CPSC-224
Software Development
Inheritance and Polymorphism

Yu Wang

wangy2@gonzaga.edu

Feb 14, 2025

Announcement

- Homework1 due day: Feb.14th
- Homework2 due day: Feb 28th
- **Exam Day: Feb 28th**
- Homework0 due day: March 3rd

Daily Attendance (01)

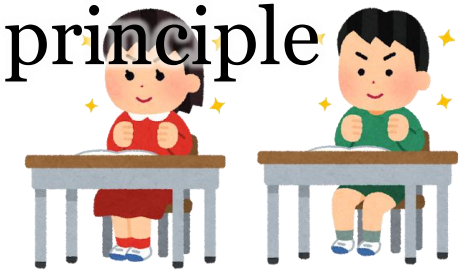
- On Paper

Daily Attendance (02)

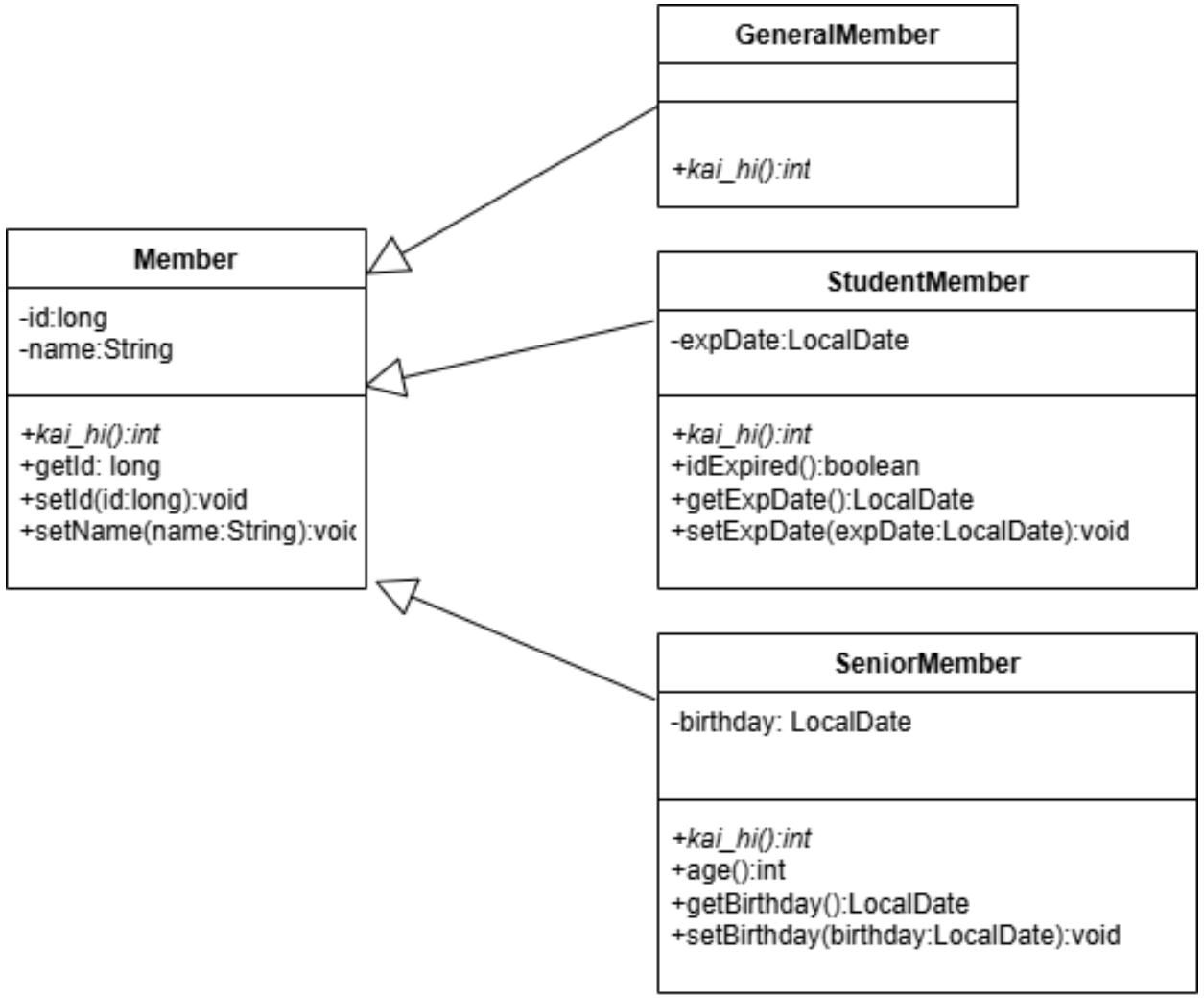
- Scan the QR Code for yourself

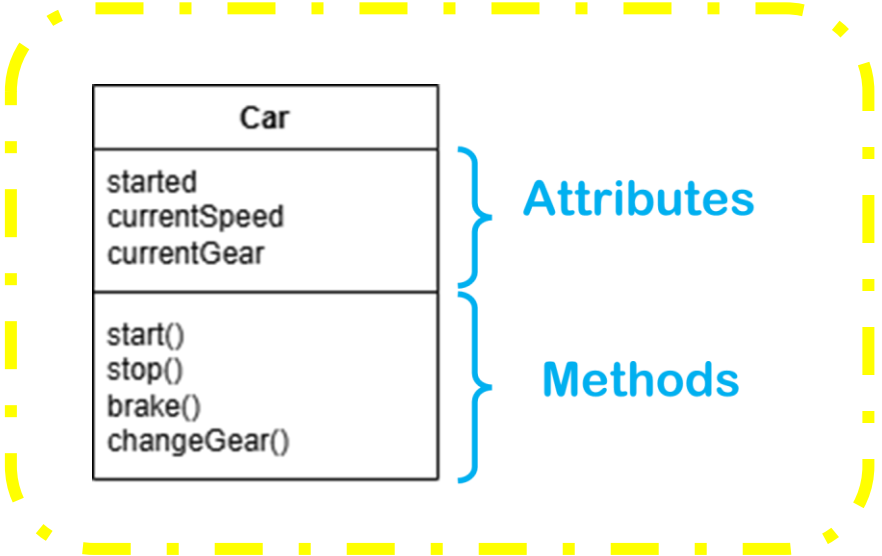
Review - Last Class

- ✓ We learned – What is Abstraction? Abstraction focuses on hiding the implementation details and showing only the necessary features
- ✓ We learned – The difference between abstraction and encapsulation. Encapsulation focuses on bundling data and methods that operate on the data into a single unit and restricting access to some of the object's components.
- ✓ We learned – What is inheritance? Inheritance is the principle that allows classes to derive from other classes.



Inheritance and Abstraction





Inheritance – override

In Java, the @Override annotation is used to indicate that a method in a subclass is intended to override a method in its superclass.

This annotation helps ensure that the method signature in the subclass matches the method signature in the superclass, providing compile-time checking to prevent errors.

Override – Code Example

GONZAGA UNIVERSITY

```
class Animal {
    void makeSound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    @Override
    void makeSound() {
        System.out.println("Dog barks");
    }
}
```

Polymorphism

- This section we will be looking at the last of the four main principles of object-oriented programming:
  - Encapsulation
  - Abstraction
  - Inheritance
  - **Polymorphism**

Polymorphism

Polymorphism describes methods that are able to take on many forms

There are two types of polymorphism

The first type is known as dynamic polymorphism

Polymorphism - Dynamic

Dynamic polymorphism occurs during the runtime of the program

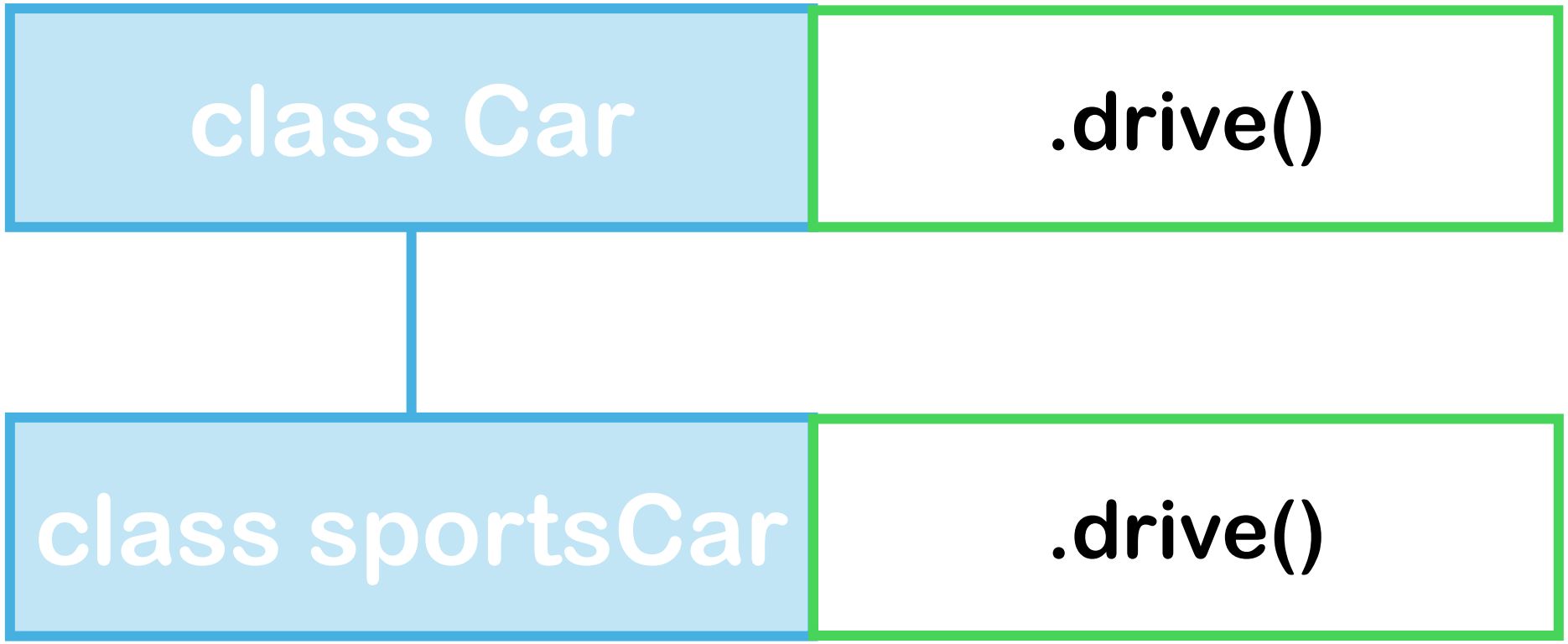
This type of polymorphism describes when a method signature is in both a subclass and a superclass

Polymorphism - Dynamic

The methods share the **same name** but have **different implementation**

The implementation of the subclass that the object is an instance of overrides that of the superclass

Polymorphism - Example



Polymorphism - Example

class Car
.drive(miles)
{Car.gas -= 0.04*miles}

class sportsCar
.drive(miles)
{Car.gas -= 0.02*miles}

Polymorphism - Example

GONZAGA UNIVERSITY

class Car

.drive(miles)

class sportsCar

.drive(miles)

mySportsCar.drive()

Polymorphism - Example

GONZAGA UNIVERSITY

class Car

.drive(miles)

class sportsCar

.drive(miles)

myCar.drive()

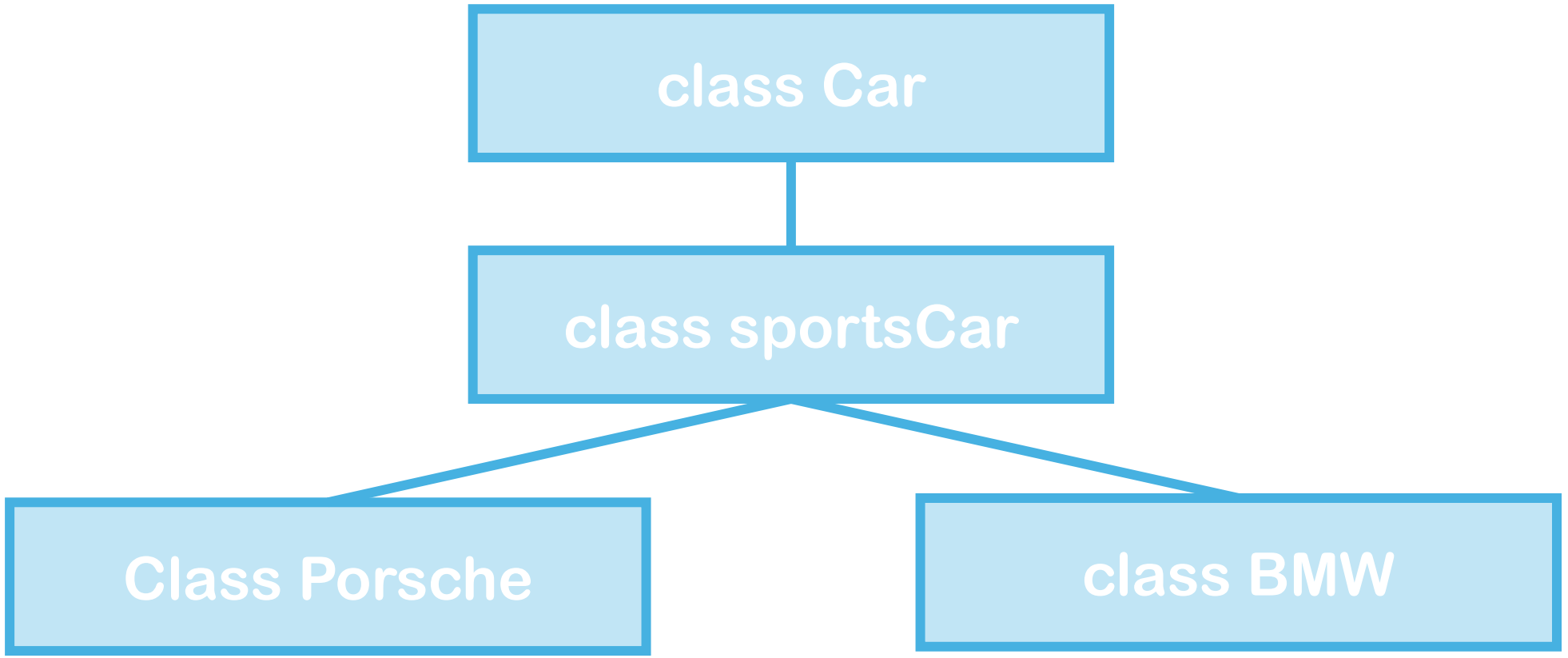
Polymorphism - Dynamic

GONZAGA UNIVERSITY

This works because the form of the method is decided based on where in the class hierarchy it is called.

The implementation of a method signature that will be used is determined dynamically as the program is run.

Polymorphism - Example



Polymorphism - Dynamic

GONZAGA UNIVERSITY

The main benefit of dynamic polymorphism is that it allows you to write methods in the super class without having to include ifs and else ifs to account for exactly which subclass is being used when the method is called.

Polymorphism - Static

GONZAGA UNIVERSITY

Static polymorphism occurs during compile-time rather than during runtime

This refers to when multiple methods with the same name but different arguments are defined in the same class

Method Overloading

GONZAGA UNIVERSITY

Ways to differentiate methods of the same name

- Different number of parameters
- Different types of parameters
- Different order of parameters

Method Overloading

GONZAGA UNIVERSITY

This is known as **method overloading**

Despite the methods having the same name, their signatures are different due to their different arguments

Polymorphism - Example

class Car	
1	.drive(int spd, string dest)
2	.drive(int spd, int dest)
3	.drive(string dest, int spd)

Polymorphism - Example

class Car	
1	.drive(int spd, string dest)
2	.drive(int spd, int dest)
3	.drive(string dest, int spd)

myCar.drive(45, “work”)

Polymorphism - Example

class Car	
1	.drive(int spd, string dest)
2	.drive(int spd, int dest)
3	.drive(string dest, int spd)

myCar.drive(15, 60)

Polymorphism - Example

class Car	
1	.drive(int spd, string dest)
2	.drive(int spd, int dest)
3	.drive(string dest, int spd)

myCar.drive(“School, 30)

Polymorphism - Example

class Car	
1	.drive(int spd, string dest)
2	.drive(int spd, int dest)
3	.drive(string dest, int spd)

Polymorphism - Static

Keep in mind that method overloading can cause trouble if you do not keep straight which parameters you need for which implementation

Using the incorrect argument may not cause an error if it matches that of another form of the method, which can cause issues

Polymorphism - Overview

GONZAGA UNIVERSITY

Overall, polymorphism allows methods to take on many different forms

When utilizing polymorphism and method overloading, be sure that you are calling the correct form of the method

Any Questions?