CSSE 220 Day 11

Designing Classes and Object Oriented Design

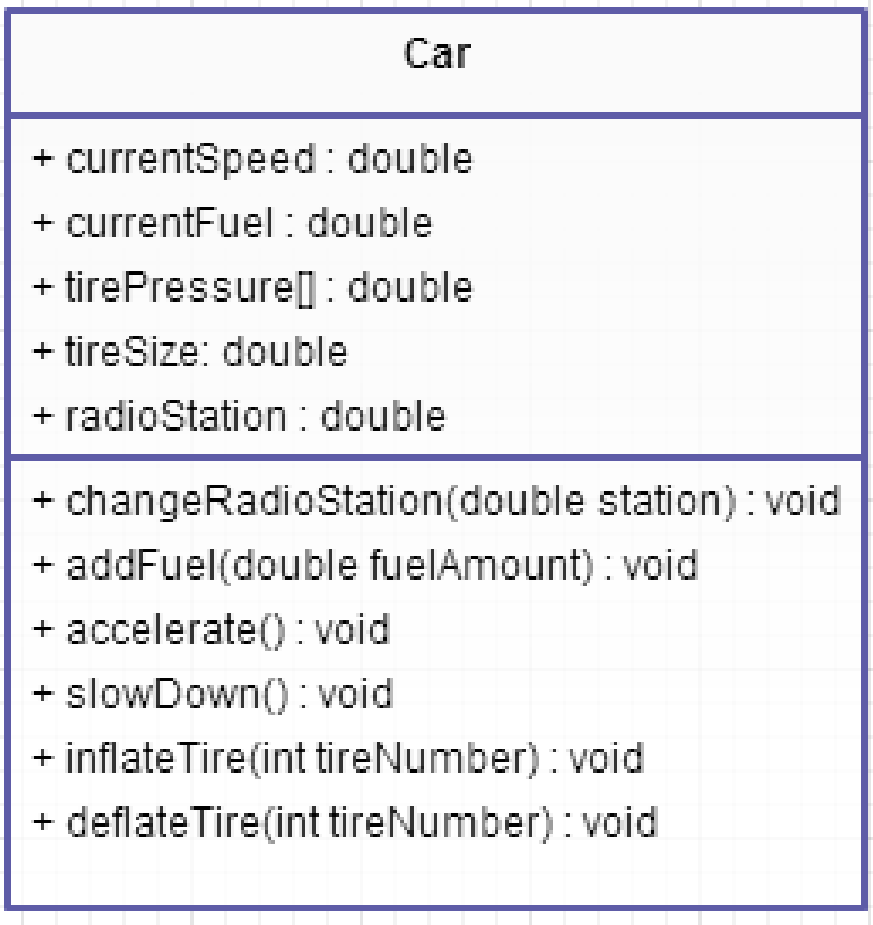
It starts with good classes…

WHAT IS GOOD OBJECT-ORIENTED DESIGN?

Why do we use classes?

- Model Real Life Scenarios
- Maintainability
  - Modular Structure
  - Hides Implementation Details
  - Data Hiding (helps prevent Data Corruption)
  - Separation of Responsibilities
- Higher Quality Software
- Software/Code Reuse

Designing a Car Class



Does something feel wrong about this?

- Cluttered
- Too Many Responsibilities
- Difficult to Understand
- Hard to Maintain

Good Classes Typically

- Come from nouns in the problem description
- May…
  - Represent single concepts
    - `Circle`, `Investment`
  - Represent visual elements of the project
    - `FacesComponent`, `UpdateButton`
  - Be abstractions of real-life entities
    - `BankAccount`, `TicTacToeBoard`
  - Be actors
    - `Scanner`, `CircleViewer`
  - Be utility classes that mainly contain static methods
    - `Math, Arrays, Collections`

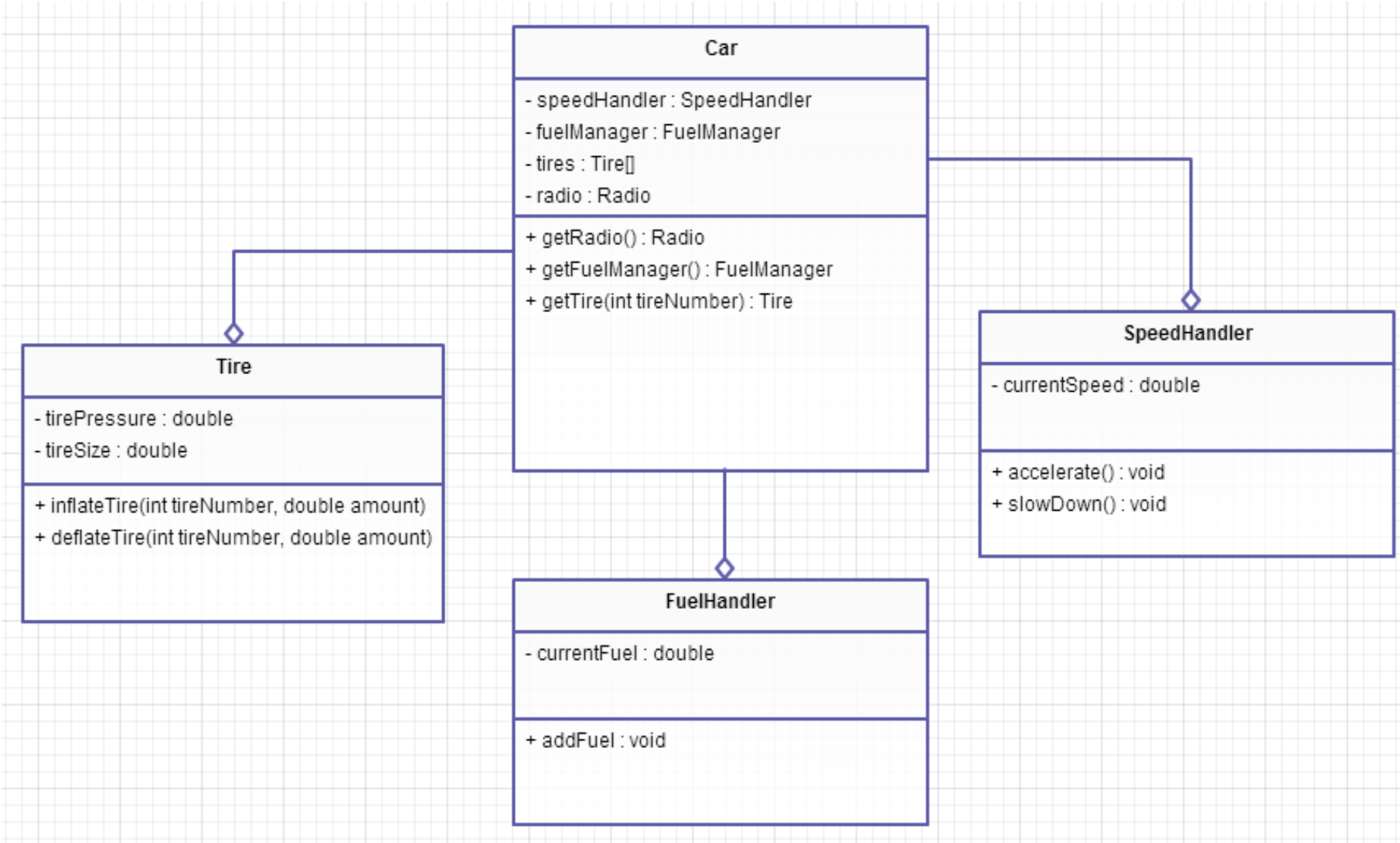
What Stinks? Bad Class Smells*

- Can't tell what it does from its name
  - `PayCheckProgram`
- Turning a single action into a class
  - `ComputePaycheck`
- Name isn't a noun
  - `Interpolate`, `Spend`

Function objects are an exception. Their whole purpose is to contain a single computation

*See http://en.wikipedia.org/wiki/Code_smell
http://c2.com/xp/CodeSmell.html

A Better Car Class



Common Code Smells

- Duplicated Code
- Long Method
- Large Class
- Too Many Parameters
- Feature Envy
- Inappropriate Intimacy
- Lazy Class
- Complex Conditionals
- Magic Numbers

Accessors and Mutators

- **Accessor method**: accesses information *without changing any*

- **Mutator method**: *modifies* the object on which it is invoked

Immutable Classes

- Accessor methods are very predictable
  - Easy to understand
- **Immutable classes**:
  - Have only accessor methods
  - No mutators
- Examples: **`String`**, **`Double`**
- Is **`Rectangle`** immutable?

Immutable Class Benefits

- Easier to reason about, less to go wrong
- Can pass around instances “fearlessly”

Q9

Side Effects

- Side effect: any modification of data

- Method side effect: any modification of data *visible* outside the method
  - Mutator methods: side effect on implicit parameter
  - Can also have side effects on other parameters:
    -
```
public void transfer(double amt, Account other)
{
    this.balance -= amt;
    other.balance += amt;
}
```

Avoid this if you can!

Perspective Matters

- How You Model Objects Depends on Your Code's Objectives
  - Movies
    - IMDB
    - Amazon
  - Car
    - Mechanic
    - Car Pool System
    - Dealership

Work with your table and map out the classes given to you

CLASS DESIGN EXERCISE

WORK TIME