Inheritance

Tobias Hanf, Maik Göken

January 9, 2023

Learn Programming with Java

Outline

Addition: 4 Pillars of OOP

Revision

Inheritance

Polymorphism

Exercise

Addition: 4 Pillars of OOP

4 Pillars of OOP

Different Defenition of the 4 pillars:

- Abstraction
  - Hide complexity
  - And Implementation
- Inheritance
  - Of attributes and methods
  - Unit 07
- Encapsulation
  - data hiding
  - today
- Polymorphisim
  - Single Interface, Multiple functionality
  - Unit 07

Abstraction

- Only show essential information
- Hide implementation detail
- Helps with understanding large systems
  - Only concerned with what is done
  - And not how it's done

Revision

Quiz

https://pingo.coactum.de/753441

Inheritance

Inheritance

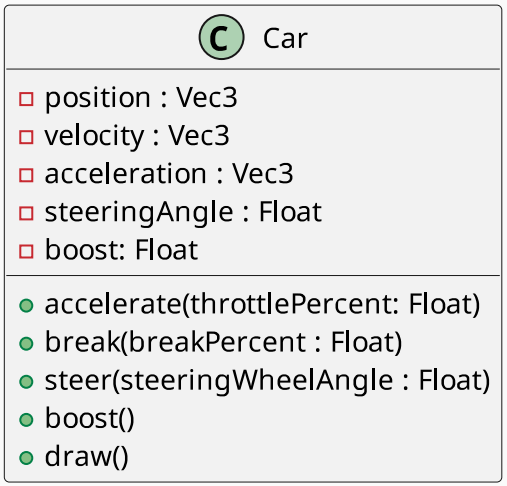
- Resusing the properties (data + code) of another class
- Used for creating class hierarchies
    - Conceptional hierarchies
- New properties can be added
    - Extending existing classes

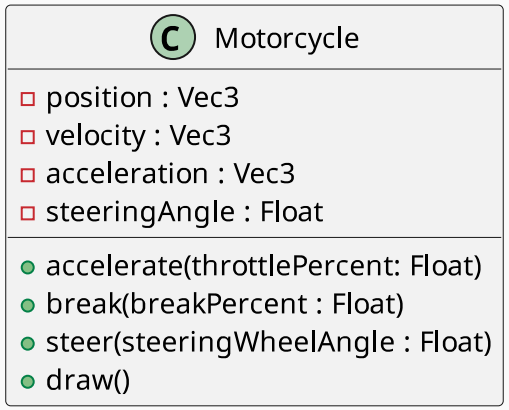
Members of Inheritance

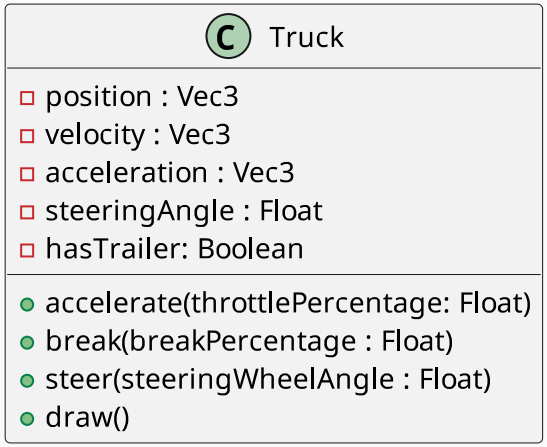
- Superclass: class from which we inherit properties
  - Also named: Parent class, Base class
  - Object is called Parent Object
- Subclass: class which inherites properties
  - Also named: Child class, Derived class
  - Object is called Child Object
  - Can access (some) properties of the superclass
  - Can `@Override` methods

Example: Vehicle

We want to model Vehicles for a Racing game. There are three different vehicle types in this game: Cars, Motorcycles and Trucks. Each type has a unique driving physics which should be implemented by the corresponding class.

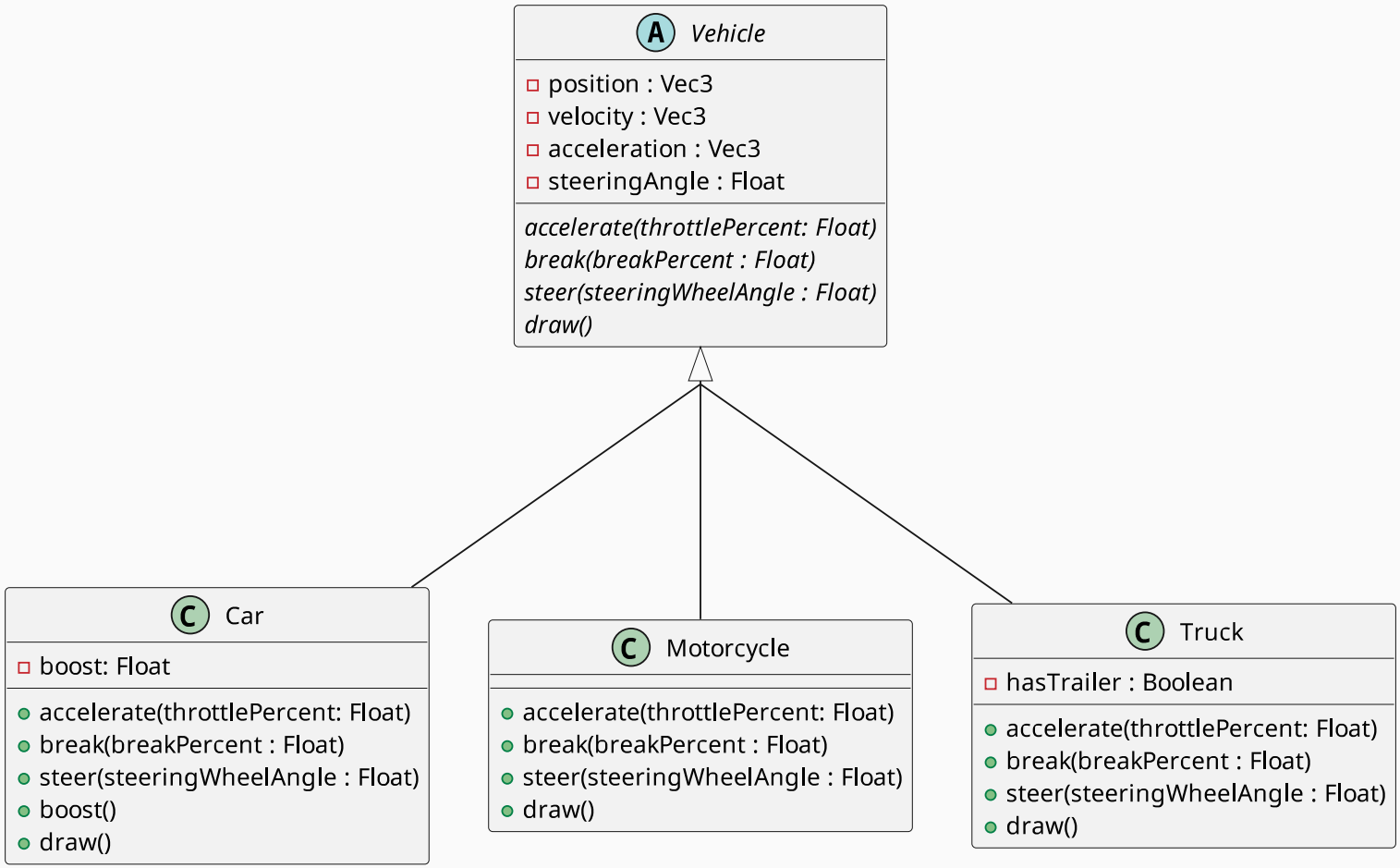






Example: Vehicle

We have a lot of redundant code (definitions). To circumvent this a Superclass can be introduced:



Inheritance in Java

- Can only inherit from one Superclass
- Can access `public` and `protected` properties if the Superclass
- Can call the constructor of the Superclass via `super()`
- Can access properties of Superclass via `super` (like `this`)

```
1 class Vehicle {
2     ...
3 }
4
5 class Car extends Vehicle {
6     ...
7 }
8
9 ...
```

Inheritance in Java

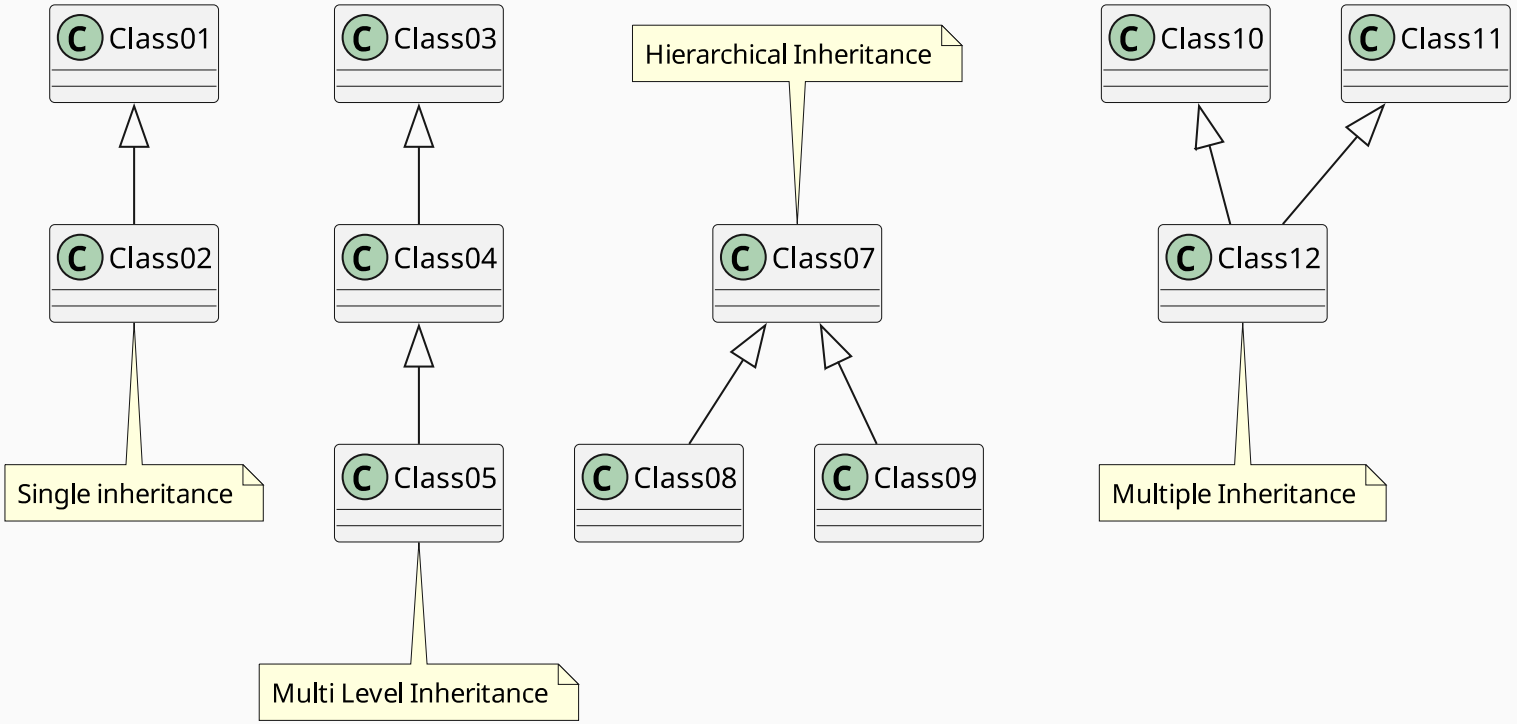
```
class <Subclass-name> extends <Superclass-name> {
    public <Subclass-name>() {
        // Call Superclass constructor
        super();
    }

    @Override
    public <Superclass-Method>(...) {
        super.<Superclass-Method>(...);
    }
}
```

Ways of constructing Inheritance

- Generalization
  - Bottom-Up
  - Extract common features into Superclass
  - Example: First Car, ... then Vehicle
- Specialization
  - Top-Down
  - Create a spezialisation from a Superclass
  - Example: First Vehicle then Car, ...

Types of inheritance



Polymorphism

Polymorphism

- Having a single Interface
- But different implementations
- Two different types:
  - Ad hoc Polymorphism
  - Subtyping

Ad hoc Polymorphism

- Depending on the type of the Argument
- A different implementation of a function is choosen
- Also known as overloading a function
- Java is only interested in the:
  - Function name
  - Parameter list

Example: sum

```
1  // f1
2  int sum(int a, int b) {
3      System.out.println("Sum int");
4      return a + b;
5      }
6
7  // f2
8  float sum(float a, float b) {
9      System.out.println("Sum float");
10     return a + b;
11 }
12
13 sum(2, 5);   // calls f1
14 sum(2.f,5.f) //calls f2
```

Subtyping

- `Override` methods of Superclass
- Implement own (specialised) logic for subtype
- Use the annotation `@Override`
- Java will use the most specific implementation

```
1 class Vehicle {
2     public void accelerate(float throttlePercentage) { }
3 }
4
5 class Car extends Vehicle {
6     @Override
7     public void accelerate(float throttlePercentage) { }
8
9 }
```

Exercise

University Resource Planner V2

In the new version of the University Resource Planner the customer wants to be able to track the courses offered by the University. Each course should have a course name, a teacher which holds that course and a list of students currently enrolled in that course.

A teacher should have a name, a year of brith, the current salary and a list of all lessons he/she teaches.

The method `toString()` should be implemented for all classes. It should return a String containing meaningful information about an object and what type it is (eg. Teacher, Student, Course).

Try to apply the newly learned principlas (Inheritance, Polymorphism)

Creating a Class diagram

Discuss how the class structure of the version could look like and create a small diagram (together).

- What classes should exists/be added
- What attributes and methods should each class have
- What inheritance should exist

Implement the new version

Implement the things we discussed on the last slide by creating a copy of the old version and adding the new features.