Luke's Language로 파싱된 이펙티브 자바

제네릭 혁명

제작. 홍성혁

Item 28 - 컬렉션과 제네릭

공변성과 반공변성 (컴퓨터 과학)

10개 언어

문서 토론 읽기 편집 역사 보기 도구

위키백과, 우리 모두의 백과사전.

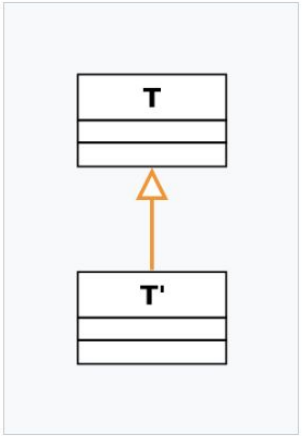
프로그래밍 언어의 **공변성**(영어: Covariance)과 **반공변성**(영어: Contravariance)은 프로그래밍 언어가 타입 생성자(영어: type constructor)에 있어 서브타입을 처리하는 방법을 나타내는 것으로, 더 복잡한 타입간의 서브타입 관계가 타입 사이의 서브타입 관계에 따라 정의되거나, 이를 배반해 정의됨을 가리킨다.

객체 지향 언어의 상속에서 [편집]

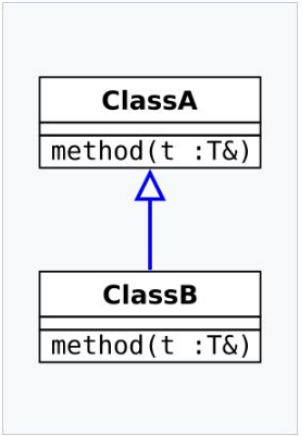
서브클래스가 슈퍼클래스의 메소드를 오버라이드 할 경우, 컴파일러는 메소드의 타입이 올바른지 확인해 타입 안전(영어: type safe)을 보장해야만 한다. 무공변만을 허용하는 일부 언어에서는 인자 타입과 반환 타입 모두 슈퍼클래스의 인자 타입 그리고 반환 타입과 정확히 같아야 한다. 그러나 특별한 규칙을 따른다면, 오버라이드된 메소드가 더 유연한 타입을 갖도록 허용하면서도 타입 안전을 보장할 수 있다. 함수 유형에 대한 일반적인 서브타이핑 규칙에 따르면, 오버라이드된 메소드는 더욱 구체적인 타입을 반환해야 하고(반환 타입 공변), 더욱 일반적인 타입을 허용해야만 한다.(인자 타입 반공변)

Item 28 - 컬렉션과 제네릭

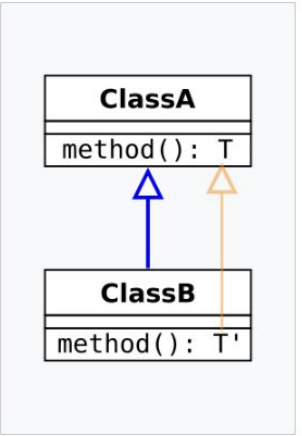
변성과 메소드 오버라이딩: 개요



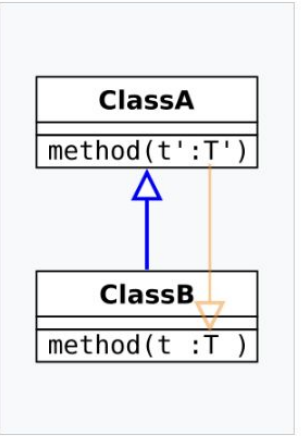
함수의 인자/반환 타입과 서브타이핑



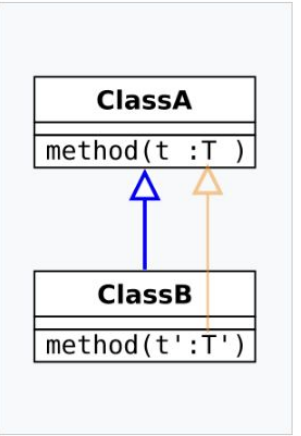
무공변. 오버라이딩된 메소드의 시그니처는 변하지 않았다. -- 타입 안전하다.



공변 반환 타입. 서브타이핑 관계는 CalssA와 ClassB 사이의 관계와 같은 방향이다. -- 타입 안전하다.



반공변 인자 타입. 서브타이핑 관계는 ClassA 그리고 ClassB 사이의 관계와 반대 방향이다. -- 타입 안전하다.



공변 인자 타입. -- 이 경우에는 타입 안전하지 않다.

Item 28 - 컬렉션과 제네릭

코드 28-1 런타임에 실패한다.

```
Object[] objectArray = new Long[1];
objectArray[0] = "타입이 달라 넣을 수 없다."; // ArrayStoreException을 던진다.
```

코드 28-2 컴파일되지 않는다!

```
List<Object> ol = new ArrayList<Long>(); // 호환되지 않는 타입이다.
ol.add("타입이 달라 넣을 수 없다.");
```

Item 28 - 컬렉션과 제네릭

코드 28-1 런타임에 실패한다.

```
Object[] objectArray = new Long[1];
objectArray[0] = "타입이 달라 넣을 수 없다."; // ArrayStoreException을 던진다.
```

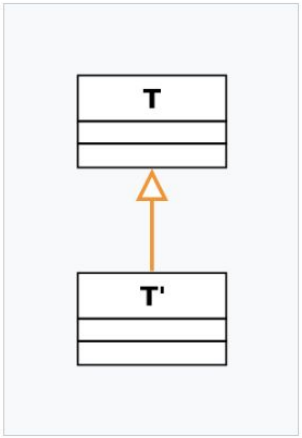
=> *공변 인자 구조*

코드 28-2 컴파일되지 않는다!

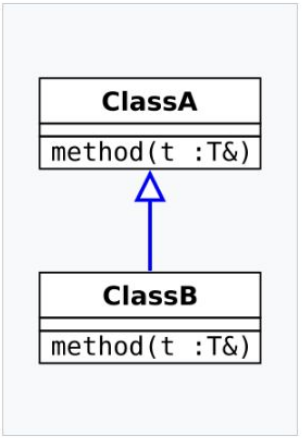
```
List<Object> ol = new ArrayList<Long>(); // 호환되지 않는 타입이다.
ol.add("타입이 달라 넣을 수 없다.");
```

Item 28 - 컬렉션과 제네릭

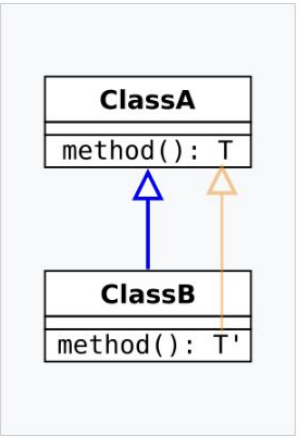
변성과 메소드 오버라이딩: 개요



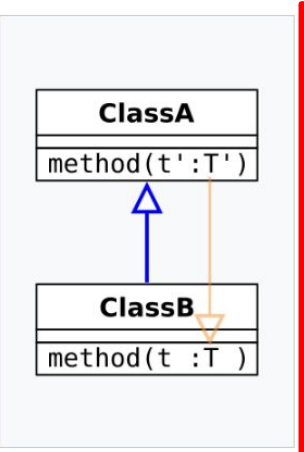
함수의 인자/반환 타입과 서브타이핑



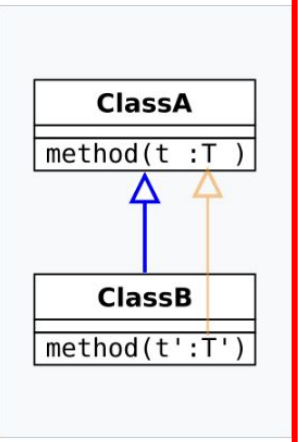
무공변. 오버라이딩된 메소드의 시그니처는 변하지 않았다. -- 타입 안전하다.



공변 반환 타입. 서브타이핑 관계는 CalssA와 ClassB 사이의 관계와 같은 방향이다. -- 타입 안전하다.



반공변 인자 타입. 서브타이핑 관계는 ClassA 그리고 ClassB 사이의 관계와 반대 방향이다. -- 타입 안전하다.



공변 인자 타입. -- 이 경우에는 타입 안전하지 않다.

Item 28 - 컬렉션과 제네릭

코드 28-1 런타임에 실패한다.

```
Object[] objectArray = new Long[1];
objectArray[0] = "타입이 달라 넣을 수 없다."; // ArrayStoreException을 던진다.
```

코드 28-2 컴파일되지 않는다!

```
List<Object> ol = new ArrayList<Long>(); // 호환되지 않는 타입이다.
ol.add("타입이 달라 넣을 수 없다.");
```

=> 복잡한 공변 구조와 상관 없이, 제네릭에 의한 판단!
불공변!

Item 29 - 클래스에 제네릭 써먹기

코드 29-1 **Object** 기반 스택 - 제네릭이 절실한 강력 후보!

```
public class Stack {
    private Object[] elements;
    private int size = 0;
    private static final int DEFAULT_INITIAL_CAPACITY = 16;

    public Stack() {
        elements = new Object[DEFAULT_INITIAL_CAPACITY];
    }

    public void push(Object e) {
        ensureCapacity();
        elements[size++] = e;
    }

    public Object pop() {
        if (size == 0)
            throw new EmptyStackException();
        Object result = elements[--size];
        elements[size] = null; // 다 쓴 참조 해제
        return result;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    private void ensureCapacity() {
        if (elements.length == size)
            elements = Arrays.copyOf(elements, 2 * size + 1);
    }
}
```

Item 29 - 클래스에 제네릭 써먹기

Object를 제네릭으로 변경해보자

코드 29-1 **Object** 기반 스택 - 제네릭이 절실한 강력 후보!

```
public class Stack {
    private Object[] elements;
    private int size = 0;
    private static final int DEFAULT_INITIAL_CAPACITY = 16;

    public Stack() {
        elements = new Object[DEFAULT_INITIAL_CAPACITY];
    }

    public void push(Object e) {
        ensureCapacity();
        elements[size++] = e;
    }

    public Object pop() {
        if (size == 0)
            throw new EmptyStackException();
        Object result = elements[--size];
        elements[size] = null; // 다 쓴 참조 해제
        return result;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    private void ensureCapacity() {
        if (elements.length == size)
            elements = Arrays.copyOf(elements, 2 * size + 1);
    }
}
```

Item 29 - 클래스에 제네릭 써먹기

Object를 제네릭으로 변경해보자

코드 29-1 **Object** 기반 스택 - 제네릭이 절실한 강력 후보!

```
public class Stack<E> {
    private E[] elements;
    private int size = 0;
    private static final int DEFAULT_INITIAL_CAPACITY = 16;

    public Stack() {
        elements = new E[DEFAULT_INITIAL_CAPACITY];
    }

    public void push(E e) {
        ensureCapacity();
        elements[size++] = e;
    }

    public E pop() {
        if (size == 0)
            throw new EmptyStackException();
        E result = elements[--size];
        elements[size] = null; // 다 쓴 참조 해제
        return result;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    private void ensureCapacity() {
        if (elements.length == size)
            elements = Arrays.copyOf(elements, 2 * size + 1);
    }
}
```

Item 29 - 클래스에 제네릭 써먹기

Object를 제네릭으로 변경해!

코드 29-1 **Object** 기반 스택 - 제네릭이 절실한 강력 후보!

```
public class Stack<E> {
    private E[] elements;
    private int size = 0;
    private static final int DEFAULT_INITIAL_CAPACITY = 16;

    public Stack() {
        elements = new E[DEFAULT_INITIAL_CAPACITY];
    }
```

```
Stack.java:8: generic array creation
        elements = new E[DEFAULT_INITIAL_CAPACITY];
                   ^
```

=> *실체 불가 타입으론 배열을 만들 수 없으므로!*

```
        E result = elements[--size];
        elements[size] = null; // 다 쓴 참조 해제
        return result;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    private void ensureCapacity() {
        if (elements.length == size)
            elements = Arrays.copyOf(elements, 2 * size + 1);
    }
}
```

Item 29 - 클래스에 제네릭 써먹기

해결책 1

생성자 시, 필드 캐스팅

코드 29-1 **Object** 기반 스택 - 제네릭이 절실한 강력 후보!

```
public class Stack<E> {
    private E[] elements;
    private int size = 0;
    private static final int DEFAULT_INITIAL_CAPACITY = 16;

    // 배열 elements는 push(E)로 넘어온 E 인스턴스만 담는다.
    // 따라서 타입 안전성을 보장하지만,
    // 이 배열의 런타임 타입은 E[]가 아닌 Object[]다!
    @SuppressWarnings("unchecked")
    public Stack() {
        elements = (E[]) new Object[DEFAULT_INITIAL_CAPACITY];
    }

    public void push(E e) {
        ensureCapacity();
        elements[size++] = e;
    }

    public E pop() {
        if (size == 0)
            throw new EmptyStackException();
        E result = elements[--size];
        elements[size] = null; // 다 쓴 참조 해제
        return result;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    private void ensureCapacity() {
        if (elements.length == size)
            elements = Arrays.copyOf(elements, 2 * size + 1);
    }
}
```

Item 29 - 클래스에 제네릭 써먹기

해결책 1

생성자 시, 필드 캐스팅

코드 29-1 Object 기반 스택 - 제네릭이 절실한 강력 후보!

```
public class Stack<E> {
    private E[] elements;
    private int size = 0;
    private static final int DEFAULT_INITIAL_CAPACITY = 16;

    // 배열 elements는 push(E)로 넘어온 E 인스턴스만 담는다.
    // 따라서 타입 안전성을 보장하지만,
    // 이 배열의 런타임 타입은 E[]가 아닌 Object[]다!
    @SuppressWarnings("unchecked")
    public Stack() {
        elements = (E[]) new Object[DEFAULT_INITIAL_CAPACITY];
    }

    public void push(E e) {
        ensureCapacity();
        elements[size++] = e;
    }

    public boolean
        return size
    }

    private void ensureCapacity() {
        if (elements.length == size)
            elements = Arrays.copyOf(elements, 2 * size + 1);
    }
}
```

```
Stack.java:8: warning: [unchecked] unchecked cast
found: Object[], required: E[]
        elements = (E[]) new Object[DEFAULT_INITIAL_CAPACITY];
                   ^
```

=> *타입 안전하지 않다는 경고*

Item 29 - 클래스에 제네릭 써먹기

해결책 1

생성자 시, 필드 캐스팅

코드 29-1 **Object** 기반 스택 - 제네릭이 절실한 강력 후보!

```
public class Stack<E> {
    private E[] elements;
    private int size = 0;
    private static final int DEFAULT_INITIAL_CAPACITY = 16;

    // 배열 elements는 push(E)로 넘어온 E 인스턴스만 담는다.
    // 따라서 타입 안전성을 보장하지만,
    // 이 배열의 런타임 타입은 E[]가 아닌 Object[]다!
    @SuppressWarnings("unchecked")
    public Stack() {
        elements = (E[]) new Object[DEFAULT_INITIAL_CAPACITY];
    }

    public void push(E e) {
        ensureCapacity();
        e
    }

    public E pop() {
        if (size == 0)
            throw new EmptyStackException();
        E result = elements[--size];
        elements[size] = null; // 다 쓴 참조 해제
        return result;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    private void ensureCapacity() {
        if (elements.length == size)
            elements = Arrays.copyOf(elements, 2 * size + 1);
    }
}
```

=> 다만 우리는 이것이 문제 없다는 것을 안

=> 경고 안뜨게 무시하자

Item 29 - 클래스에 제네릭 써먹기

해결책 2

꺼낼때, 캐스팅

코드 29-1 Object 기반 스택 - 제네릭이 절실한 강력 후보!

```
public class Stack<E> {
    private Object[] elements;
    private int size = 0;
    private static final int DEFAULT_INITIAL_CAPACITY = 16;

    public Stack() {
        elements = new Object[DEFAULT_INITIAL_CAPACITY];
    }

    public void push(E e) {
        ensureCapacity();
        elements[size++] = e;
    }

    // 비검사 경고를 적절히 숨긴다
    public E pop() {
        if (size == 0)
            throw new EmptyStackException();

        // push에서 E 타입만 허용하므로 이 형변환은 안전하다.
        @SuppressWarnings("unchecked") E result = (E) elements[--size];

        elements[size] = null; // 다 쓴 참조 해제
        return result;
    }

    private void ensureCapacity() {
        if (elements.length == size)
            elements = Arrays.copyOf(elements, 2 * size + 1);
    }
}

            elements = Arrays.copyOf(elements, 2 * size + 1);
    }
}
```

Item 29 - 클래스에 제네릭 써먹기

해결책 2

꺼낼때, 캐스팅

코드 29-1 Object 기반 스택 - 제네릭이 절실한 강력 후보!

```
public class Stack<E> {
    private Object[] elements;
    private int size = 0;
    private static final int DEFAULT_INITIAL_CAPACITY = 16;

    public Stack() {
        elements = new Object[DEFAULT_INITIAL_CAPACITY];
    }

    public void push(E e) {
        ensureCapacity();
        elements[size++] = e;
    }

    // 비검사 경고를 적절히 숨긴다
    public E pop() {
        if (size == 0)
            throw new EmptyStackException();

        // push에서 E 타입만 허용하므로 이 형변환은 안전하다.
        @SuppressWarnings("unchecked") E result = (E) elements[--size];

        elements[size] = null; // 다 쓴 참조 해제
        return result;
    }
}
```

=> 타입 안전하지 않다는 경고

```
Stack.java:19: warning: [unchecked] unchecked cast
found: Object, required: E
        E result = (E) elements[--size];
                                ^
```

Item 29 - 클래스에 제네릭 써먹기

해결책 2

꺼낼때, 캐스팅

코드 29-1 Object 기반 스택 - 제네릭이 절실한 강력 후보!

```
public class Stack<E> {
    private Object[] elements;
    private int size = 0;
    private static final int DEFAULT_INITIAL_CAPACITY = 16;

    public Stack() {
        elements = new Object[DEFAULT_INITIAL_CAPACITY];
    }

    public void push(E e) {
        ensureCapacity();
        elements[size++] = e;
    }

    // 비검사 경고를 적절히 숨긴다
    public E pop() {
        if (size == 0)
            throw new EmptyStackException();

        // push에서 E 타입만 허용하므로 이 형변환은 안전하다.
        @SuppressWarnings("unchecked") E result = (E) elements[--size];
```

=> 다만 우리는 이또한 문제 없다는 것을 안다

=> 경고 안뜨게 무시하자

```
        private void ensureCapacity() {
            if (elements.length == size)
                elements = Arrays.copyOf(elements, 2 * size + 1);
        }
    }

                elements = Arrays.copyOf(elements, 2 * size + 1);
        }
    }
```

Item 30 - 메서드에 제네릭 써먹기 1

제네릭 메서드

- union 예제

```
public static <E> Set<E> union(Set<E> s1, Set<E> s2) {
    Set<E> result = new HashSet<>(s1);
    result.addAll(s2);
    return result;
}
```

```
public static void main(String[] args) {
    Set<String> guys = Set.of("톰", "딕", "해리");
    Set<String> stooges = Set.of("래리", "모에", "컬리");
    Set<String> aflCio = union(guys, stooges);
    System.out.println(aflCio);
}
```

Item 30 - 메서드에 제네릭 써먹기 2

제네릭 싱글턴 패턴

- 항등함수 예제

```
private static UnaryOperator<Object> IDENTITY_FN = (t) -> t;

@SuppressWarnings("unchecked")
public static <T> UnaryOperator<T> identityFunction() {
    return (UnaryOperator<T>) IDENTITY_FN;
}
```

```
public static void main(String[] args) {
    String[] strings = { "삼베", "대마", "나일론" };
    UnaryOperator<String> sameString = identityFunction();
    for (String s : strings)
        System.out.println(sameString.apply(s));

    Number[] numbers = { 1, 2.0, 3L };
    UnaryOperator<Number> sameNumber = identityFunction();
    for (Number n : numbers)
        System.out.println(sameNumber.apply(n));
}
```

Item 30 - 메서드에 제네릭 써먹기 3

재귀적 타입 한정

- max 예제

```
public interface Comparable<T> {
    int compareTo(T o);
}
```

```
public static <E extends Comparable<E>> E max(Collection<E> c) {
    if (c.isEmpty())
        throw new IllegalArgumentException("컬렉션이 비어 있습니다.");

    E result = null;
    for (E e : c)
        if (result == null || e.compareTo(result) > 0)
            result = Objects.requireNonNull(e);

    return result;
}
```

Item 30 - 메서드에 제네릭 써먹기 3

재귀적 타입 한정

- max 예제

```
public interface Comparable<T> {
    int compareTo(T o);
}
```

```
public static <E extends Comparable<E>> E max(Collection<E> c) {
    if (c.isEmpty())
        null;
    for (E e : c)
        if (result == null || e.compareTo(result) > 0)
            result = Objects.requireNonNull(e);

    return result;
}
```

*=> **E** 를 비교 가능하도록 제네릭을 통해 한정 짓는다.*

Item 30 - 메서드에 제네릭 써먹기 3

재귀적 타입 한정

- max 예제

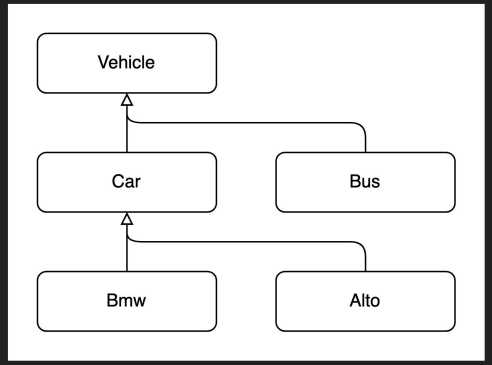
```
public interface Comparable<T> {
    int compareTo(T o);
}
```

```
public static <E extends Comparable<E>> E max(Collection<E> c) {
    if (c.isEmpty())
        throw new IllegalArgumentException("컬렉션이 비어 있습니다.");

    E result = null;
    for (E e : c)
        if (result == null || e.compareTo(result) > 0)
            result = Objects.requireNonNull(e);

    return result;
}
```

https://medium.com/@isuru89/java-producer-extends-consumer-super-9fbb0e7dd268

보충 - 제네릭 한정

객체 다형성을 이용한 제네릭 한정

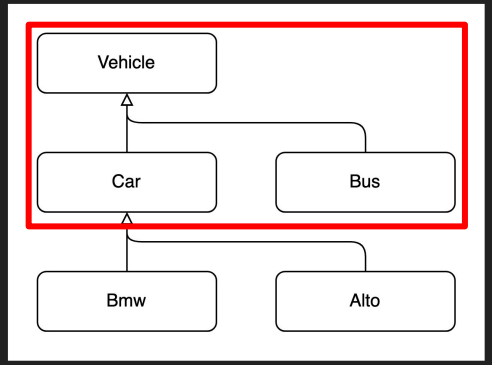



```
List<Vehicle> garage = new ArrayList<>();
garage.add(new Car());
garage.add(new Bus());

// reading behavior
Vehicle firstVehicle = garage.get(0);
```

https://medium.com/@isuru89/java-producer-extends-consumer-super-9fbb0e7dd268

보충 - 제네릭 한정

객체 다형성을 이용한 제네릭 한정

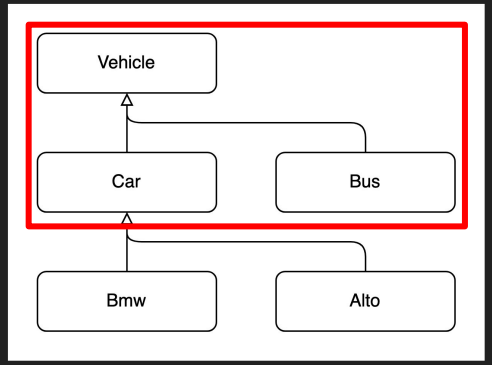



```
List<Vehicle> garage = new ArrayList<>();
garage.add(new Car());
garage.add(new Bus());

// reading behavior
Vehicle firstVehicle = garage.get(0);
```

https://medium.com/@isuru89/java-producer-extends-consumer-super-9fbb0e7dd268

보충 - 제네릭 한정

객체 다형성을 이용한 제네릭 한정

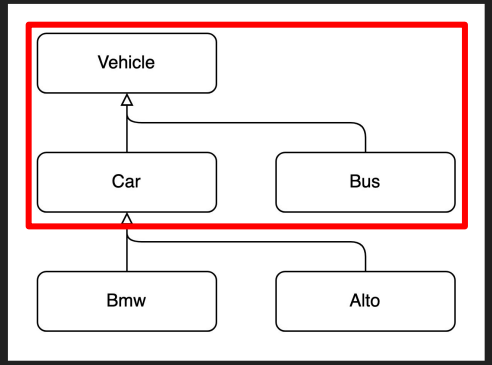



```
List<Vehicle> garage = new ArrayList<>();
garage.add(new Car());
garage.add(new Bus());

// reading behavior
Vehicle firstVehicle = garage.get(0);
```

https://medium.com/@isuru89/java-producer-extends-consumer-super-9fbb0e7dd268

보충 - 제네릭 한정

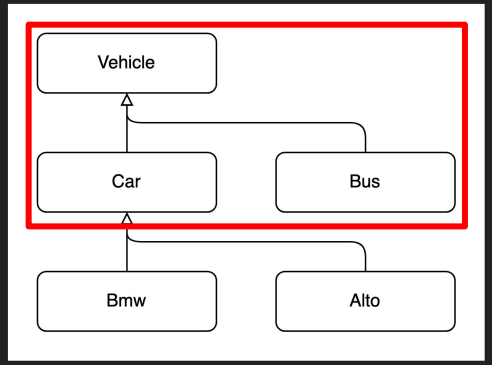
객체 다형성을 이용한 제네릭 한정




```
List<Vehicle> garage = new ArrayList<>();
garage.add(new Car());
garage.add(new Bus());

// reading behavior
Vehicle firstVehicle = garage.get(0);
```

https://medium.com/@isuru89/java-producer-extends-consumer-super-9fbb0e7dd268

보충 - 제네릭 한정



객체 다형성을 이용한 제네릭 한정

```
List<Vehicle> garage = new ArrayList<>();
garage.add(new Car());
garage.add(new Bus());

// reading behavior
Vehicle firstVehicle = garage.get(0);
```

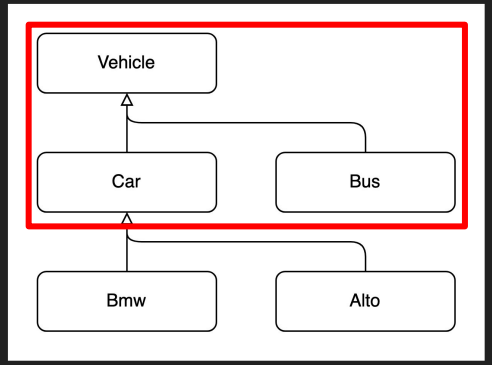
```
List<Car> cars = new ArrayList<>();
List<Vehicle> garage = cars;          // compilation error
```

=> 제네릭 자체는 불공변성

https://medium.com/@isuru89/java-producer-extends-consumer-super-9fbb0e7dd268

보충 - 제네릭 한정

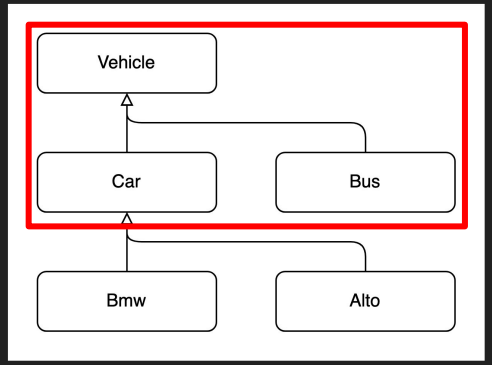
Upper bound Wildcard <? extends Vehicle>




```
List<? extends Vehicle> garage = new ArrayList<>();
garage.add(new Vehicle());  // compilation error
garage.add(new Car());      // compilation error
garage.add(new Bus());      // compilation error

// reading behavior
Vehicle vehicle = garageB.get(1);
```

https://medium.com/@isuru89/java-producer-extends-consumer-super-9fbb0e7dd268

보충 - 제네릭 한정



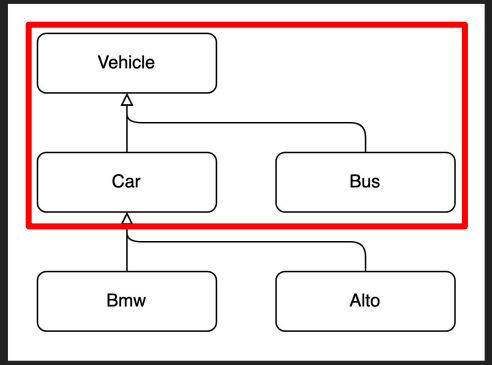
Upper bound Wildcard <? extends Vehicle>

```
List<? extends Vehicle> garage = new ArrayList<>();
garage.add(new Vehicle());  // compilation error
garage.add(new Car());      // compilation error
garage.add(new Bus());      // compilation error

// reading behavior
Vehicle vehicle = garageB.get(1);
```

? extends Vehicle 은 Vehicle의 자식 클래스

https://medium.com/@isuru89/java-producer-extends-consumer-super-9fbb0e7dd268

보충 - 제네릭 한정



Upper bound Wildcard <? extends Vehicle>

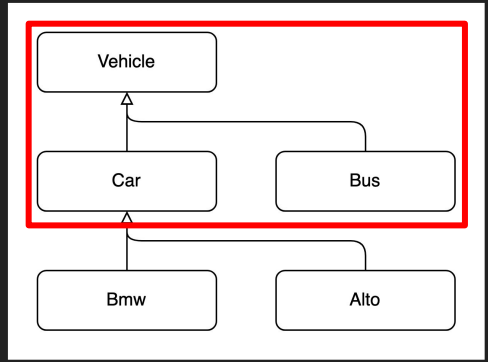
```
List<? extends Vehicle> garage = new ArrayList<>();
garage.add(new Vehicle());  // compilation error
garage.add(new Car());      // compilation error
garage.add(new Bus());      // compilation error

// reading behavior
Vehicle vehicle = garageB.get(1);
```

? extends Vehicle 은 Vehicle의 자식 클래스

? 는 뭘까? 만약 Bmw라면? Allo라면?

https://medium.com/@isuru89/java-producer-extends-consumer-super-9fbb0e7dd268

보충 - 제네릭 한정



Upper bound Wildcard <? extends Vehicle>

```
List<? extends Vehicle> garage = new ArrayList<>();
garage.add(new Vehicle());  // compilation error
garage.add(new Car());      // compilation error
garage.add(new Bus());      // compilation error

// reading behavior
Vehicle vehicle = garageB.get(1);
```

? extends Vehicle 은 Vehicle의 자식 클래스

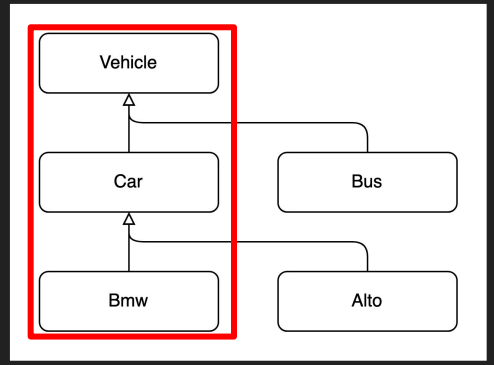
? 는 뭘까? 만약 Bmw라면? Allo라면?

? 는 특정 불가

https://medium.com/@isuru89/java-producer-extends-consumer-super-9fbb0e7dd268

보충 - 제네릭 한정

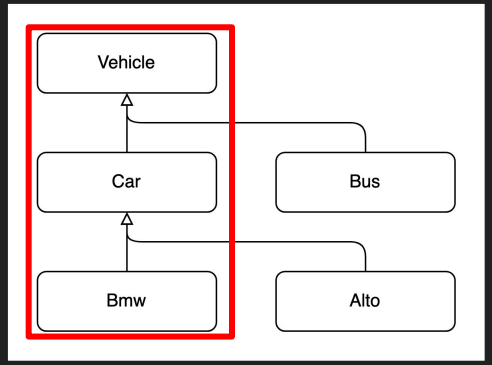
Low bound Wildcard <? super Car>




```
List<? super Car> garage = new ArrayList<>();
garage.add(new BMW());
garage.add(new Alto());
garage.add(new Vehicle());    // compilation error

// reading behavior
Object object = garage.get(0);    // I don't get a Car, why?
```

https://medium.com/@isuru89/java-producer-extends-consumer-super-9fbb0e7dd268

보충 - 제네릭 한정



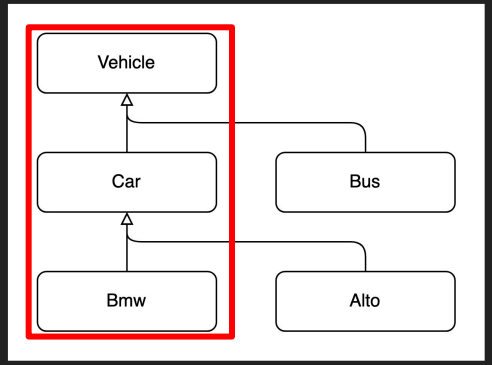
Low bound Wildcard <? super Car>

```
List<? super Car> garage = new ArrayList<>();
garage.add(new BMW());
garage.add(new Alto());
garage.add(new Vehicle());    // compilation error

// reading behavior
Object object = garage.get(0);    // I don't get a Car, why?
```

? super Car 은 Car의 부모 클래스

https://medium.com/@isuru89/java-producer-extends-consumer-super-9fbb0e7dd268

보충 - 제네릭 한정



Low bound Wildcard <? super Car>

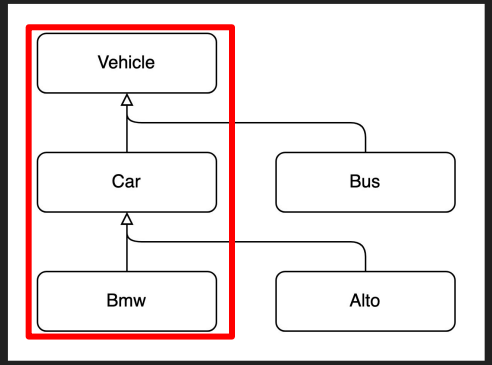
```
List<? super Car> garage = new ArrayList<>();
garage.add(new BMW());
garage.add(new Alto());
garage.add(new Vehicle());    // compilation error

// reading behavior
Object object = garage.get(0);    // I don't get a Car, why?
```

? super Car 은 Car의 부모 클래스

? 는 뭘까? Vehicle 등이 될 수 있겠다.

https://medium.com/@isuru89/java-producer-extends-consumer-super-9fbb0e7dd268

보충 - 제네릭 한정



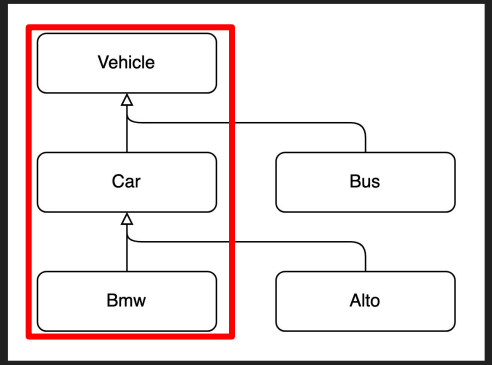
Low bound Wildcard <? super Car>

```
List<? super Car> garage = new ArrayList<>();
garage.add(new BMW());
garage.add(new Alto());
garage.add(new Vehicle());    // compilation error

// reading behavior
Object object = garage.get(0);    // I don't get a Car, why?
```

? super Car 은 Car의 부모 클래스
? 는 뭘까? Vehicle 등이 될 수 있겠다.
? 는 특정 불가한것은 마찬가지이나, Car 이하가 가능한 공변성을 가진다.

https://medium.com/@isuru89/java-producer-extends-consumer-super-9fbb0e7dd268

보충 - 제네릭 한정



Low bound Wildcard <? super Car>

```
List<? super Car> garage = new ArrayList<>();
garage.add(new BMW());
garage.add(new Alto());
garage.add(new Vehicle());    // compilation error

// reading behavior
Object object = garage.get(0);    // I don't get a Car, why?
```

? super Car 은 Car의 부모 클래스

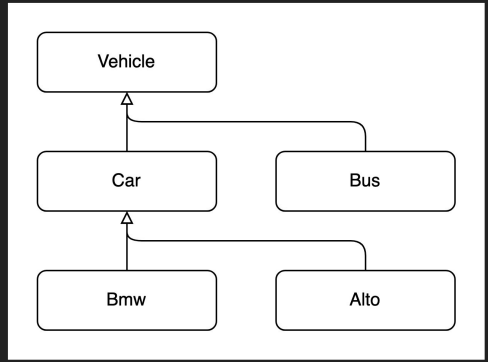
? 는 뭘까? Vehicle 등이 될 수 있겠다.

? 는 특정 불가한것은 마찬가지이나, Car 이하가 가능한 공변성을 한다.

관, Vehicle은 안됨! Car 이하가 아니므로

https://medium.com/@isuru89/java-producer-extends-consumer-super-9fbb0e7dd268

보충 - 제네릭 한정



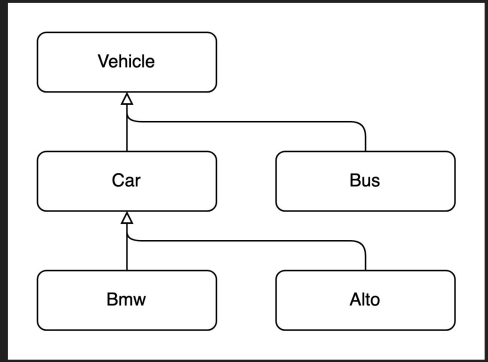
Unbounded Wildcard <?>

```
List<?> garage = new ArrayList<>();
garage.add(1);                  // compilation error
garage.add(new Object());       // compilation error
garage.add(new Car());          // compilation error

// reading behavior
Object o = list.get(0);
```

? 는 뭘까?

https://medium.com/@isuru89/java-producer-extends-consumer-super-9fbb0e7dd268

보충 - 제네릭 한정



Unbounded Wildcard <?>

```
List<?> garage = new ArrayList<>();
garage.add(1);                  // compilation error
garage.add(new Object());       // compilation error
garage.add(new Car());          // compilation error

// reading behavior
Object o = list.get(0);
```

? 는 뭘까?

뭐가 될 수 있을지 모름.. 그래서 안됨!

E.O.D.