

Object – Oriented Programming

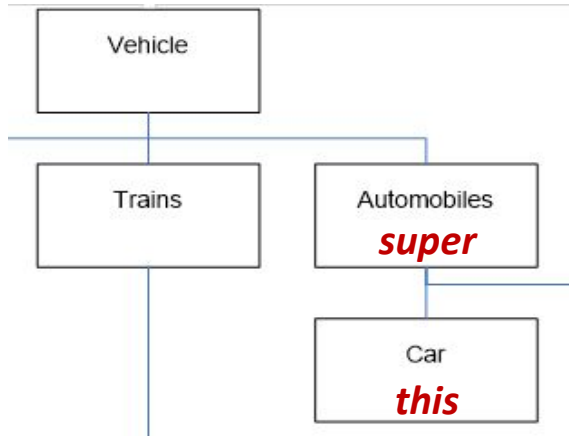
LAB #8. Polymorphism and Abstract Classes + UML

Polymorphism

- late binding 메커니즘을 통해 하나의 메소드 이름에 많은 의미를 연결 하는 기능
- late binding 혹은 dynamic binding 이라고 알려진 특별한 메커니즘을 통해 이루어짐

Polymorphism

- Polymorphism은 상속된 클래스의 메소드 정의를 재정의(Override)하며, 이러한 변경 사항을 base class (부모 클래스) 용으로 작성된 소프트웨어에 적용할 수 있게 함



```
public class Automobile{
    ---
    public String toString(){
        ---
    }
}

public class Car{
    ---
    public String toString(){
        ---
    }
}
```

이름이 똑같네...

?

Some auto

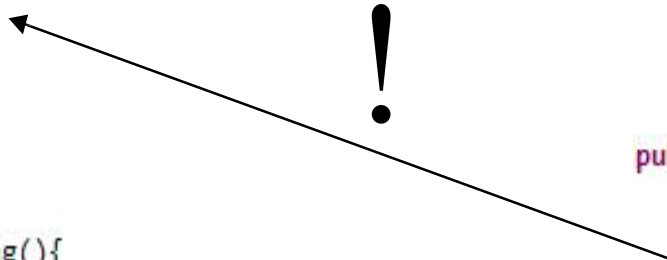
Binding

- 메소드 호출과 메소드 정의를 연결하는 과정

```
public class Automobile{  
    ---  
    public String toString(){  
        ---  
    }  
}  
  
public class Car{  
    ---  
    public String toString(){  
        ---  
    }  
}
```

!

```
public static void main (String[] args){  
    Automobile auto = new Automobile();  
    System.out.println(auto.toString());  
}
```

A diagram illustrating the binding process. A large black exclamation mark is positioned between the two code blocks. An arrow originates from the `auto.toString()` call within the `main` method of the right-hand code block and points to the `public String toString()` definition within the `Automobile` class of the left-hand code block.

Binding

- Early binding (static binding)
 - 코드가 컴파일 될 때, 메소드 정의가 메소드 호출과 연결됨
 - Late binding (dynamic binding)
 - 메소드가 런타임에 호출될 때, 메소드 정의가 메소드 호출과 연결됨
 - Java는 모든 메소드에 대하여 late binding을 사용한다. (final과 static은 예외)
-
- Compile time : 코드 작성시
 - Run time : 실행시
 - new 연산자를 통해 객체 생성 후 객체를 통해 호출되는 메소드들은 런타임에서 메소드 정의와 바인딩 되지만 static과 final의 경우 객체 생성없이 컴파일 타임에 바인딩

No Late Binding for Static Methods

- 컴파일 타임에서 사용할 메소드의 정의를 결정할 때, 이를 static binding 혹은 early binding 이라 한다.
 - “객체를 이름 지은” 변수의 타입에 기반하여 결정됨 (객체의 클래스가 아닌 **변수의 클래스**)
<https://stackoverflow.com/questions/18804328/why-static-method-of-parent-class-is-called-when-subclass-has-already-overridden>
- Java는 private, final, static method에 static binding을 사용한다.
 - private, final 메소드의 경우, late binding은 의미 없음
 - private 메소드는 해당 클래스 자체에만 소속된 메소드
 - final은 상속 못함, 따라서 해당 클래스에만 소속됨

Recap: The *final* Modifier

- `final`로 선언된 메소드는 상속된 클래스에서 새로운 정의로 오버라이딩 될 수 없다.
 - ➔ `final`의 경우, 컴파일러는 해당 메소드에 early binding을 함
- `final`로 선언된 클래스는 다른 클래스로 (메소드와 변수를) 상속하는
base class(부모 클래스)로 사용될 수 없다.

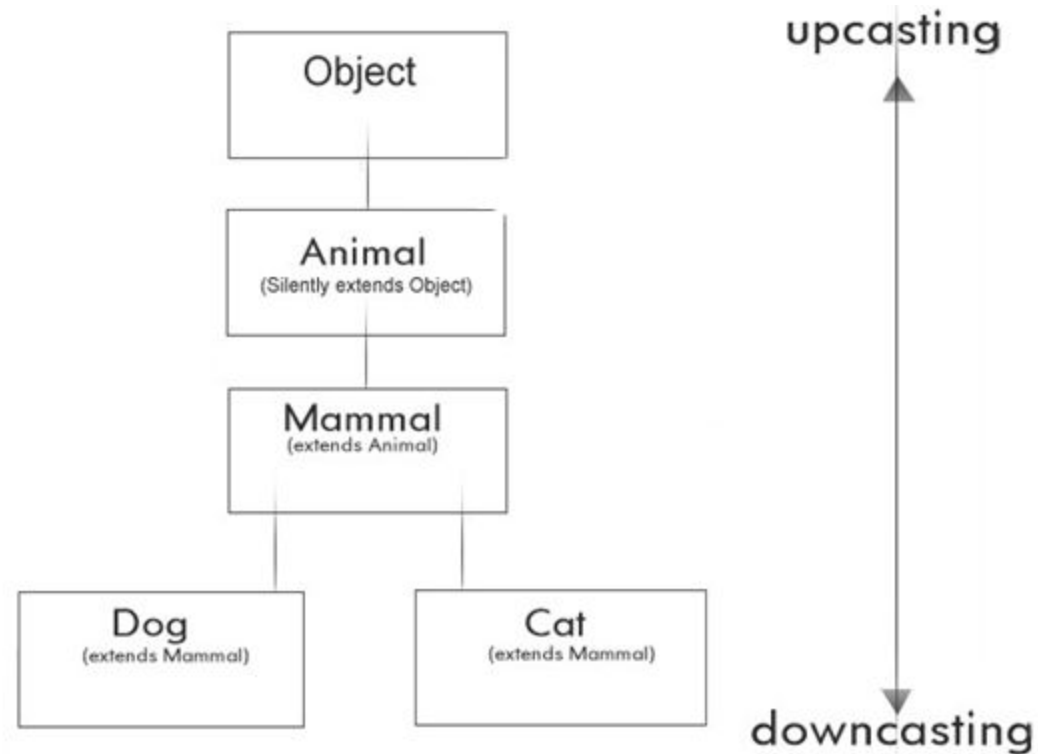
Upcasting and Downcasting

- Upcasting
 - 상속된 클래스의 객체가 parent class 혹은 ancestor class의 변수에 할당될 때
- Downcasting
 - parent class가 상속된 클래스로 타입 캐스팅이 수행될 때
 - ancestor class가 그 클래스의 하위 클래스로 타입 캐스팅이 수행될 때

Upcasting and Downcasting

```
Cat c = new Cat();  
Mammal m = c; // upcasting
```

```
Cat c1 = new Cat();  
Animal a = c1; // automatic upcasting to Animal  
Cat c2 = (Cat) a; // manual downcasting back to a Cat
```



Recap: isinstance

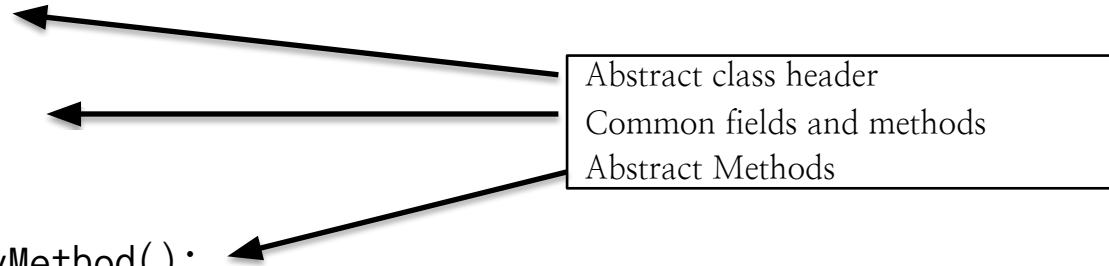
- 특정 객체가 무슨 타입인지 알려주는 연산자
 - [객체] isinstance [클래스]
 - [sub class의 객체] isinstance [super class] == true
 - [super class의 객체] isinstance [sub class] == false

Abstract Classes

- 하나 이상의 Abstract Method(추상 메소드)를 포함한 클래스
- abstract method : 완전한 정의가 없는 메소드 (단순한 placeholder)
 - ➔ 선언만 되어있고 구현부가 비어있음
- concrete class : 어떠한 abstract method도 포함하지 않는 클래스
 - ➔ 인스턴스화 가능 (객체를 만들 수 있음)

Defining Abstract Class

```
public abstract class Myclass {  
    // class constructors  
    // accessors and mutators  
    // other methods  
  
    public abstract returnType myMethod();  
}
```



When to use (일반적인 상속과 비슷)

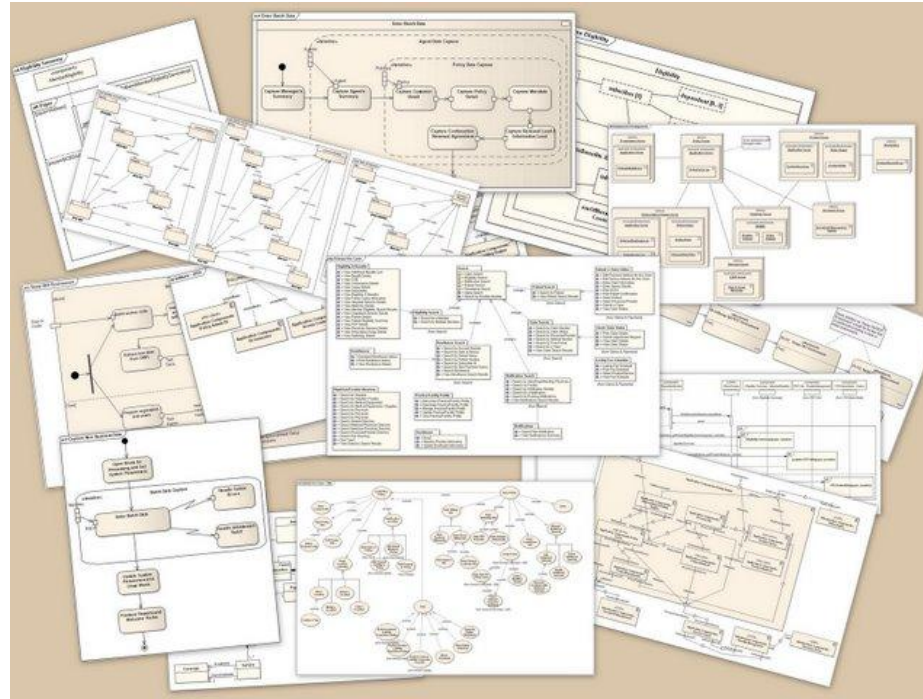
- 밀접하게 관련된 클래스들이 코드를 공유하게 만들고 싶은 경우
- Method Body가 아직 구현되지 않은 메소드를 사용하는 다른 메소드가 있는 경우, 상속받은 클래스에서 미 구현된 메소드를 구현해서 사용 (hook 메소드)
<https://mrtint.tistory.com/358>
- Abstract class를 확장하는 클래스가 여러 개의 공통 메소드 혹은 필드를 가지고 있거나 public이 아닌 다른 Access modifier를 요구할 수 있을 경우(protected)
<https://www.geeksforgeeks.org/method-overriding-with-access-modifier/>
(상속시 좀 더 넓은 범위로 변경 가능, abstract만 특정한건 아니고 모든 상속 공통 특징)

*abstract class vs interface

<https://yaboong.github.io/java/2018/09/25/interface-vs-abstract-in-java8/>

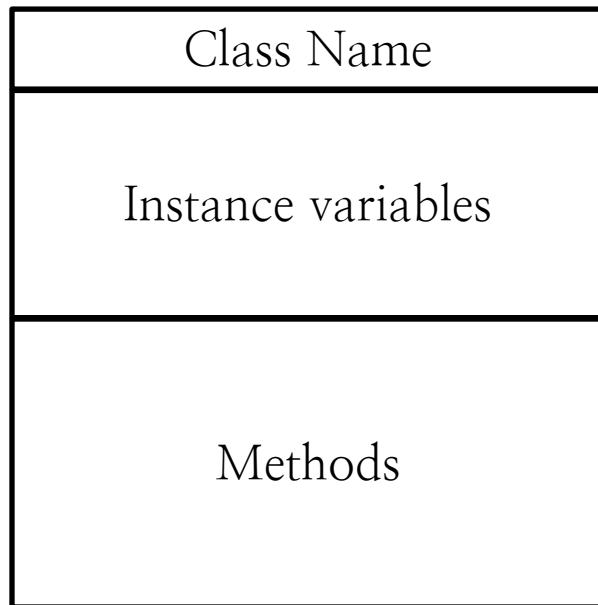
UML

- UML(Unified Modeling Language, 통합 모델링 언어)
: 객체 지향 프로그래밍 소프트웨어에서 설계, 문서화하기 위해 사용되는 그래픽 언어



UML

– Class Diagram



Instance variable:

(modifier) (variable name): (type)

ex) private double side

➔ - side: double

Method:

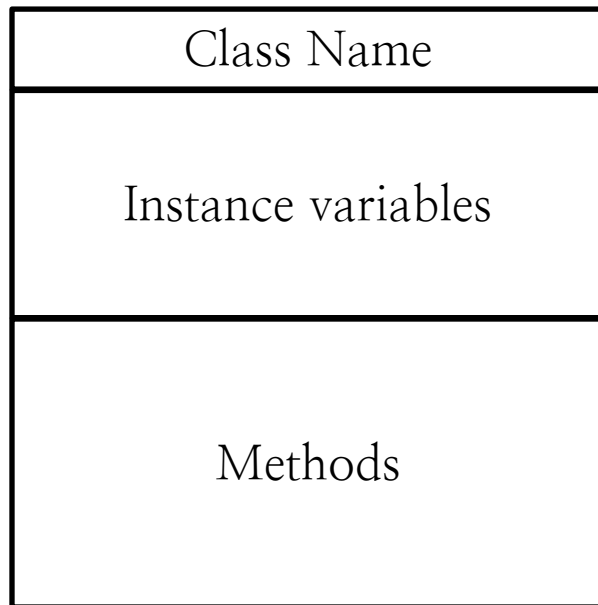
(modifier) (method name)((parameters)): (return type)

ex) public void reSize(double newSide){...}

➔ + resize(double newSide): void

UML

– Class Diagram



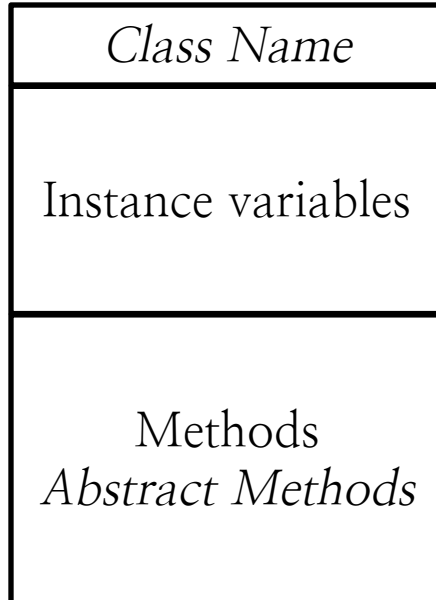
Modifier

- private: minus(−)
- public: plus(+)
- protected: sharp(#)
- package: tilde(~)
- static: underline
- final: ALL CAPITAL LETTERS

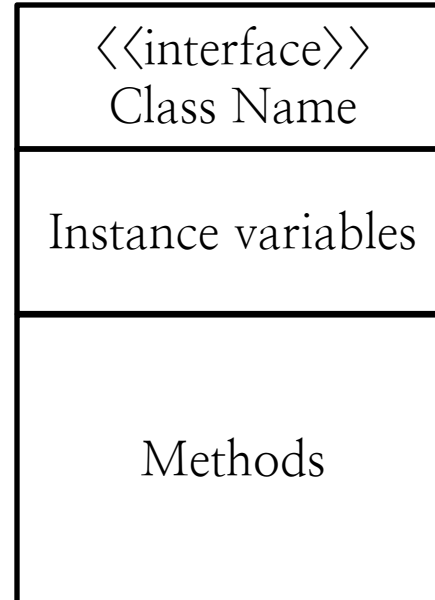
UML

– Class Diagram

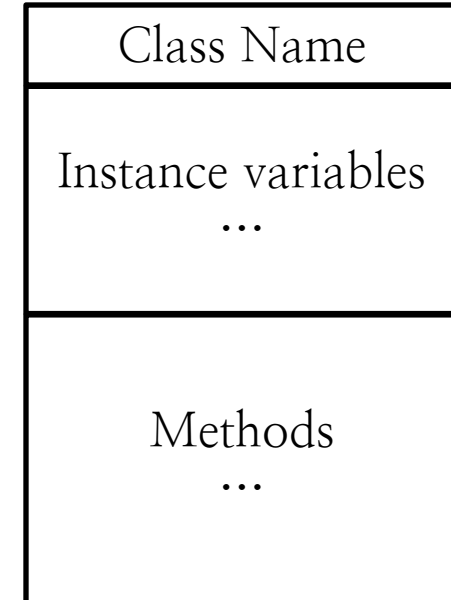
✓ Abstract: *italic*



✓ Interface: <<interface>>



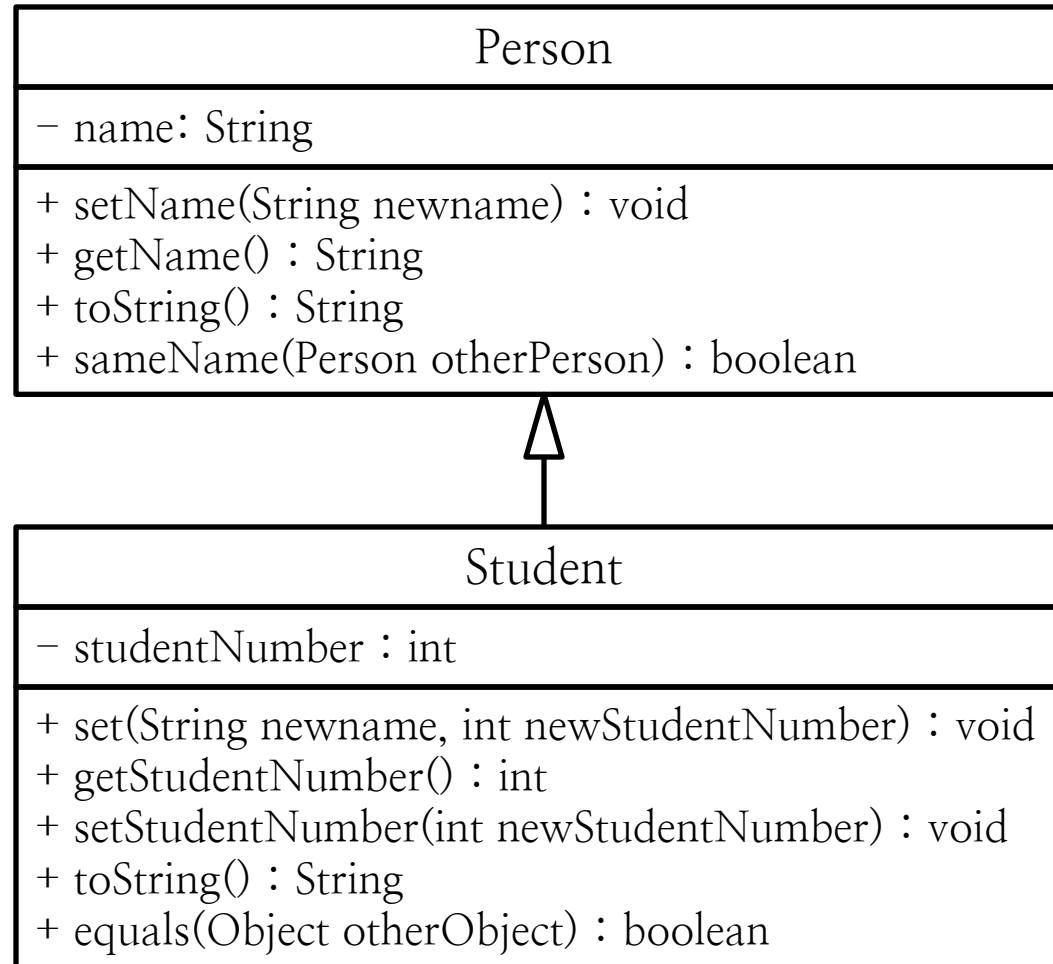
✓ Ellipsis: ...



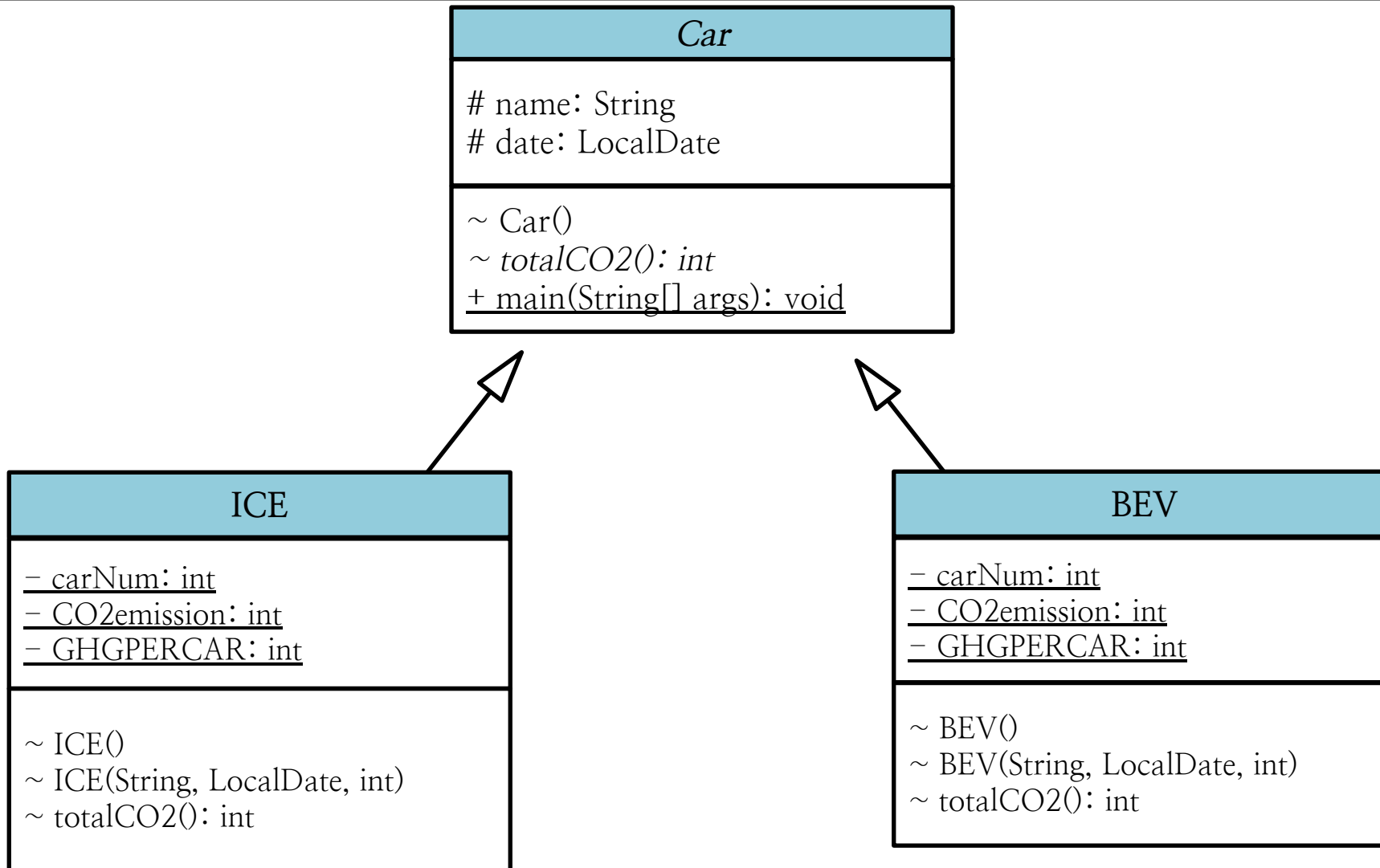
UML

– Class Diagram

✓ Inheritance



실습



실습 과제

- Car, ICE, BEV Class를 생성한다. *ICE: 내연기관차, BEV: 전기차
 - Car class는 추상클래스로 선언한다.
 - Car class 의 필드는 아래와 같이 정의한다.
protected String name
protected LocalDate date
 - 아무 값도 받지않는 기본 생성자를 만들고 초기값을 아래와 같이 설정한다.
name="Car Frame", date=null;
 - Abstract method totalCO2()을 작성한다. (return: int)
언급하지 않은 액세스 권한(Access Rights) 및 제어자(modifier)는 앞의 UML을 참고하여 작성한다.

실습 과제

- ICE class 는 Car class를 extends 한다.
 - ICE class 필드 구성
private int carNum
private int CO2emission
private int GHGPERCAR = 35
 - 아무 값도 받지 않은 기본 생성자를 작성한다.
 - String name, LocalDate date, int carNum 을 인자로 받는 생성자를 작성한다.
 - ➔ 이 생성자로 객체를 생성할 때 다음과 같이 필드 값을 바꾼다
 - name과 date는 입력받은 값으로 바꾼다.
 - static carNum 필드 변수 값에 인자로 받은 carNum을 더해준다.
 - static CO2emission 필드 변수 값에 인자로 받은 carNum과 GHGPERCAR를 곱한 값을 더해준다.
 - ICE의 name과 date가 같으면 true를 반환하는 equals(Object obj)를 작성한다.
 - name, date, carNum 을 출력하는 toString() 메소드를 작성한다.
 - totalCO2 메소드를 작성한다.
 - "ICE emit CO2 most when driving" 출력
 - CO2emission 값을 리턴한다.

실습 과제

- BEV class 는 Car class를 extends 한다.
 - BEV class 필드 구성
private int carNum
private int CO2emission
private int GHGPERCAR = 25
 - 아무 값도 받지 않은 기본 생성자를 작성한다.
 - String name, LocalDate date, int carNum 을 인자로 받는 생성자를 작성한다.
 - ➔ 이 생성자로 객체를 생성할 때 다음과 같이 필드 값을 바꾼다
 - name과 date는 입력받은 값으로 바꾼다.
 - static carNum 필드 변수 값에 인자로 받은 carNum을 더해준다.
 - static CO2emission 필드 변수 값에 인자로 받은 carNum과 GHGPERCAR를 곱한 값을 더해준다.
 - BEV의 name과 date가 같으면 true를 반환하는 equals(Object obj)를 작성한다.
 - name, date, carNum 을 출력하는 toString() 메소드를 작성한다.
 - totalCO2 메소드를 작성한다.
 - "BEV emit CO2 most when generating electric energy" 출력
 - CO2emission 값을 리턴한다.

main method

```
public class Car {  
    ...  
    public static void main(String[] args) {  
        Car protoICE = new ICE();  
        Car protoBEV = new BEV();  
        System.out.println(protoICE);  
        System.out.println(protoBEV);  
  
        ICE protoType1 = new ICE("Test1", LocalDate.of(1886, 01, 29), 1); System.out.println(protoType1);  
        ICE newICE = new ICE("ICE1", LocalDate.now(), 800000); System.out.println(newICE);  
        ICE addICE = new ICE("ICE1", LocalDate.now(), 200000); System.out.println(addICE);  
  
        System.out.println(protoType1.equals(newICE));  
        System.out.println(newICE.equals(addICE));  
  
        BEV protoType2 = new BEV("Test2", LocalDate.of(1832, 01, 01), 1); System.out.println(protoType2);  
        BEV newBEV = new BEV("BEV1", LocalDate.now(), 1000000); System.out.println(newBEV);  
        BEV addBEV = new BEV("BEV1", LocalDate.now(), 300000); System.out.println(addBEV);  
        BEV BEVplusplus = new BEV("BEV1++", LocalDate.now(), 100000); System.out.println(BEVplusplus);  
  
        System.out.println(newBEV.equals(addBEV));  
        System.out.println(addBEV.equals(BEVplusplus));  
  
        System.out.println(protoICE.totalCO2());  
        System.out.println(protoBEV.totalCO2());  
    }  
}
```

실습 과제

- Car.java, ICE.java, BEV.java 를 제출

```
name: Car Frame, date: null, carNum: 0
name: Car Frame, date: null, carNum: 0
name: Test1, date: 1886-01-29, carNum: 1
name: ICE1, date: 2023-04-19, carNum: 800001
name: ICE1, date: 2023-04-19, carNum: 1000001
false
true
name: Test2, date: 1832-01-01, carNum: 1
name: BEV1, date: 2023-04-19, carNum: 1000001
name: BEV1, date: 2023-04-19, carNum: 1300001
name: BEV1++, date: 2023-04-19, carNum: 1400001
true
false
ICE emit CO2 most when driving
35000035
BEV emit CO2 most when generating electric energy
35000025
```