CS 417
Design Patterns

Creational patterns: Builder

Dr. Chad Williams
Central Connecticut State University

Announcements

- Final projects **pushing due date Monday 12/7** (3weeks)
  - Presentation slides (includes select UML patterns presented) **at beginning of class**
  - All other final document/code versions due at 11:59

- Presentation:
  - Describe overall concept of simulation/game
  - Included patterns where included and what about that situation made the pattern appropriate
    - Explain UML for a selection of them
  - Application demo (your config then class picks the config)

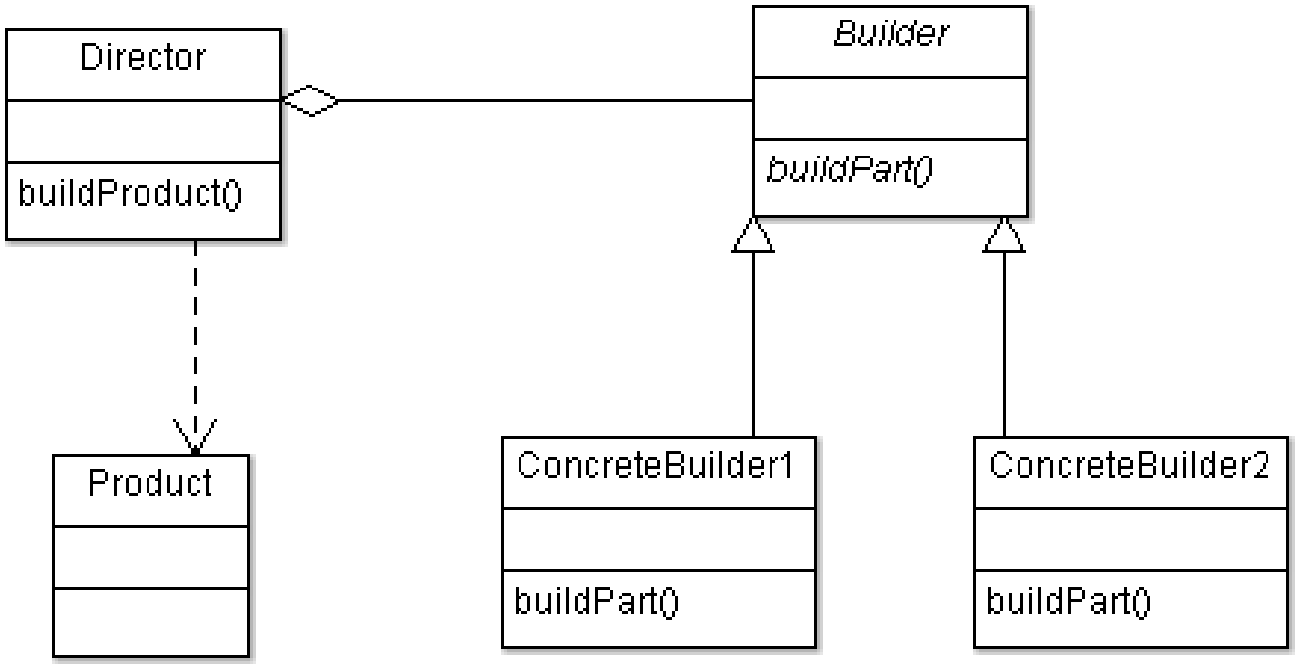
More details

- Presentation order will be first come first serve. You are due to present when you slot comes up whether that ends up being Mon 12/7 or Wed 12/9 so be prepared.
- Sign up will be posted soon. If you don't sign up for a slot you will be given first empty one.
- Not being ready to present Monday (if called) 10% off regardless of if code turned in by end of day, in this case code can be submitted until Tuesday 11:59pm same 10%.
- If you aren't able to present by end of 12/9 due to being unprepared 10% off for Wednesday as well, clock keeps ticking 10% per day until Git pull request. You will need to make arrangements with me to present individually **on my schedule**.

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

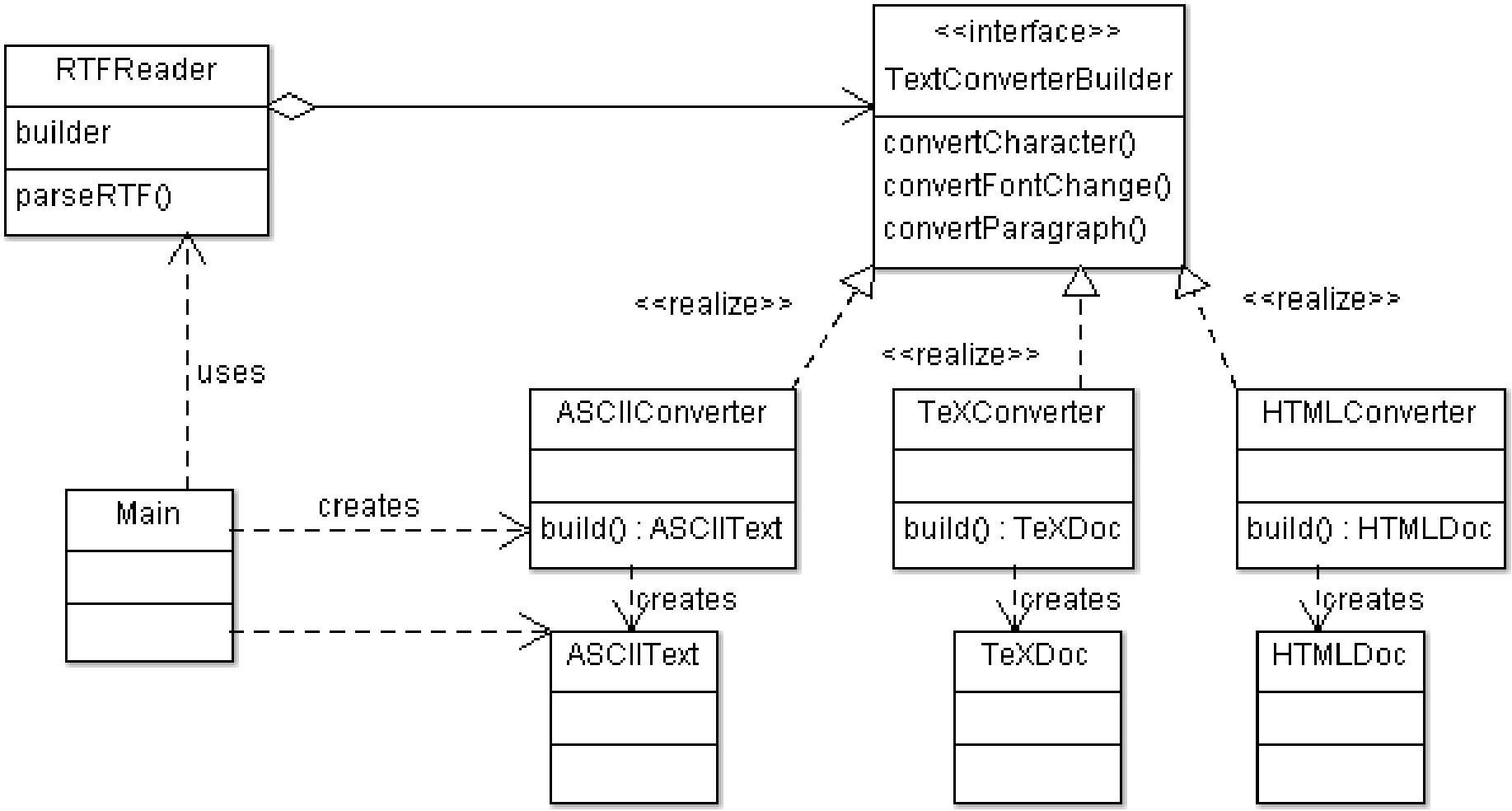
Builder UML


Director
buildProduct()
Builder
buildPart()
Product
ConcreteBuilder1
buildPart()
ConcreteBuilder2
buildPart()

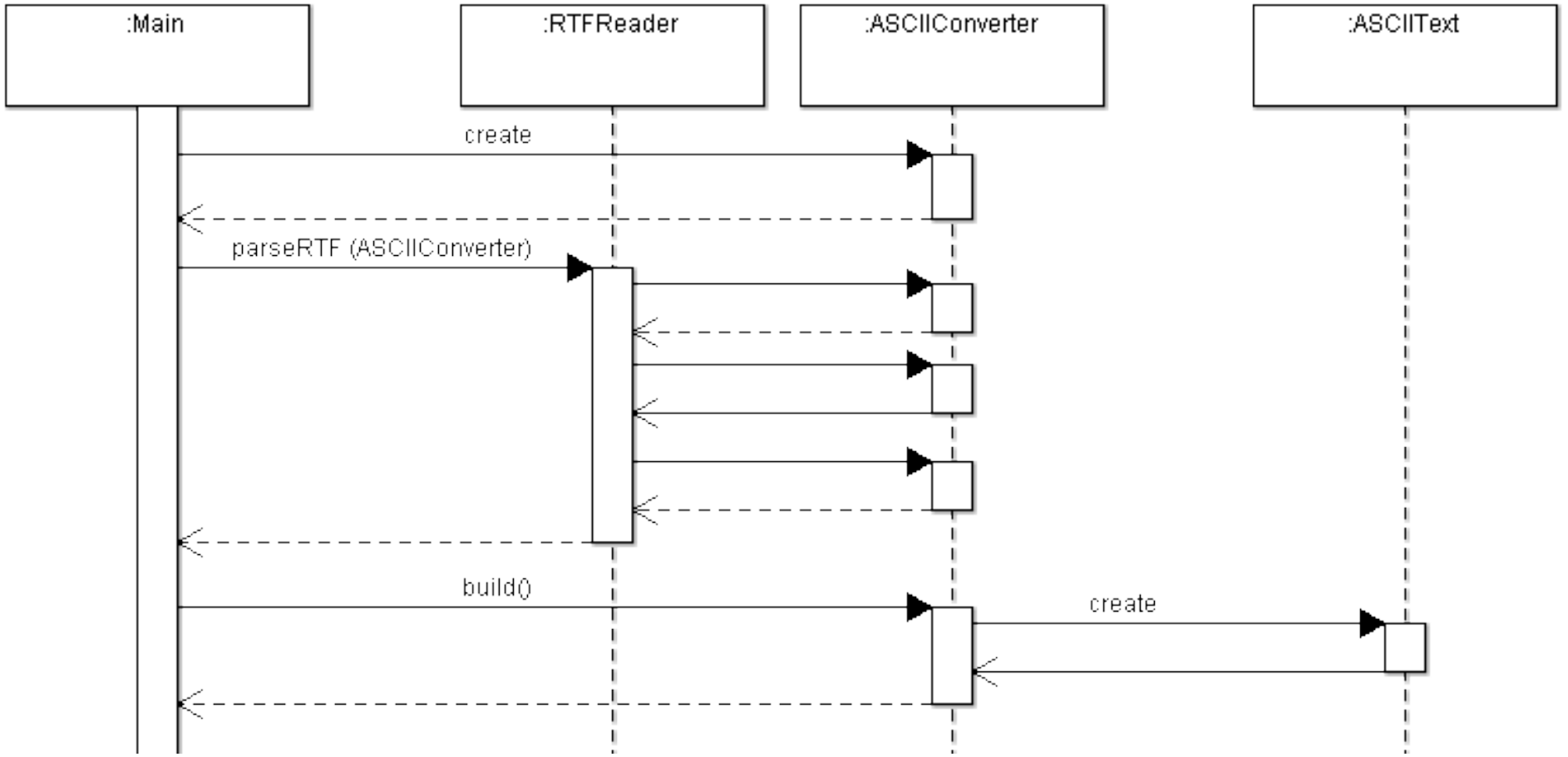
Builder roles

- **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

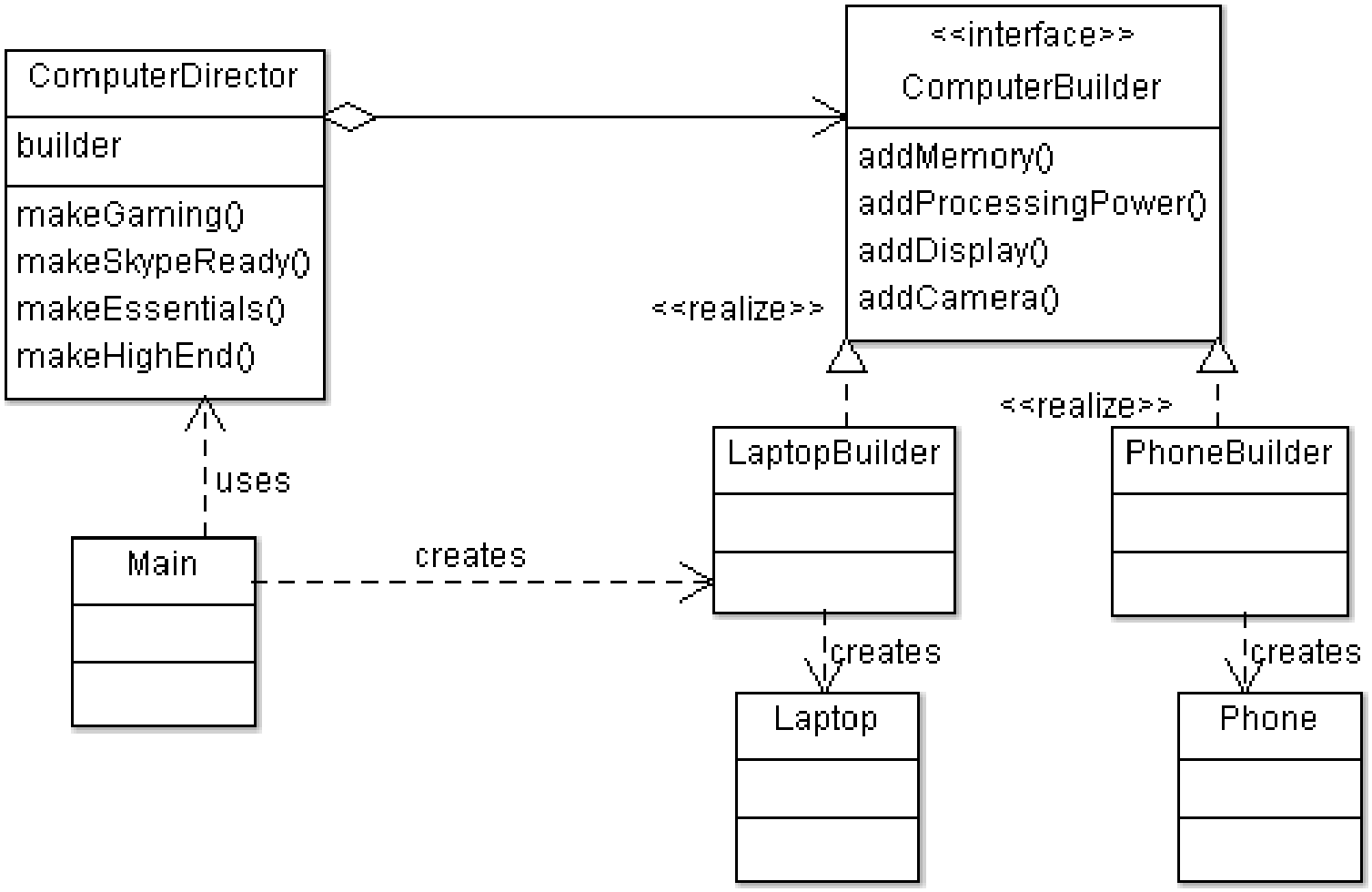
Document converter



Builder sequence



Computer/phone builder

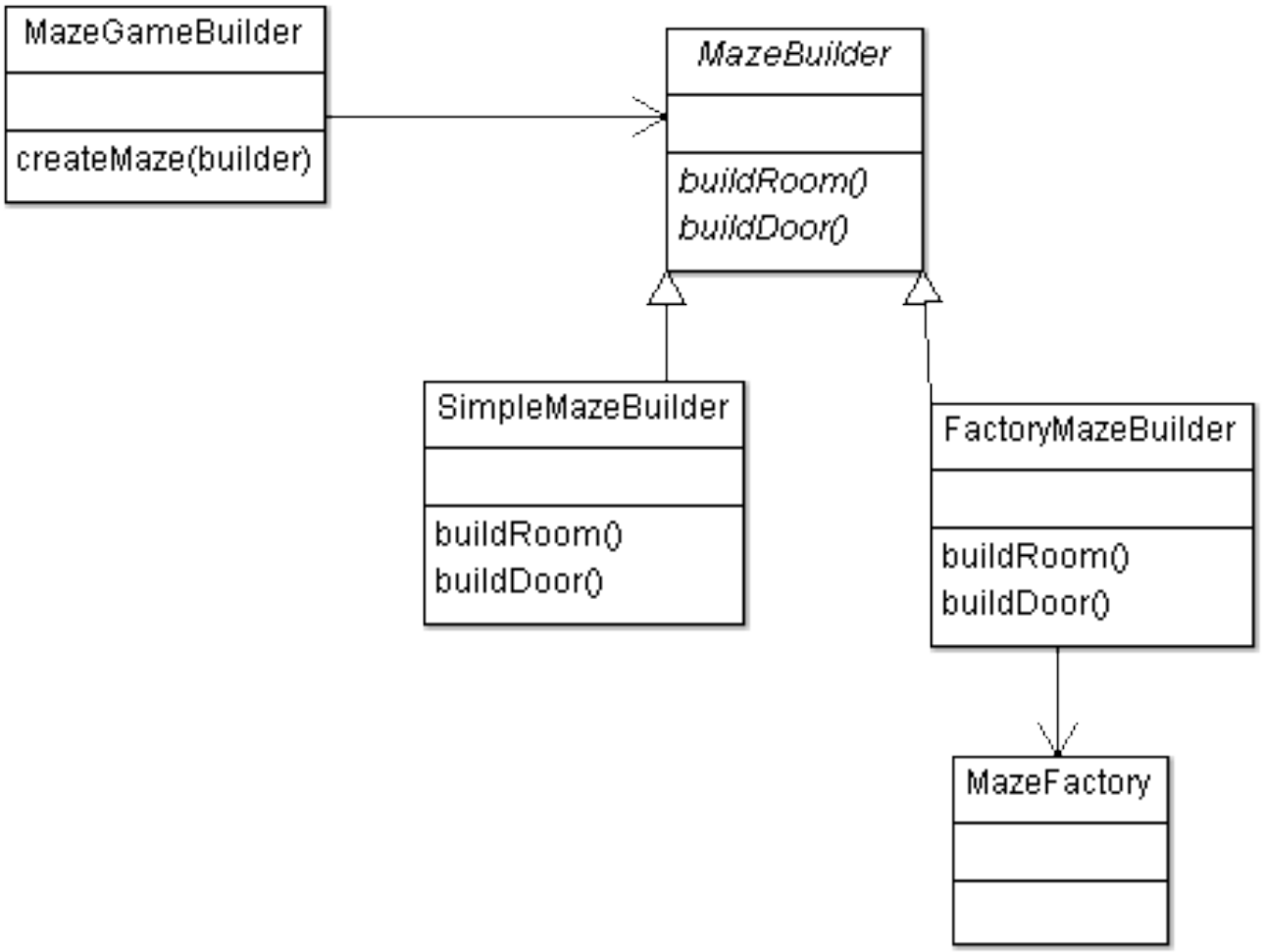


Builder pattern

- 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

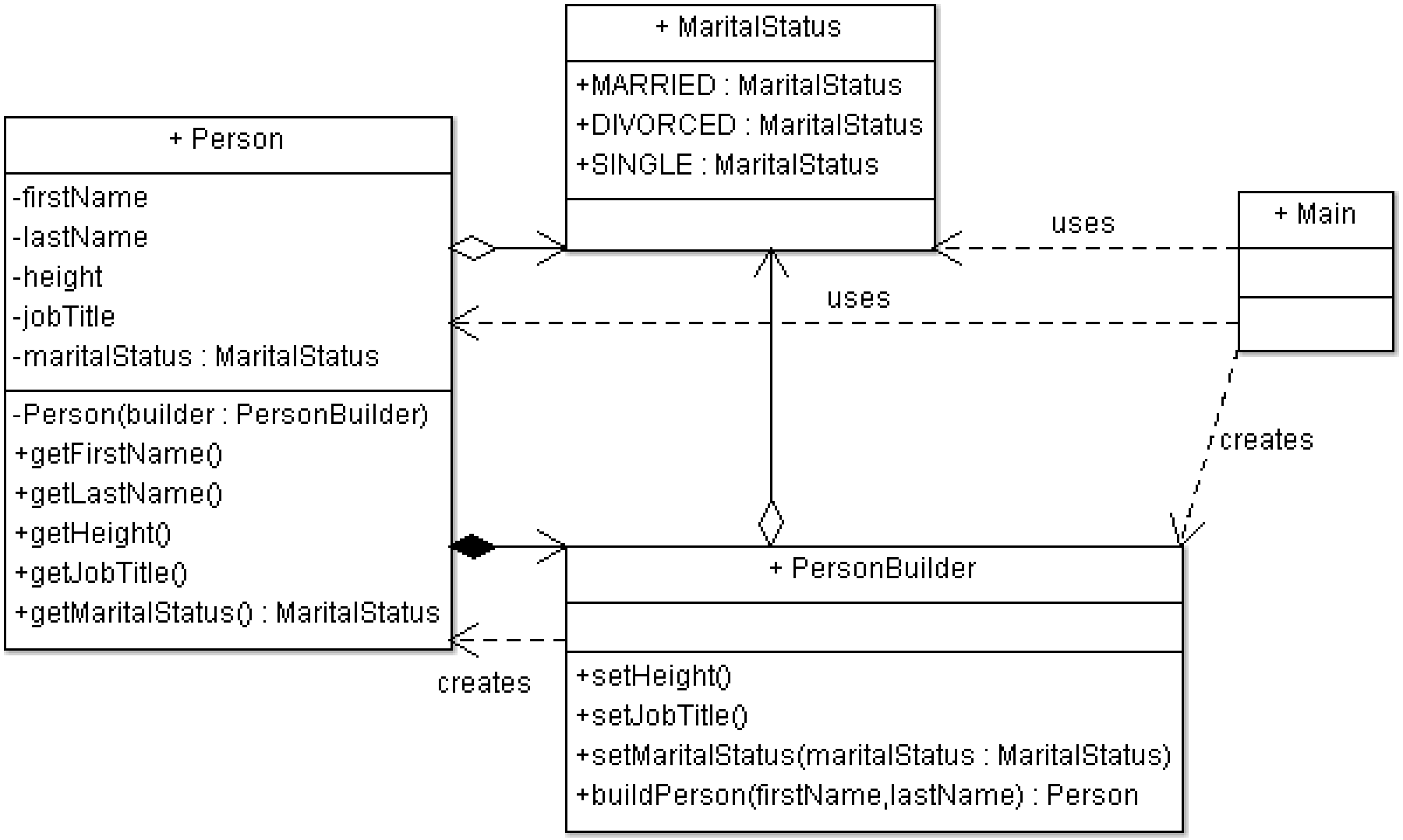
Within nested builder common to return an instance of that builder with each call:

```
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();
}
```

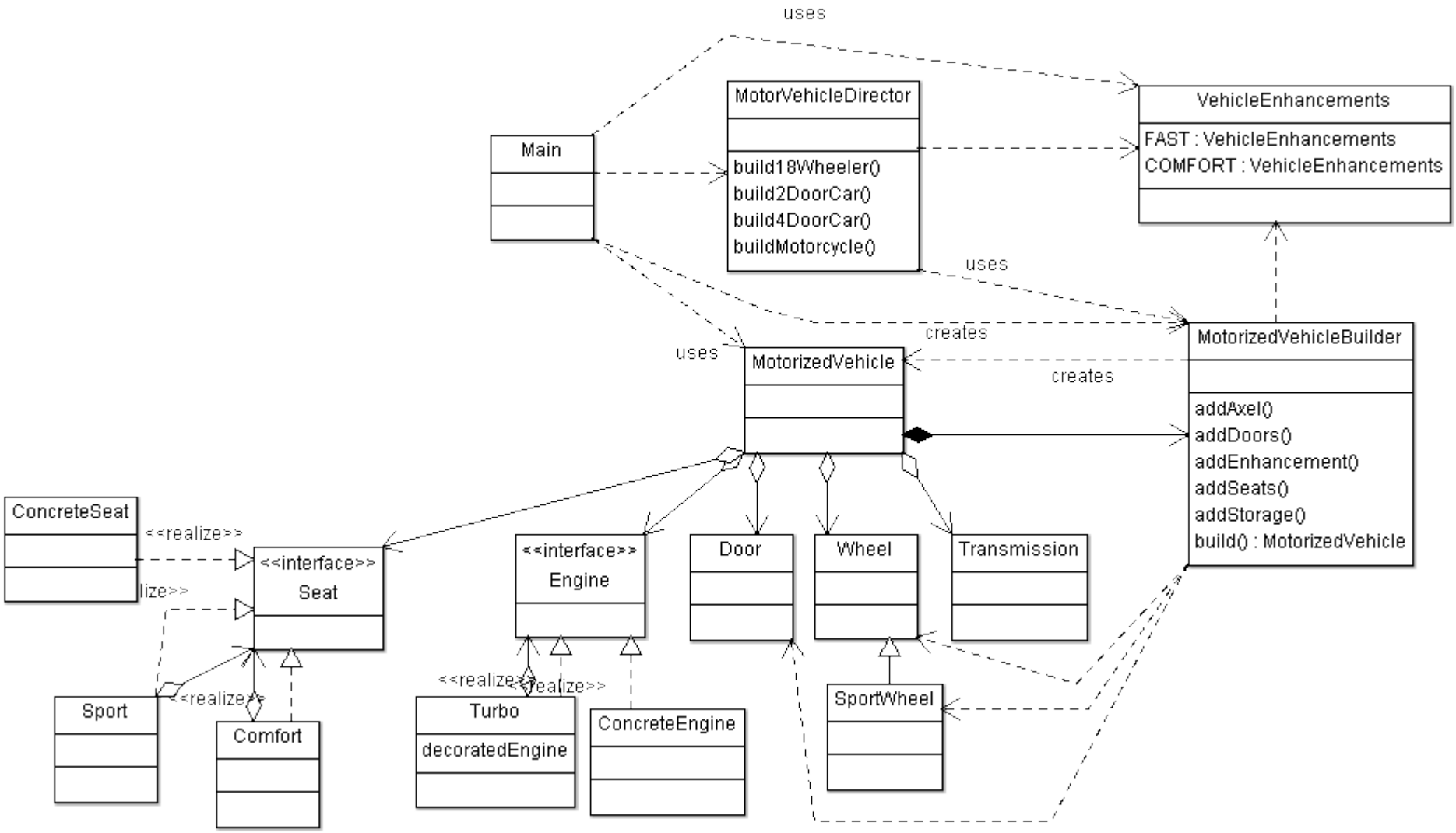
Simple builder version



“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