CPSC 327: Object-oriented Programming

Lecture 2

9/3/2021

Professor Barron

Lecture quizzes

- About half the class has done the Lecture 1 quiz
- Due tomorrow night
- Today’s quiz due Tuesday night (extra day for the long weekend)

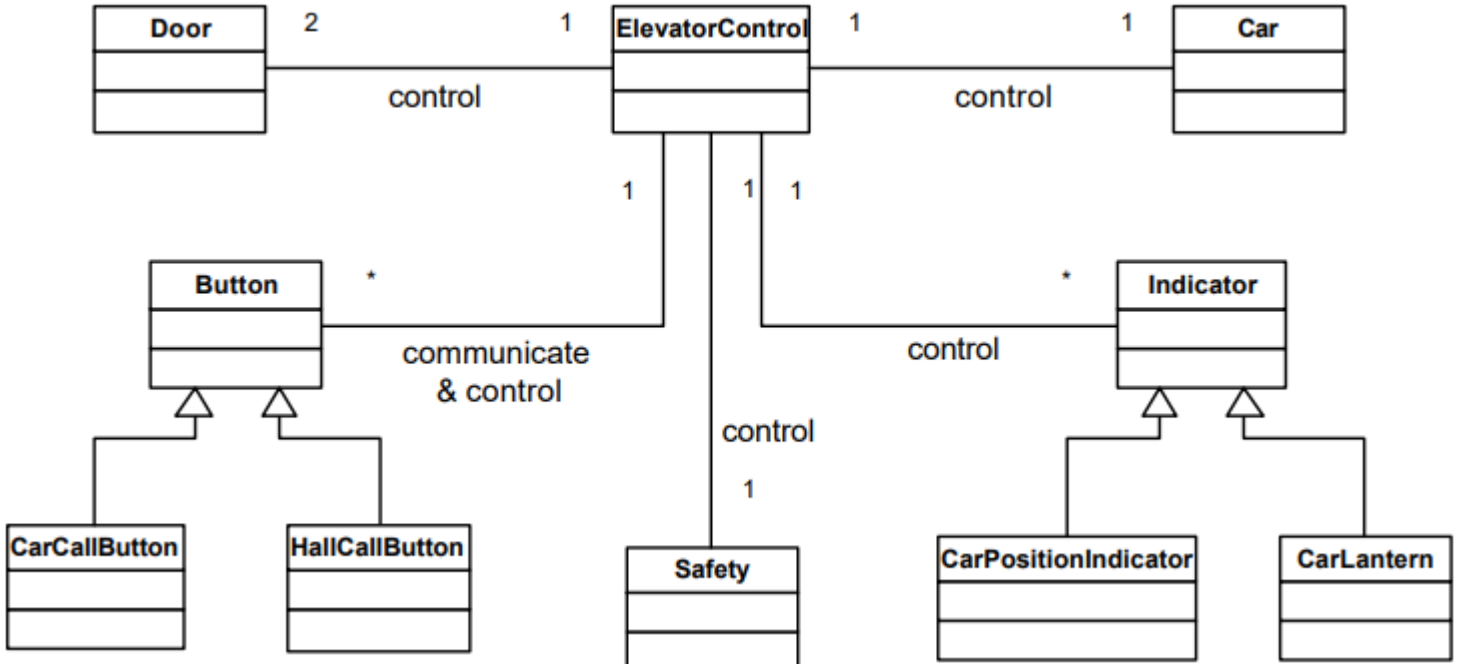
Outline

- Last time
  - What/why of OOP
  - Objects
  - Instances
  - Data
  - Behaviors
- Today
  - More key concepts
  - Relationships between objects
  - More UML
  - Python implementation

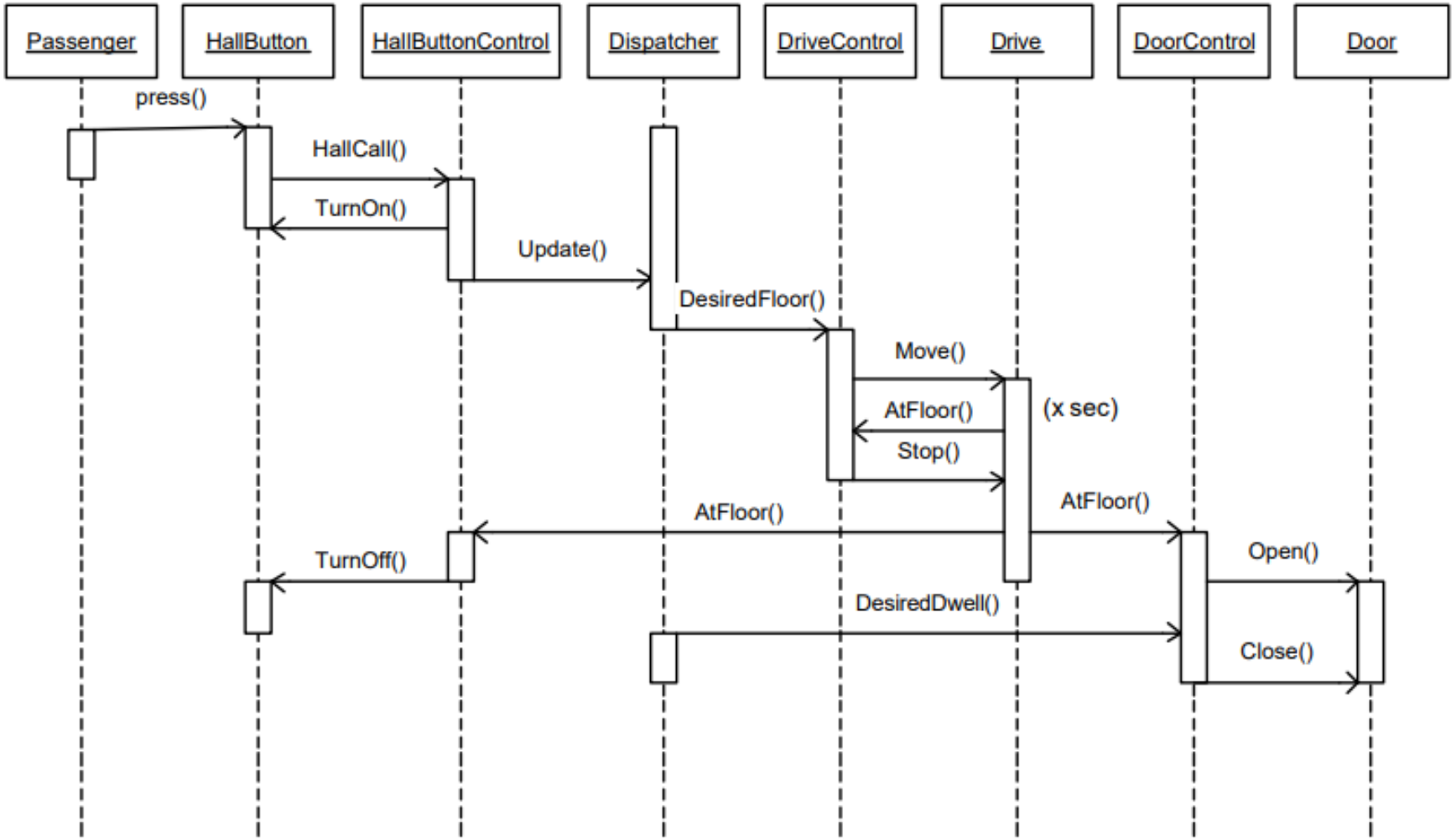
Start of Chapter 2

Elevator UML class diagram

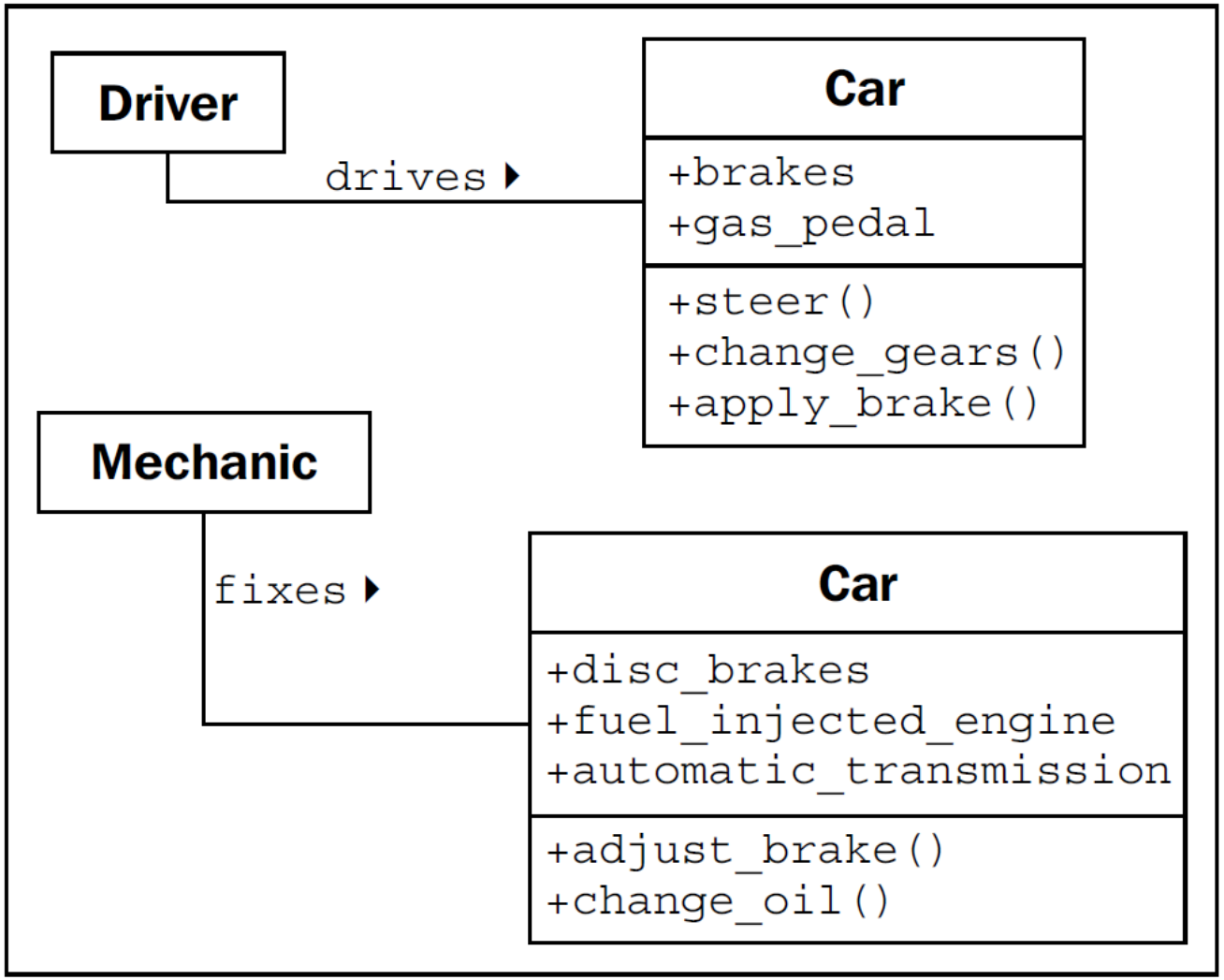
- https://www.cs.cmu.edu/~luluo/Courses/18540PhDreport.pdf



Elevator UML sequence diagram



Different levels of abstraction



More key concepts

- Abstraction
- Encapsulation
- Information hiding
- Public interface

All closely related!

More key concepts

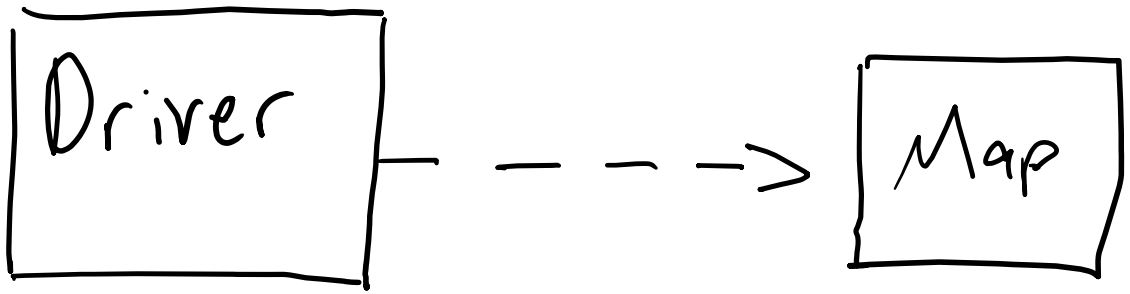
- Abstraction
  - Removing details that aren’t important
  - Focus on what is needed
- Encapsulation
  - Bundling details within a Class
- Information hiding
  - Hiding details that other objects do not care about
- Public interface
  - Attributes and methods used to interact with an object from the outside
  - Only what is necessary while the rest remains hidden
  - Need to be very careful when designing or changing this

Relationships

- Dependency
- Association
- Aggregation
- Composition
- Inheritance

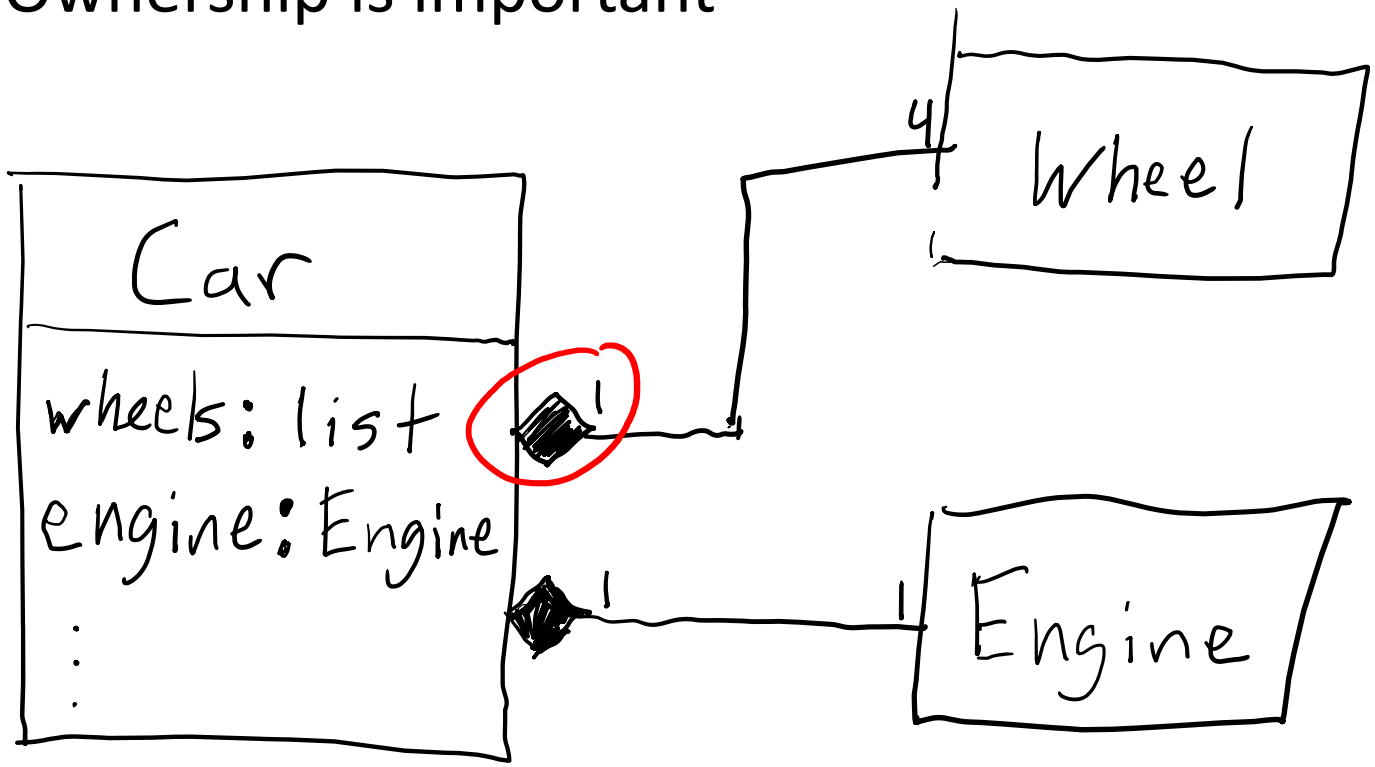
Dependency

- One object “uses” another object
- Very broad
- If object A depends on object B and B is changed, then A may need to change
- Taking an object in as a method parameter is a common type of temporary dependency



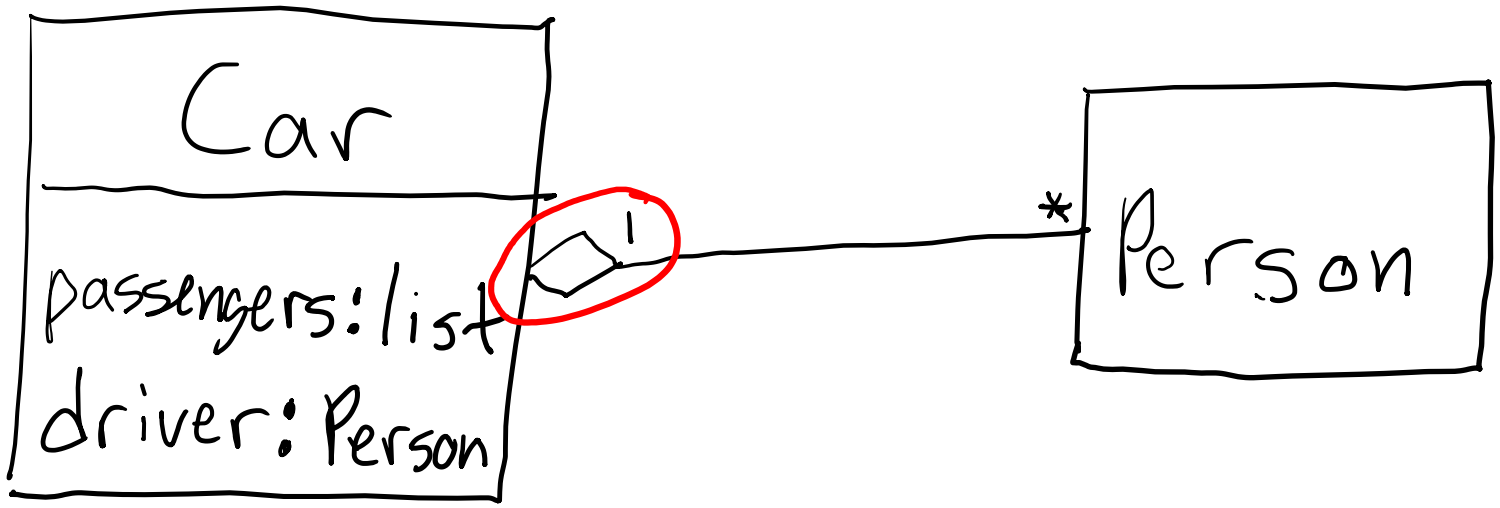
Composition

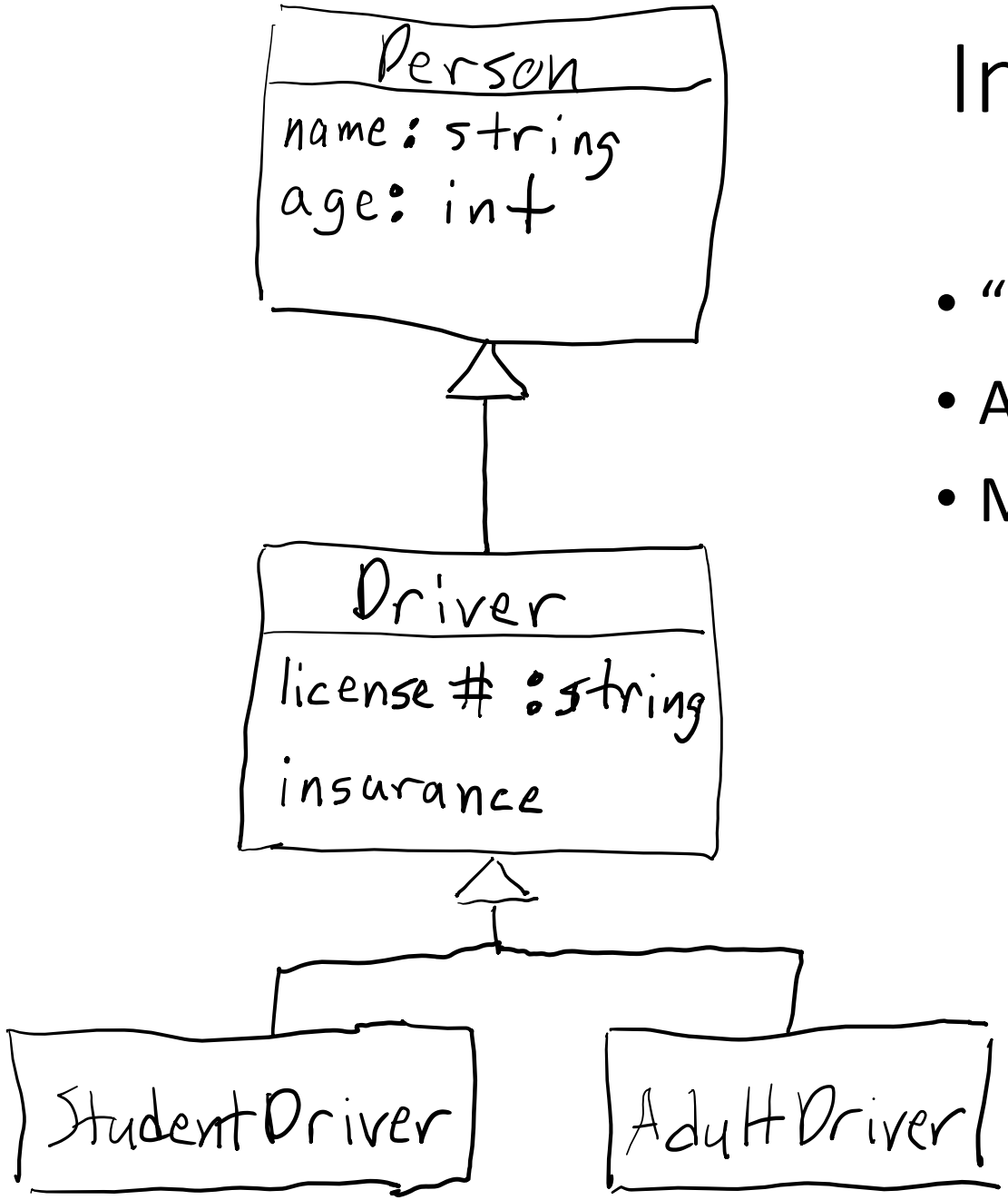
- “Has-a” relationship
- A car has an engine, windshield, wheels, doors, seats, etc.
- Ownership is important



Aggregation

- “Has-a” relationship
- Nearly the same as composition
- Can the object exist on its own?
- Aggregation is more general than composition





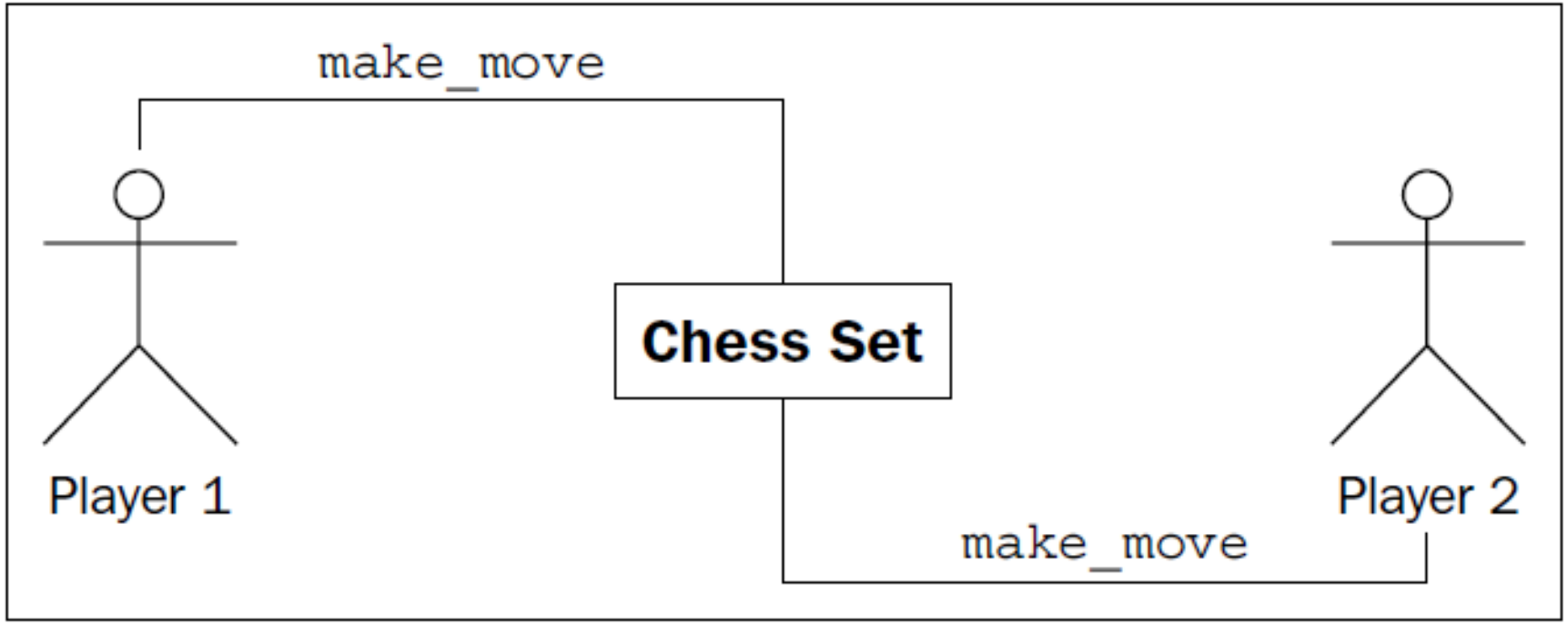
Inheritance

- “Is-a” relationship
- A driver is a person
- More specific version of an object

Chess example

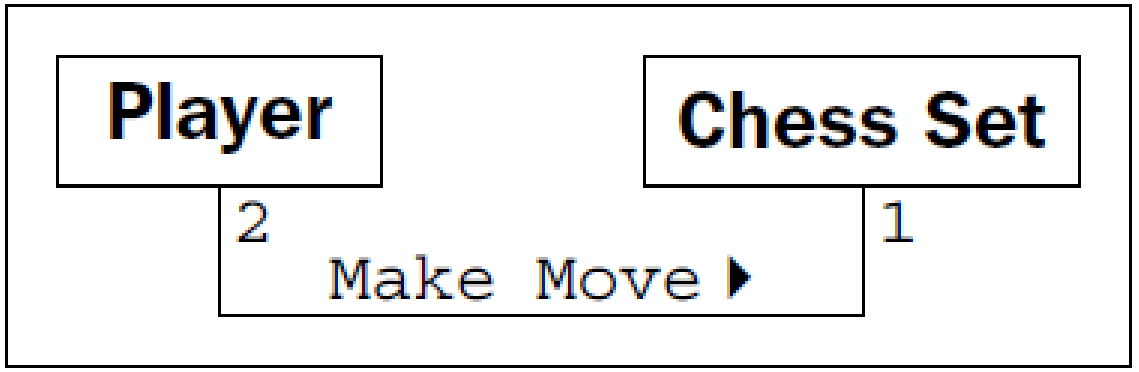
Chess UML

- Object diagram

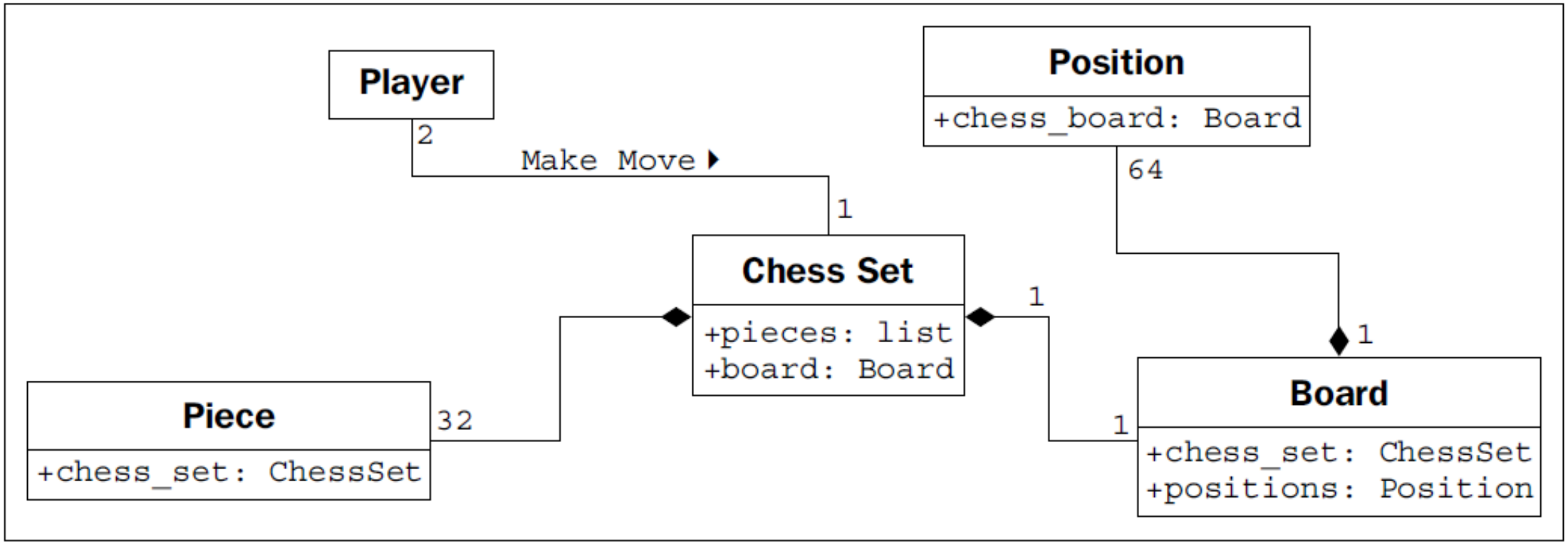


Chess UML

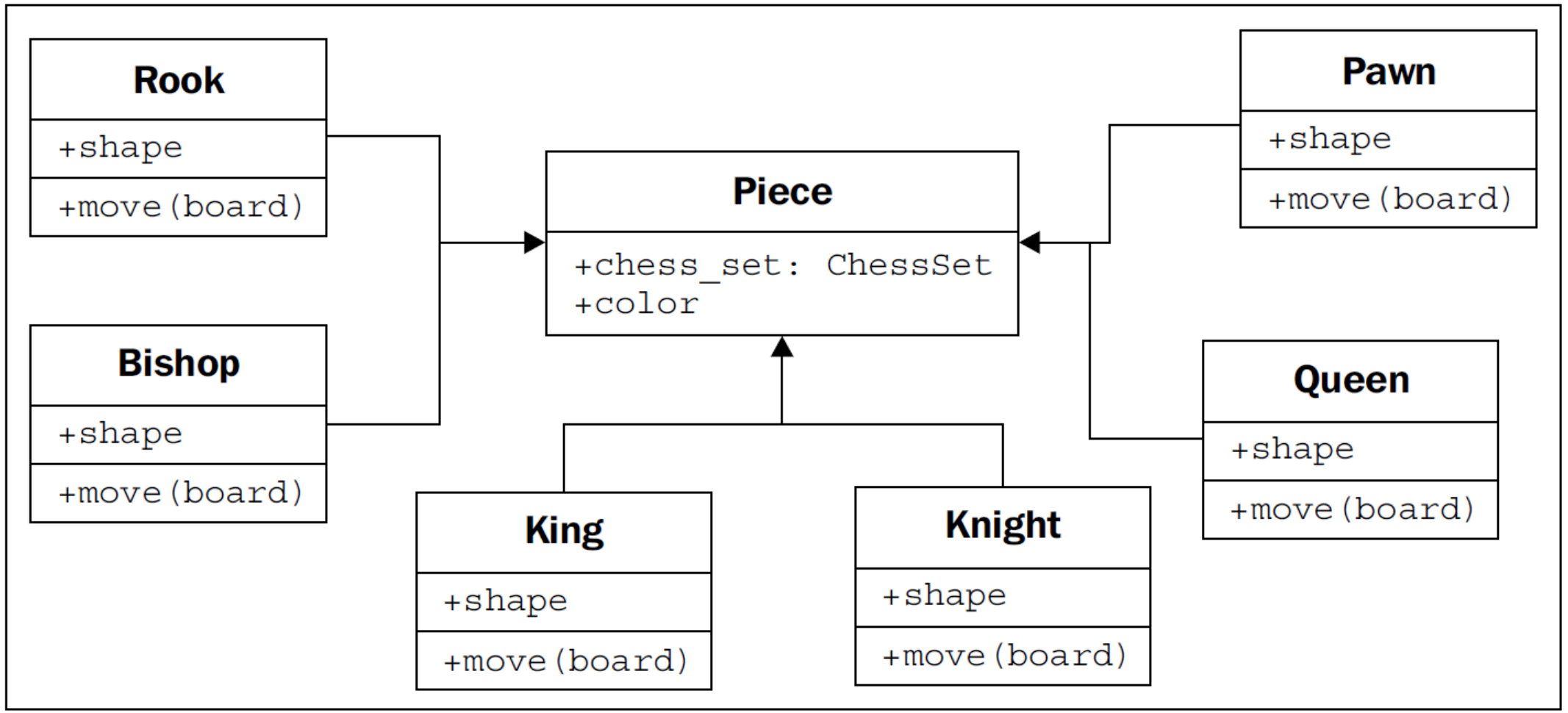
- Class diagram



Chess UML


Player
2
Make Move
1
Position
+chess_board: Board
64
1
Chess Set
+pieces: list
+board: Board
1
Piece
+chess_set: ChessSet
32
1
Board
+chess_set: ChessSet
+positions: Position

Chess UML


Rook
+shape
+move(board)
Bishop
+shape
+move(board)
Piece
+chess_set: ChessSet
+color
King
+shape
+move(board)
Knight
+shape
+move(board)
Pawn
+shape
+move(board)
Queen
+shape
+move(board)

Polymorphism

```
p1 = Queen()
p2 = King()

pieces = [p1, p2]

for p in pieces:
    p.move(board)
```

King and Queen are both Piece objects

All Piece objects have a move method

Python duck typing

- "If it walks like a duck and it quacks like a duck, then it must be a duck"
- Goes a step further than polymorphism
- Doesn’t even matter if p inherits from Piece or not
- As long as move is defined for p, it will run
- Runtime error if the object does not have a move method
- EAFP style, Easier to ask forgiveness than permission
  - As opposed to LBYL style, look before you leap

Live examples

- Pipenv
- Creating classes and objects in Python
- public vs private
- Class vs static vs instance