Object-Oriented
Programming

Multiplicity and ArrayLists

Outline

- Multiplicity
- ArrayLists
- Practice Exercise

Review of Class Relationships

DIRECTED
Association

- Source Class has a method that accepts Destination Class as parameter
- Destination Class has NO method that accepts Source Class as parameter

BIDIRECTIONAL
Association

- Source Class has a method that accepts Destination Class as parameter
- Destination Class has a method that accepts Source Class as parameter

REFLEXIVE
Association

- Source and Destination Class are the same
- Source Class has a method that accepts itself as parameter

Review of Class Relationships

AGGREGATION

Part-Whole

- Source Class (Part) is a data type of one of the properties of the Destination Class (Whole)
- Destination Class (Whole) has a method or constructor that accepts the Source Class (Part) as a parameter
- The accepting method sets its parameter to the property

COMPOSITION

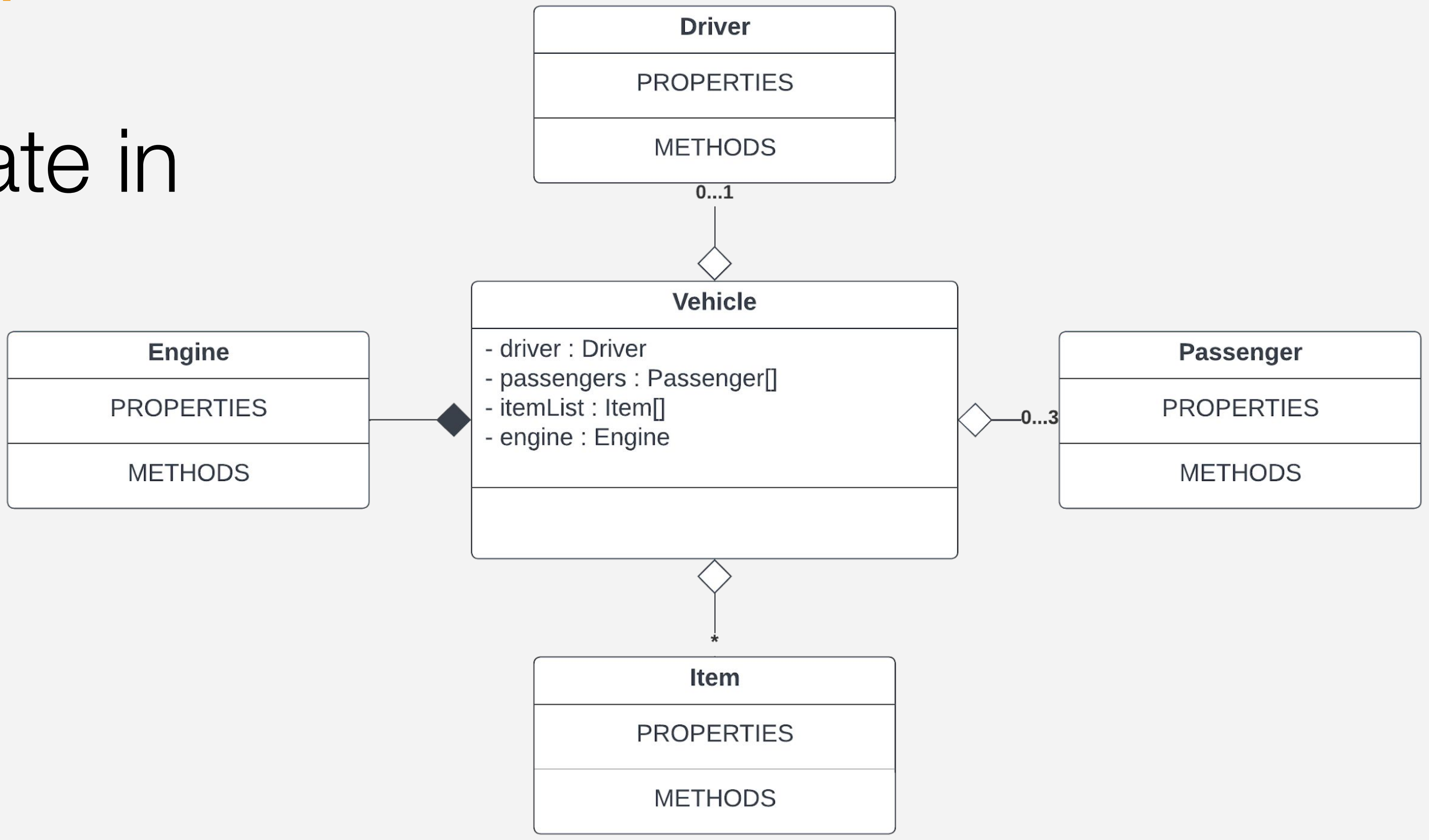
Part-Whole

- Source Class (Part) is a data type of one of the properties of the Destination Class (Whole)
- Destination Class (Whole) has NO method that accepts Source Class as parameter
- Destination Class (Whole) creates an object of type Source Class (Part) inside itself, and sets its property with the object

Any questions? ☺️

Multiplicity

- Defines the count of objects that are expected to participate in a relationships



Multiplicity

- Notation

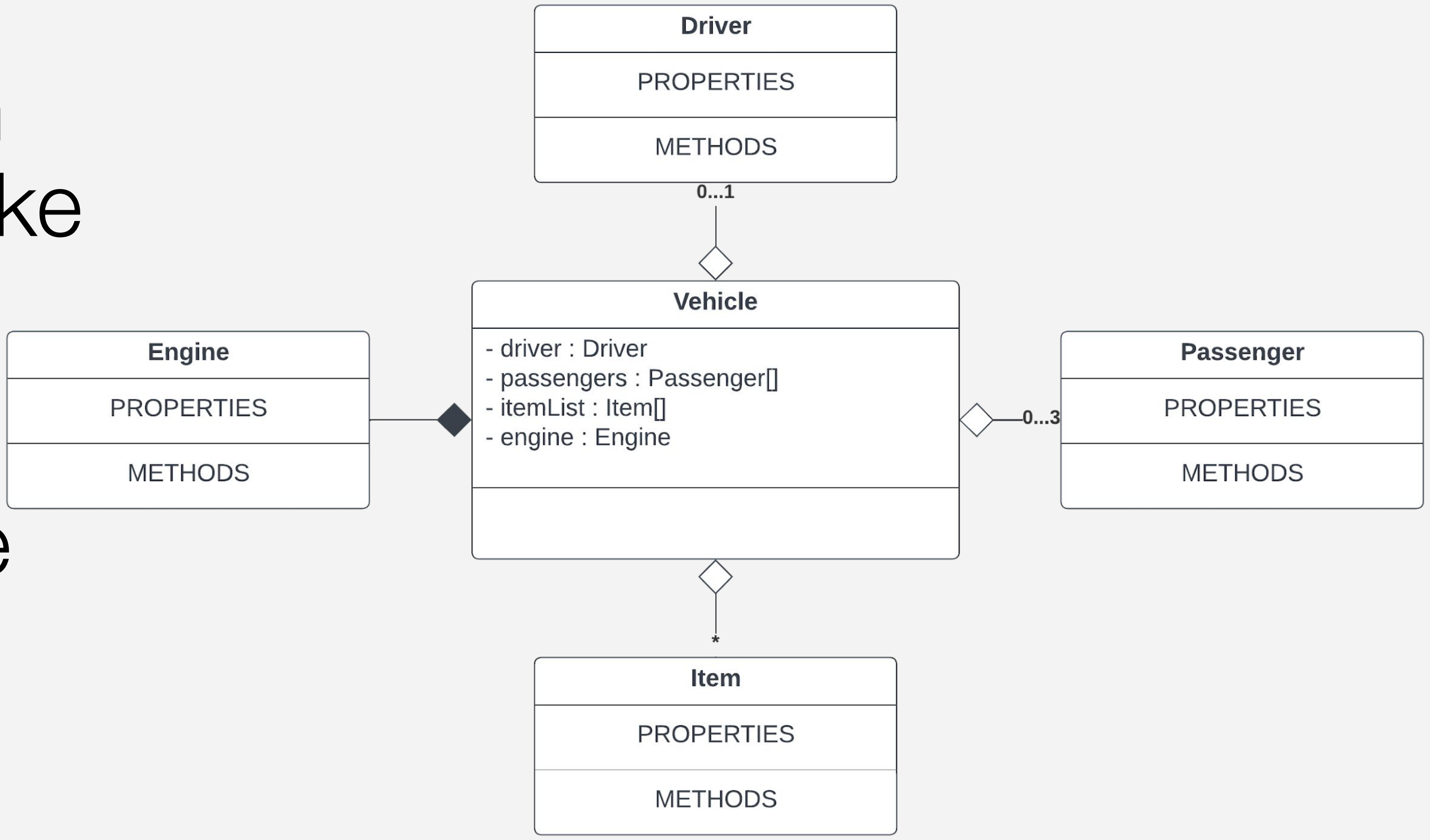
•1	exactly 1, no more, no less
•0…1	zero or 1
•*	(start) zero or more
• n…m	n (lower bound) to m (upper bound)

- When no multiplicity is specified, it is safe to assume that there's only 1 instance

Multiplicity

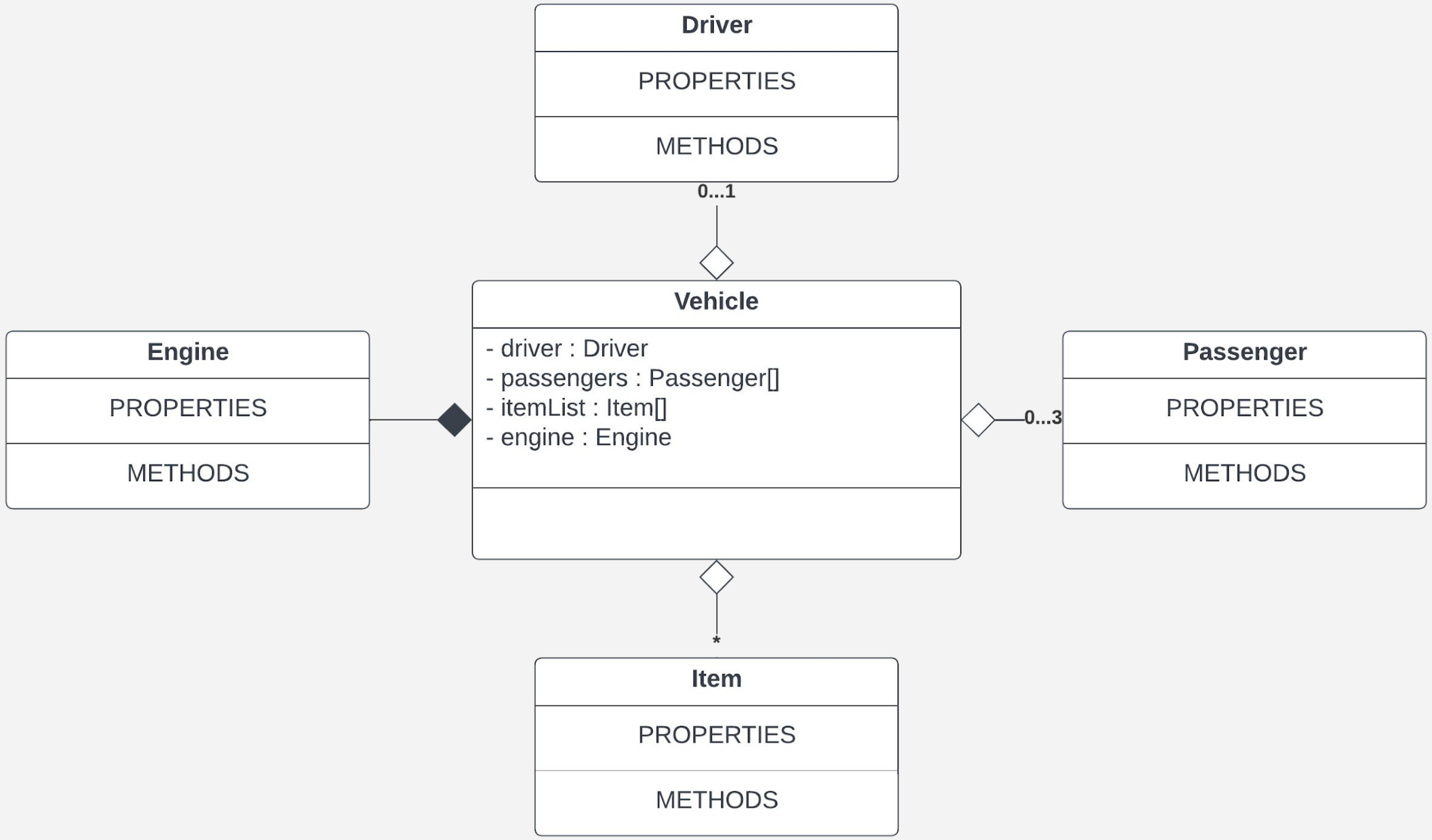
- Sometimes the multiplicity is written into the attributes, like
  - Item[*]
- For our class,

  declare them on the relationship line



Multiplicity

- Associations are typically 1:1, so there’s usually no need to specify the value
- Best to specify for aggregation and composition



Array vs ArrayList

- Arrays (what most are familiar with) are fixed in terms of size

```
Student[] studentArr = new Student[10];
```

- ArrayList is a dynamic array (no need to specify the size) that extends the List interface

```
ArrayList<Student> studentList = new ArrayList<Student>();
```

Technically, you can also declare an ArrayList like…

```
ArrayList studentList = new ArrayList();
```

…but it isn't considered good practice as the compiler can't be certain of the type expected

Array vs ArrayList

- Arrays can contain primitive and reference types

```
int[] arr = new int[10];
Integer[] arr = new Integer[10];
```

- ArrayLists can only contain reference types

```
ArrayList<Integer> arrL = new ArrayList<Integer>();
ArrayList<Object> arrL = new ArrayList<Object>();
```

Important!

- If you’re going to use ArrayList, you must **import** it

```
import java.util.ArrayList;
```

Adding values to Array / ArrayList

- Arrays

```
arr[<index>] = <value>;
```

- ArrayList

```
arrL.add(<object>);
```

Getting values from Array / ArrayList

- Arrays

```
var = arr[<index>];
```

- ArrayList

```
var = arrL.get(<index>);
```

Getting size of Array / ArrayList

- Arrays

```
arr.length;
```

- ArrayList

```
arrL.size();
```

Questions? ☺️

The official Java API is your best friend ☺️

https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

Practice Exercise

- Open notes and by MP Pairs
- Recent slides are already available in Canvas
- Join the breakout room number of your pair
    - *Ex. MP Pair 1 -> Room 1*
- You can unmute and talk actively
- Please share one screen throughout the exercise

Practice Exercise ☺️

Let's head over to Canvas

Keep learning…