2024학년도 1학기

신입생 Java 교육

교육부장 20 이어진

교육 커리큘럼

1주차 변수,연산자 + 조건문(if)
2주차 조건문(switch) + 반복문
3주차 배열

<중간고사>

4주차 객체, 메소드 오버로딩과 생성자
5주차 상속과 오버라이딩
6주차 다형성과 인터페이스
7주차 예외처리

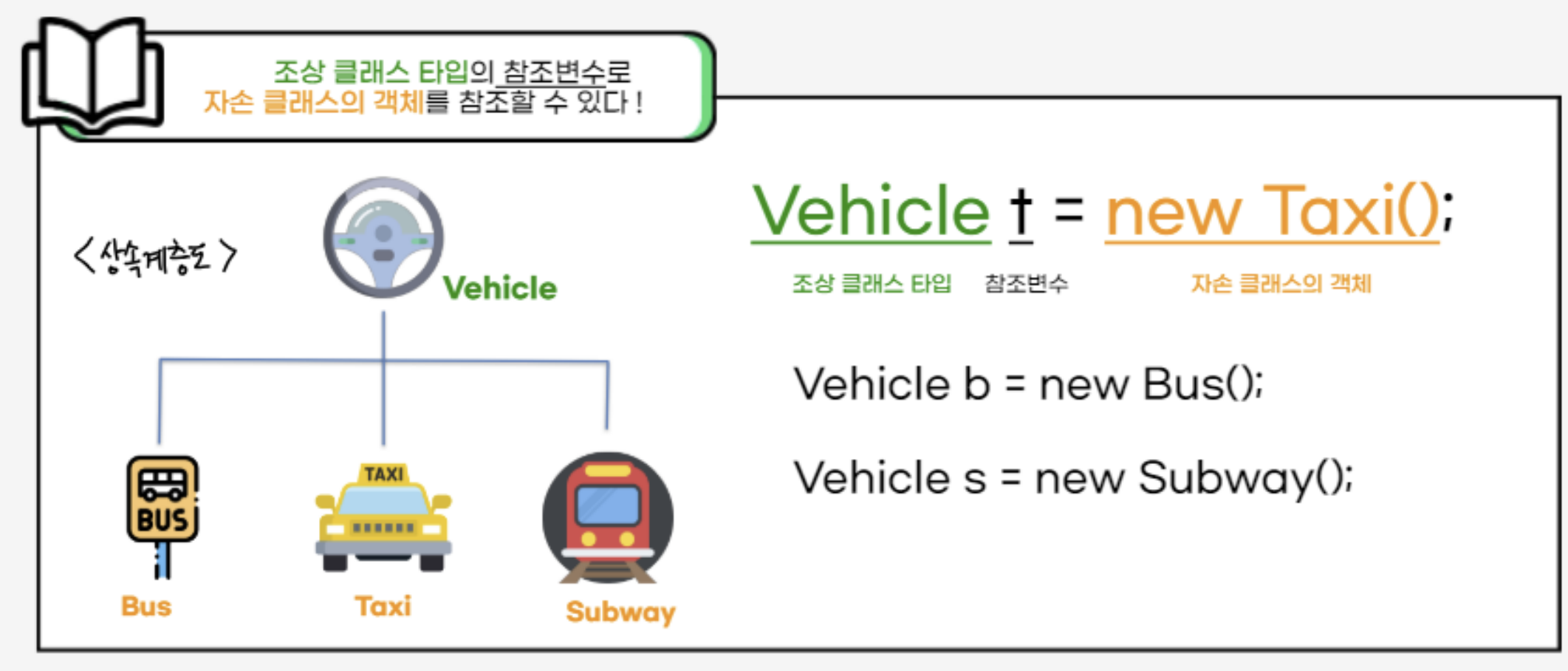
<기말고사>

Part 1

다형성

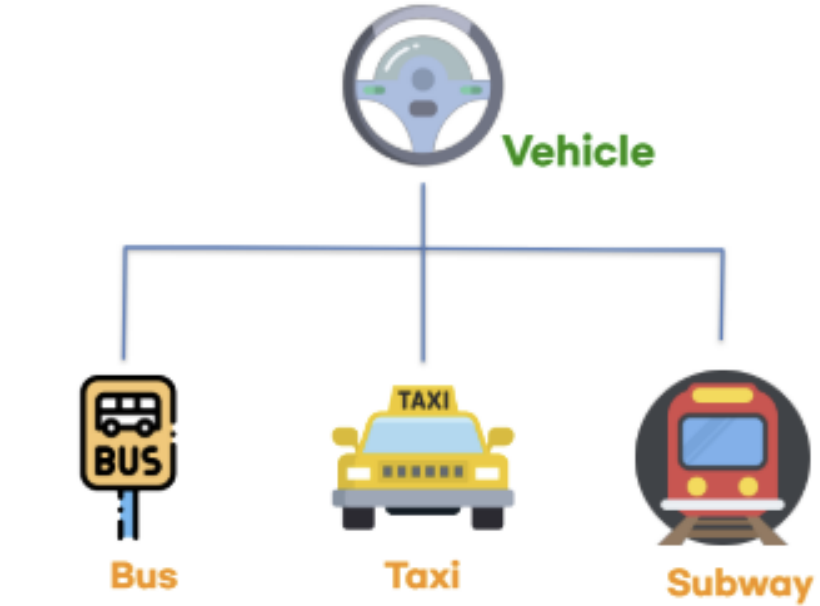
Part 1 다형성

한 타입의 참조변수로 여러 타입의 객체를 참조할 수 있는 것



Part 1 다형성 사용의 잘못된 예

자손 클래스 타입의 참조변수로
조상 클래스의 객체를 참조할 수 없다




```
Bus b = new Vehicle();

Taxi t = new Vehicle();

Subway s = new Vehicle();
```

셋 다 잘못된 예 : 자손 a = new 조상(); 형태 불가능

Part 1

instanceof 연산자

- 참조변수가 참조하는 인스턴스의 실제 타입을 체크하는데 사용
- instanceof의 연산결과가 true면, 해당 타입으로 형변환이 가능

```
class GrandParent { }

class Parent extends GrandParent { }

class Child extends Parent { }
```

```
GrandParent g = new GrandParent();
Parent p = new Parent();
Child c = new Child();
```

```
if (g instanceof GrandParent) {
    System.out.println("g는 GrandParent의 인스턴스입니다.");
}
if (g instanceof Parent) {
    System.out.println("g는 Parent의 인스턴스입니다.");
}
if (g instanceof Child) {
    System.out.println("g는 Child의 인스턴스입니다.");
}
```

```
g는 GrandParent의 인스턴스입니다.
=====================================
p는 GrandParent의 인스턴스입니다.
p는 Parent의 인스턴스입니다.
=====================================
c는 GrandParent의 인스턴스입니다.
c는 Parent의 인스턴스입니다.
c는 Child의 인스턴스입니다.
```

Part 1

참조변수와 인스턴스변수의 연결

- 멤버변수가 중복정의 된 경우, 참조변수의 타입에 따라 연결되는 멤버변수가 달라진다. (참조변수 타입에 영향 받음)

- 메소드가 중복정의 된 경우, 참조변수의 타입에 관계없이 항상 실제 인스턴스의 타입에 정의된 메소드가 호출된다. (참조변수 타입에 영향 받지 않음)

Part 1 참조변수와 인스턴스변수의 연결

- 멤버변수가 중복정의 된 경우, 참조변수의 타입에 따라 연결되는 멤버변수가 달라진다. (참조변수 타입에 영향 받음)

```
class Parent {
    int x = 10;
}

class Child extends Parent {
    int x = 20;
}
```

```
Parent p = new Parent();
Child c = new Child();

System.out.println(p.x);
System.out.println(c.x);
```

```
10
20
```

```
Parent pc = new Child();
```

참조변수 인스턴스

```
10
```

Part 1 참조변수와 인스턴스변수의 연결

- 메소드가 중복정의 된 경우, 참조변수의 타입에 관계없이 항상 실제 인스턴스의 타입에 정의된 메소드가 호출된다. (참조변수 타입에 영향 받지 않음)

```
class Parent {
    void show() {
        System.out.println("Parent");
    }
}

class Child extends Parent {
    void show() {
        System.out.println("Child");
    }
}
```

```
Parent p = new Parent();
Child c = new Child();

p.show();
c.show();
```

```
Parent
Child
```

```
Parent pc = new Child();
```

참조변수 인스턴스

```
Child
```

Part 2

추상클래스

Part 2

추상클래스(abstract class)

선언부만 있고 구현부가 없는 메서드를 포함하고 있는 클래스

클래스가 '설계도'라면, 추상클래스는 '미완성 설계도'

```
abstract class Animal {

}
```

Part 2 추상메소드(abstract method)

선언부만 있고 구현부가 없는 메소드

abstract 리턴타입 메소드이름();

```
abstract class Animal {
    abstract void sound();
}
```

꼭 필요하지만 자손마다 다르게 구현될 것으로 예상되는 경우에 사용

추상클래스를 상속받는 자식 클래스는 반드시 추상메소드를 오버라이딩하여 구현해야 함

Part 2 추상클래스 예시

```java
abstract class Animal {
    abstract void sound();
}
```

```java
class Cat extends Animal {
    @Override
    void sound() {
        System.out.println("야옹");
    }
}

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

```java
public static void main(String[] args) {
    // Animal animal = new Animal();
    // 추상 클래스는 인스턴스를 생성할 수 없다.

    Cat cat = new Cat();
    cat.sound();

    Dog dog = new Dog();
    dog.sound();
}
```

```
야옹
멍멍
```

Part 3

인터페이스

Part 3 인터페이스(interface)

추상클래스의 극단적인 경우로,
내부의 메소드가 모두 추상메소드로만 이루어진 경우

```
interface Animal {
    abstract void sound();
}
```

- class 대신 interface를 사용한다는 것 외에는 클래스 작성과 동일
- 하지만, 구성요소(멤버)는 추상메소드와 상수만 가능

Part 3 인터페이스의 구현

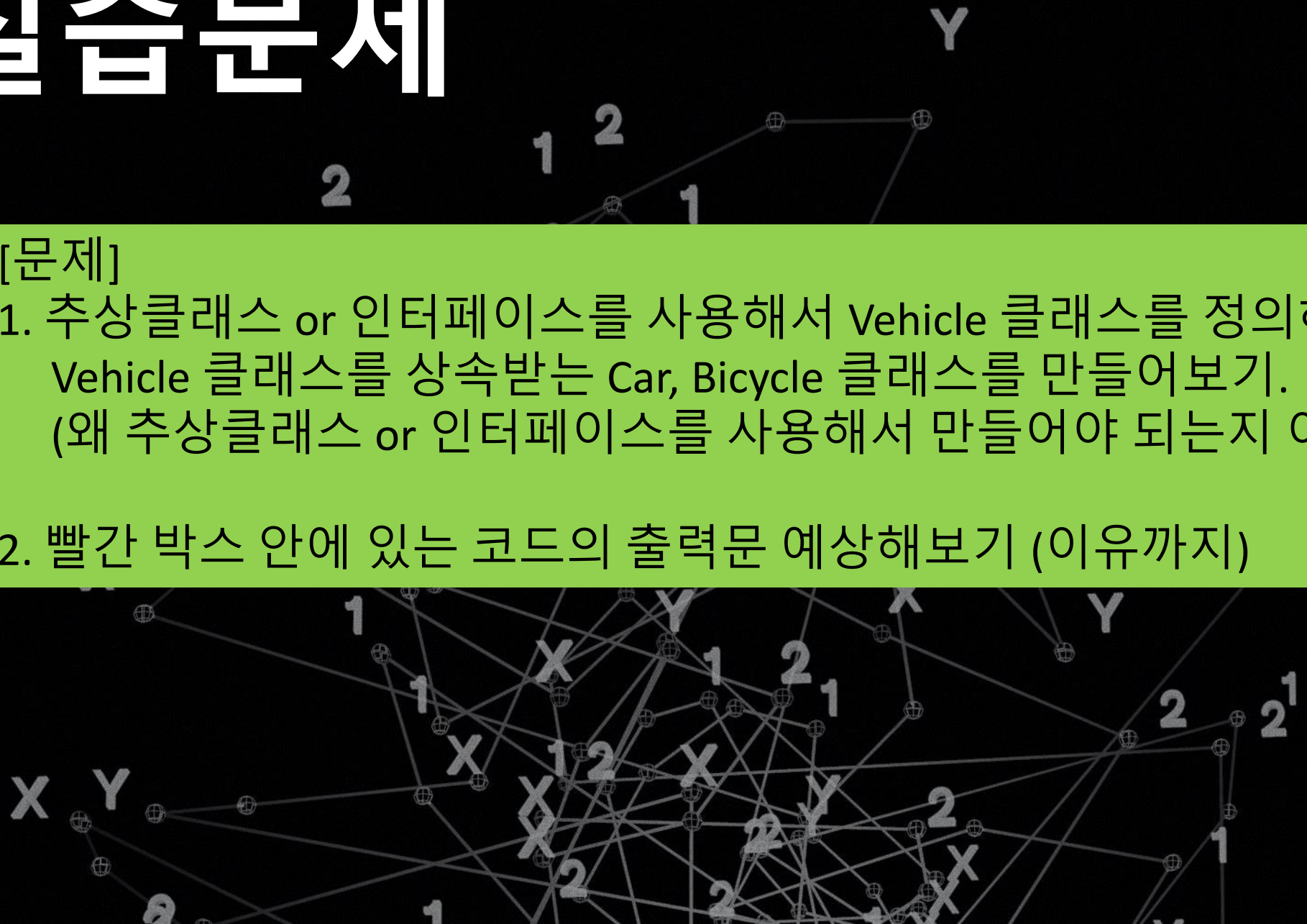
```
class Cat implements Animal {
    public void sound() {
        System.out.println("야옹");
    }
}
```

extends 대신 implements 사용

```
interface Animal {
    abstract void sound();
}

interface Animal2 {
    abstract void sound2();
}

class Cat implements Animal, Animal2 {
    @Override
    public void sound() {
        System.out.println("야옹");
    }

    @Override
    public void sound2() {
        System.out.println("야옹2");
    }
}
```

다중 상속 허용

실습문제



[문제]
1. 추상클래스 or 인터페이스를 사용해서 Vehicle 클래스를 정의하고
   Vehicle 클래스를 상속받는 Car, Bicycle 클래스를 만들어보기.
   (왜 추상클래스 or 인터페이스를 사용해서 만들어야 되는지 이유 생각)

2. 빨간 박스 안에 있는 코드의 출력문 예상해보기 (이유까지)

해설

```java
public class Main {
    Run | Debug
    public static void main(String[] args) {
        Car car = new Car();
        Bicycle bicycle = new Bicycle();

        car.speedCheck();
        car.speedUp();
        car.speedCheck();

        bicycle.speedCheck();
        bicycle.speedUp();
        bicycle.speedCheck();

        Vehicle vehicle = new Car();
        System.out.println(vehicle.name);

        Vehicle vehicle2 = new Bicycle();
        vehicle2.sound();
    }
}
```

올바른 출력문

```
0km/h
10km/h
0km/h
5km/h
```

```java
Vehicle {
    String name = "Vehicle";
    int speed = 0;

    void sound() {
        System.out.println("부우우웅");
    }

    void speedCheck() {
        System.out.println(speed + "km/h");
    }

    abstract void speedUp();
}

class Car Vehicle {
    String name = "Car";

    @Override
    void speedUp() {
        speed += 10;
    }
}

class Bicycle Vehicle {
    @Override
    void sound() {
        System.out.println("따르릉");
    }

    @Override
    void speedUp() {
        speed += 5;
    }
}
```

해설

클래스 내부의 메소드가 추상메소드만 있는 것이 아니기 때문에 abstract class로 만들어야 한다.

```java
public class Main {
    Run | Debug
    public static void main(String[] args) {
        Car car = new Car();
        Bicycle bicycle = new Bicycle();

        car.speedCheck();
        car.speedUp();
        car.speedCheck();

        bicycle.speedCheck();
        bicycle.speedUp();
        bicycle.speedCheck();

        Vehicle vehicle = new Car();
        System.out.println(vehicle.name);

        Vehicle vehicle2 = new Bicycle();
        vehicle2.sound();
    }
}
```

올바른 출력문

```
0km/h
10km/h
0km/h
5km/h
```

Vehicle
(참조변수 타입에 영향)

따르릉
(실제 인스턴스 타입에 영향)

```java
abstract class Vehicle {
    String name = "Vehicle";
    int speed = 0;

    void sound() {
        System.out.println("부우우웅");
    }

    void speedCheck() {
        System.out.println(speed + "km/h");
    }

    abstract void speedUp();
}

class Car extends Vehicle {
    String name = "Car";

    @Override
    void speedUp() {
        speed += 10;
    }
}

class Bicycle extends Vehicle {
    @Override
    void sound() {
        System.out.println("따르릉");
    }

    @Override
    void speedUp() {
        speed += 5;
    }
}
```

Q&A

감사합니다