Flow – Create builder, populate builder, build

Oddities of coding for convenience

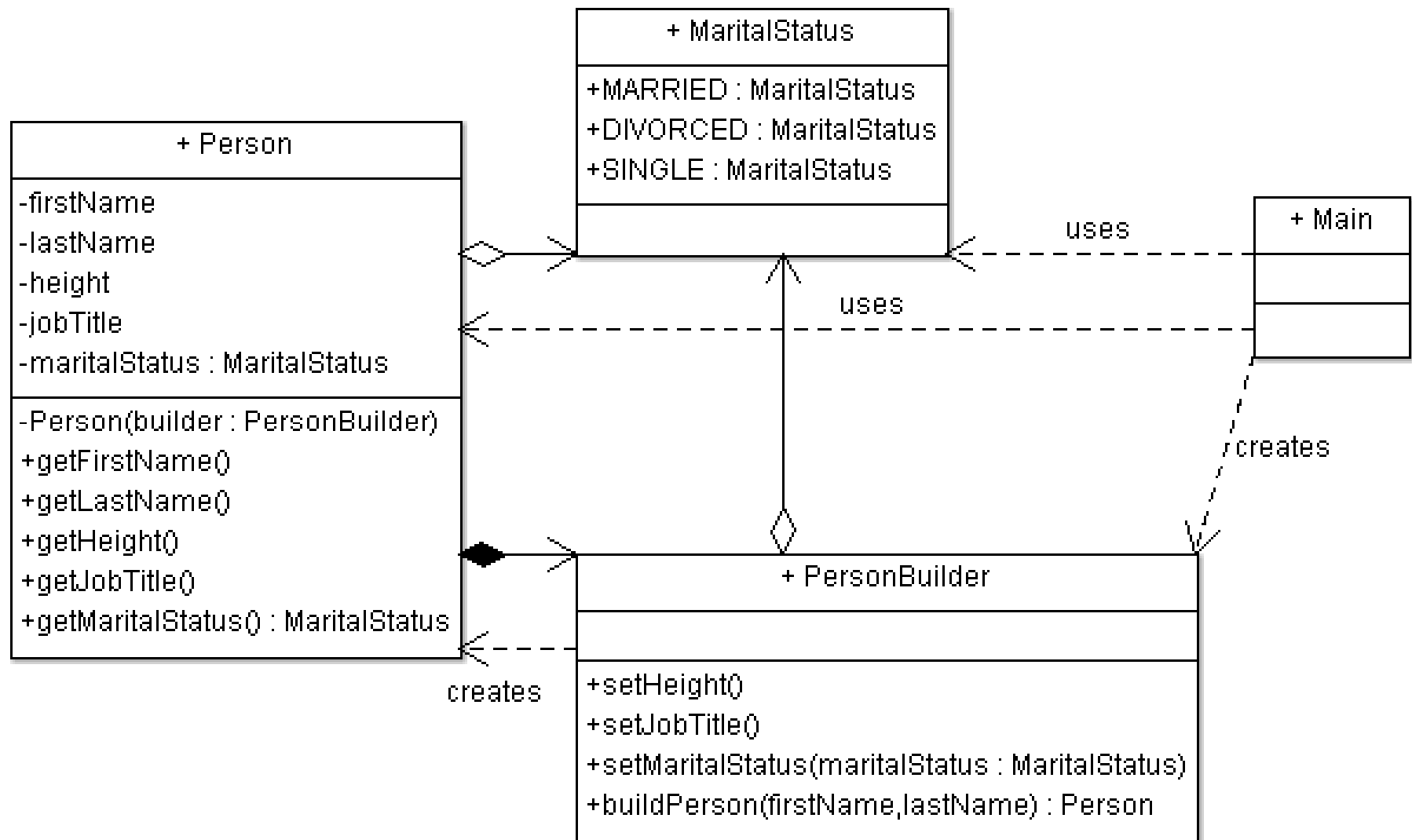
Within nested builder common to return an instance of that builder with each call:

```
public PersonBuilder age(int passedAge) {  
    this.nestedAge = passedAge;  
    return this;  
}
```

Why?

```
Person myPerson = new Person.PersonBuilder("John", "Doe")  
    .age(30)  
    .phone("1234567")  
    .address("Fake address 1234")  
    .build();  
}
```

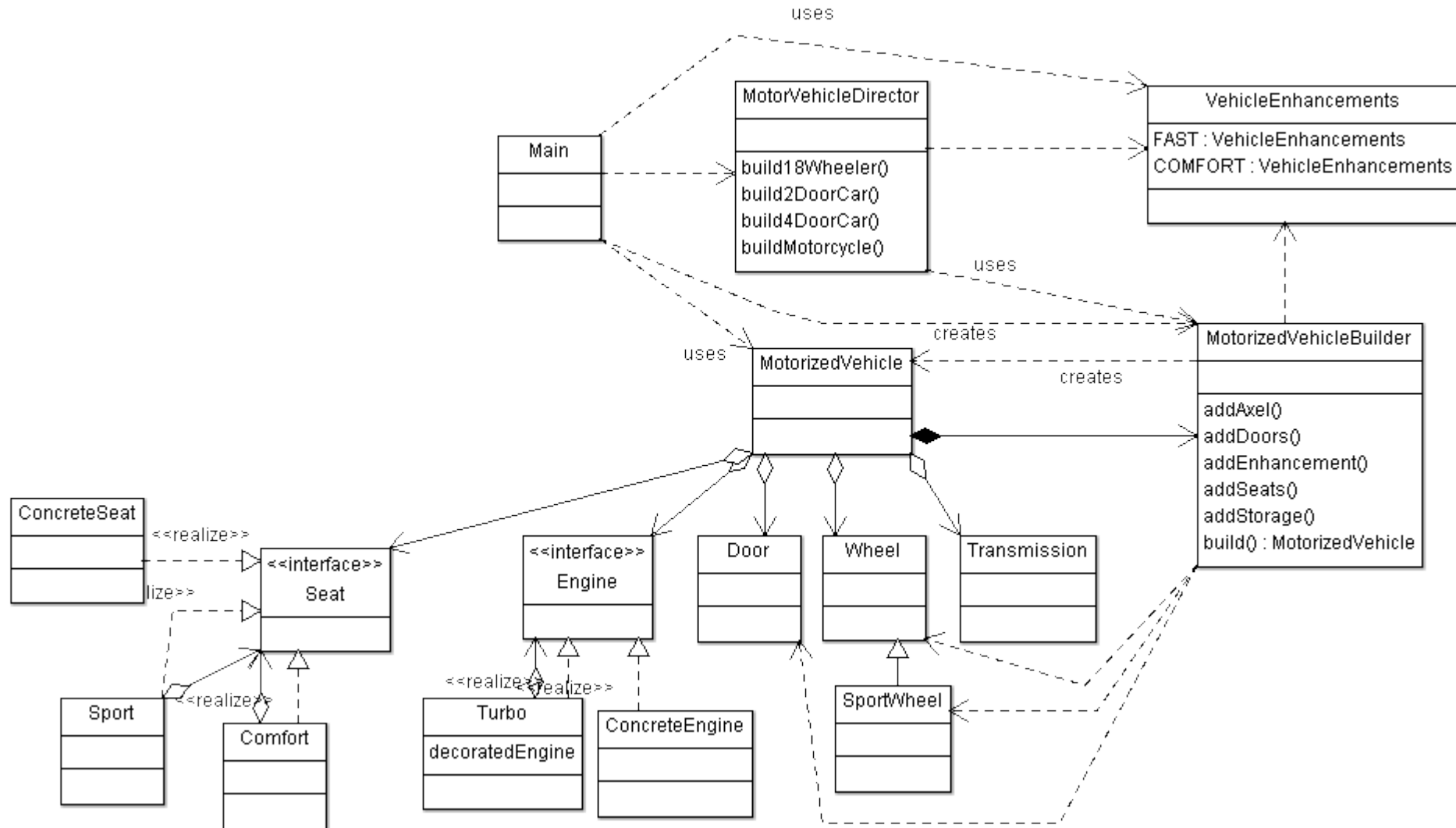
Simple builder version



“Complex construction builder”

- Intent: For objects that have many different parts, dependencies, and ways of being constructed; separate construction of complex object from complexities of internal representation.
 - External object asks to build concept – director and builder hide complexities that make up internal construction and structure
- General form
 - Class with several aggregate classes that can be identified from a general concept
 - Nested static builder with private outer class constructor
 - Director that encapsulates complex or repetitive build operations
 - Flow – Create builder, call director passing builder, call build on builder

“Complex construction builder”



Group work

- Describe in detail how you could use either the Gang of 4 builder pattern or the “Complex construction builder” pattern within your project