



JAVA 객체지향 언어

객 체 지 향 언 어 J A V A

김현진

INDEX

JAVA

- 객체 지향

- JAVA

- 자바 백준

- 계획 ..

절차지향 ... 객체지향 ?



절차 지향과 객체 지향의 차이점을 저는 이번에
알았어요..^^;;

절차지향 VS 객체지향

절차 지향

- 실행되는 순서가 **선형**으로 진행
- 자크 **출함**
- 속
- 데이터 **관계를**
갖지 못함

C언어!

객체 지향

- 데이터를 **객체** 단위로
- 생
- C++
Python
JAVA
- 속
- 코드의 **재활용성**이 높음

대표적 언어

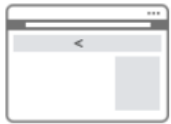
My pick

JAVA

JAVA

SW Expert Academy

SW Expert Academy Works



Code

매주 제
세요. 스
습니다

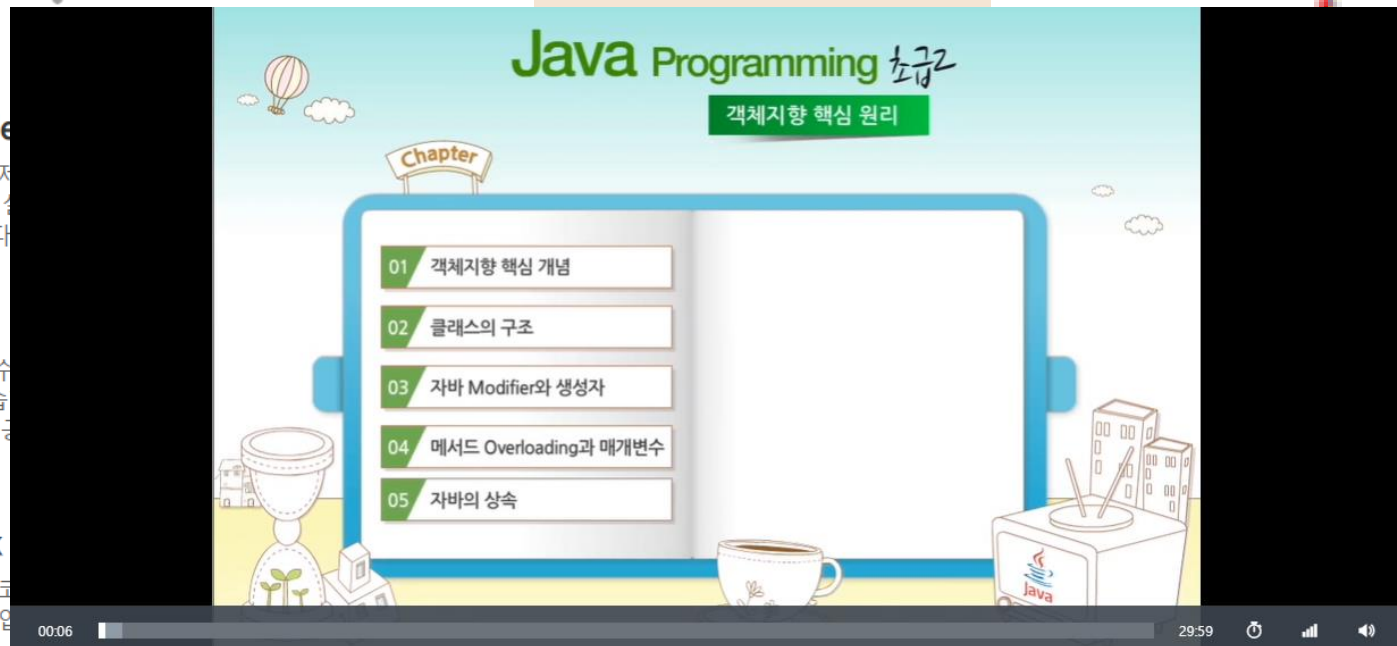
Learn Intensive Study

프로그래밍 역량강화를 위한 필수
을 개인 역량에 따라 스스로 학습
다양한 유형의 학습 콘텐츠를 제



Talk

함께 코
며 협업



JAVA

SW Expert Academy

학습 현황

차시정보

1차시

학습 중 과정 : 2

수료한 과정 : 0

2차시

학습중 Programming Beginner

마지막 학습일 : 2020-05-11 12:07

3차시

Java Programming 초급(1) - 자바언어의 구조와 기본문법 | 3 차시 자바 연산자 및 배열 학습 완료 (100%)

과정 수료하기

4차시

학습중 Programming Beginner

마지막 학습일 : 2020-05-11 05:30

Java Programming 초급(2) - 객체지향 핵심 원리 | 5 차시 자바의 상속 학습 시작 (44%)

이어 시작하기

4차시

메서드 Overloading과 매개변수

학습시간 : 20분 | 강의형태 : 동영상

진도율 100%

완료

JAVA

SW Expert Academy

차시정보 | 총 4 차시

1차시	자바 프로그램 개요 및 실습환경 구축 자바 프로그램 기초, 실습환경 구축 학습시간 : 35분 강의형태 : 동영상
2차시	자바 프로그램 구조 및 데이터 타입 자바 프로그램 기초, 자바 데이터 타입과 변수 학습시간 : 30분 강의형태 : 동영상
3차시	자바 연산자 및 배열 자바 연산자, 자바의 배열 학습시간 : 35분 강의형태 : 동영상
4차시	자바 제어문 자바 제어문 학습시간 : 20분 강의형태 : 동영상

5 차시

1차시	객체지향 핵심 개념 학습시간 : 20분 강의형태 : 동영상
2차시	클래스의 구조 학습시간 : 20분 강의형태 : 동영상
3차시	자바 Modifier와 생성자 학습시간 : 20분 강의형태 : 동영상
4차시	메서드 Overloading과 매개변수 학습시간 : 20분 강의형태 : 동영상

JAVA

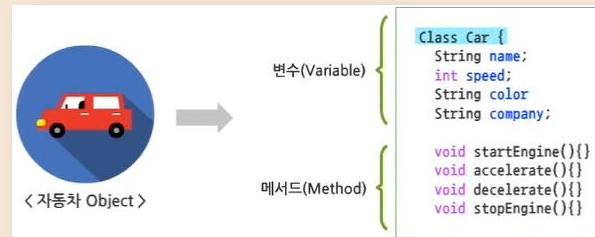
객체

의미를 부여하고 분류하는
논리적인 단위



클래스

객체를 클래스로 생성하여
코드에서 객체 사용



인스턴스

클래스로부터 생성된
메모리상의 객체

```
class Car {  
    int speed;  
    int Gear;  
    ...  
}  
  
public class Hello {  
    public static void main(String[] args) {  
        Car red = new Car(); // 인스턴스를 생성하여 객체에 의해 메모리가 생성됨.  
  
        red.speed = 10;  
        red.Gear = 1;  
        ..  
    }  
}  
  
// new 연산자를 통해 클래스로부터 인스턴스를 생성하여 객체에 의해 메모리가 생성된 것.
```

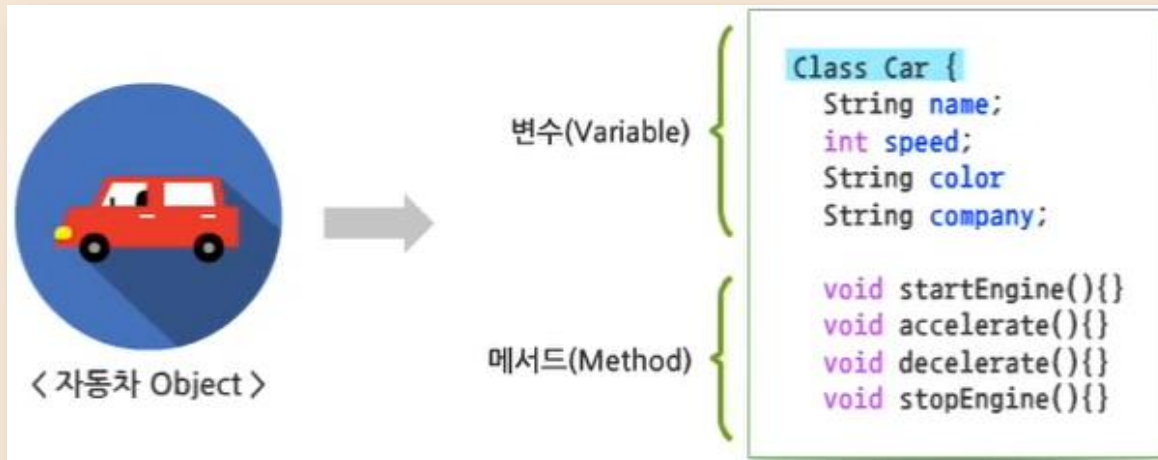
객체



Object

현실세계에 존재하는 모든
것들을 의미하며 이것의
의미를 부여하고 분류하는
논리적인 단위

클래스



Class

현실세계의 객체를 클래스로
선언하여 코드에서 객체를
사용할 수 있게 함

인스턴스



객체

```
class Car {  
    int speed;  
    int Gear;  
    ...  
}  
  
public class Hello {  
    public static void main(String[] args) {  
        Car red = new Car(); // 인스턴스를 생성하여 객체에 의  
  
        red.speed = 10;  
        red.Gear = 1;  
        ..  
    }  
}  
  
// new 연산자를 통해 클래스로부터 인스턴스를 생성하여 객체에 의해 메모리가 생성된 것.
```

Instance

클래스로부터 생성된
메모리 상의 객체

new 연산자를 통해 클래스로
부터 인스턴스를 생성

JAVA

상속 (extend)

```
public class Car {
    String name;
    int speed;

    public void showInfo() {
        System.out.println("Car: " + name + " Speed: " + speed);
    }
}

public class MyCar extends Car {
    int price;

    public void showInfo() {
        System.out.println("MyCar: " + name + " Speed: " + speed + " Price: " + price);
    }
}
```

기존의 **class**를 **이용**하여
변수와 메서드를 추가로 정의한
새로운 클래스를 만드는 것

추상화 (abstract)

```
abstract class Car {
    public void showInfo() {
        System.out.println("Car: " + name + " Speed: " + speed);
    }
}

abstract class MyCar extends Car {
    int price;

    public void showInfo() {
        System.out.println("MyCar: " + name + " Speed: " + speed + " Price: " + price);
    }
}
```

현실 세계에 존재하는 다양한
객체들의 공통된 **특성**을 모아
일반화해 놓은 것

특징

```
public class Car {
    String name;
    int speed;

    public void showInfo() {
        System.out.println("Car: " + name + " Speed: " + speed);
    }
}

public class MyCar extends Car {
    int price;

    public void showInfo() {
        System.out.println("MyCar: " + name + " Speed: " + speed + " Price: " + price);
    }
}
```

하나의 **인터페이스**를 이용하여
서로 다른 구현을 제공함
(메서드 오버로딩, 메서드 오버라이딩)

다형성

변수와 메서드를 하나의 추상화된
클래스로 **묶는 과정**

```
public class Car {
    public void showInfo() {
        System.out.println("Car: " + name + " Speed: " + speed);
    }
}

public class MyCar extends Car {
    int price;

    public void showInfo() {
        System.out.println("MyCar: " + name + " Speed: " + speed + " Price: " + price);
    }
}
```

캡슐화

JAVA

상속
(extend)

기존
변수
새로

하나의
서로
(메서드)

다형성

```
public class Car {  
    String name;  
    int currentGear;  
  
    public void changeGear(int gear) {  
        System.out.println(" 기어를 " + gear + "단으로 변경한다.");  
        currentGear = gear;  
    }  
  
    public String get_CurrentState() {  
        return name + "의 현재 기어 상태:" + currentGear;  
    }  
}  
  
public class Taxi extends Car {  
    int fare;  
    int passenger;
```

추상화
(abstract)

```
abstract class Car {  
    public void startEngine() {  
        System.out.println(" 엔진을 켜고 있습니다.");  
    }  
}  
  
class Car implements Car {  
    public void startEngine() {  
        System.out.println(" 엔진을 켜고 있습니다.");  
    }  
}  
  
abstract class Motor {  
    public void startEngine() {  
        System.out.println(" 엔진을 켜고 있습니다.");  
    }  
}
```

상호화된

캡슐화

```
public class Car {  
    public String name;  
    private int currentGear;  
    public void startEngine() {  
        System.out.println(" 엔진을 켜고 있습니다.");  
    }  
}  
  
public class Motor {  
    public void startEngine() {  
        System.out.println(" 엔진을 켜고 있습니다.");  
    }  
}
```

JAVA

상속 (extend)

```
public class Car {
    private String name;
    private int speed;

    public Car(String name, int speed) {
        this.name = name;
        this.speed = speed;
    }

    public void printInfo() {
        System.out.println("Car: " + name + " Speed: " + speed);
    }
}
```

기존의 **class**를 **이용**하여
변수와 메소드를 추가로 정의한
새로운 클래스를 만드는 것

추상화 (abstract)

```
abstract class Car {
    public void printInfo() {
        System.out.println("Car: " + name + " Speed: " + speed);
    }
}

abstract class Car {
    public void printInfo() {
        System.out.println("Car: " + name + " Speed: " + speed);
    }
}
```

현실 세계에 존재하는 다양한
객체들의 공통된 **특성**을 모아
일반화해 놓은 것

특징

```
public class Car {
    private String name;
    private int speed;

    public Car(String name, int speed) {
        this.name = name;
        this.speed = speed;
    }

    public void printInfo() {
        System.out.println("Car: " + name + " Speed: " + speed);
    }
}
```

하나의 **인터페이스**를 이용하여
서로 다른 구현을 제공함
(메소드 오버로딩, 메소드 오버라이딩)

다형성

변수와 메소드를 하나의 추상화된
클래스로 **묶는 과정**

```
public class Car {
    private String name;
    private int speed;

    public Car(String name, int speed) {
        this.name = name;
        this.speed = speed;
    }

    public void printInfo() {
        System.out.println("Car: " + name + " Speed: " + speed);
    }
}
```

캡슐화

상속 (extend)

```
public class Car {
    String name;
    int currentGear;

    public void changeGear(int gear) {
        System.out.println("기어를 " + gear + "단으로 변경한다.");
        currentGear = gear;
    }

    public String getCurrentState() {
        return name + "의 현재 기어 상태: " + currentGear;
    }
}
```

기존의 class
변수와 메서드
새로운 클래스

다형성

하나의 인터페이스
서로 다른 구현
(메소드 오버로딩)

```
public class Car {
    String name;
    int currentGear;

    public void changeGear(int gear) {
        System.out.println("기어를 " + gear + "단으로 변경한다.");
        currentGear = gear;
    }

    public String getCurrentState() {
        return name + "의 현재 기어 상태: " + currentGear;
    }
}

public class MyTaxi extends Car {
    public void changeGear(int gear) {
        System.out.println("기어를 " + gear + " 변경후 고정한다.");
        currentGear = gear;
    }
}

public static void main(String[] args) {
    MyTaxi taxi = new MyTaxi();
    taxi.name = "24바 3982";
    taxi.currentGear = 1;
    taxi.fare = 6200;
    taxi.passenger = 2;
    taxi.changeGear(2);
    taxi.getCurrentState();
}
```

추상화 (abstract)

```
abstract class Car {
    public void getCurrentState() {
        System.out.println("기어 상태: " + currentGear);
    }
}

class MyTaxi extends Car {
    public void changeGear(int gear) {
        System.out.println("기어를 " + gear + "단으로 변경한다.");
        currentGear = gear;
    }
}
```

는 다양한
성을 모아

나의 추상화된

캡슐화

```
public class Car {
    private String name;
    private int currentGear;

    public void changeGear(int gear) {
        System.out.println("기어를 " + gear + "단으로 변경한다.");
        currentGear = gear;
    }

    public String getCurrentState() {
        return name + "의 현재 기어 상태: " + currentGear;
    }
}
```

오버 라이딩

JAVA

상속 (extend)

```
public class Car {
    String name;
    int speed;

    public void engineOn() {
        System.out.println("Car is now running");
    }

    public void engineOff() {
        System.out.println("Car is now stopped");
    }
}

public class FastCar extends Car {
    int fuel;
    int maxSpeed;
}
```

기존의 **class**를 **이용**하여
변수와 메소드를 추가로 정의한
새로운 클래스를 만드는 것

추상화 (abstract)

```
abstract class Car {
    public void engineOn() {
        System.out.println("Car is now running");
    }

    public void engineOff() {
        System.out.println("Car is now stopped");
    }
}

abstract class FastCar {
    public void engineOn() {
        System.out.println("FastCar is now running");
    }

    public void engineOff() {
        System.out.println("FastCar is now stopped");
    }
}
```

현실 세계에 존재하는 다양한
객체들의 공통된 **특성**을 모아
일반화해 놓은 것

특징

```
public class Car {
    String name;
    int speed;

    public void engineOn() {
        System.out.println("Car is now running");
    }

    public void engineOff() {
        System.out.println("Car is now stopped");
    }
}

public class FastCar {
    String name;
    int speed;
    int fuel;
    int maxSpeed;

    public void engineOn() {
        System.out.println("FastCar is now running");
    }

    public void engineOff() {
        System.out.println("FastCar is now stopped");
    }
}
```

하나의 **인터페이스**를 이용하여
서로 다른 구현을 제공함
(메소드 오버로딩, 메소드 오버라이딩)

다형성

변수와 메소드를 하나의 추상화된
클래스로 **묶는 과정**

```
public class Car {
    public void engineOn() {
        System.out.println("Car is now running");
    }

    public void engineOff() {
        System.out.println("Car is now stopped");
    }
}

public class FastCar {
    public void engineOn() {
        System.out.println("FastCar is now running");
    }

    public void engineOff() {
        System.out.println("FastCar is now stopped");
    }
}
```

캡슐화

상속 (extend)

```
public class Car {
    String name;
    int speed;

    public void showInfo() {
        System.out.println("자동차 이름: " + name + ", 속도: " + speed);
    }
}

public class MyCar extends Car {
    int fuel;

    public void showInfo() {
        System.out.println("자동차 이름: " + name + ", 속도: " + speed + ", 연료: " + fuel);
    }
}
```

기존의 class
변수와 메소드
새로운 클래스

```
public class Car {
    String name;
    int speed;

    public void showInfo() {
        System.out.println("자동차 이름: " + name + ", 속도: " + speed);
    }
}

public class MyCar extends Car {
    int fuel;

    public void showInfo() {
        System.out.println("자동차 이름: " + name + ", 속도: " + speed + ", 연료: " + fuel);
    }
}
```

하나의 인터페이스
서로 다른 구현
(메소드 오버로딩)

다형성

```
abstract class Car {

    public void CurrentState () {
        System.out.println(" 움직이고 있습니다.");
    }

}

class Bus extends Car {
    String color;
    public void Color(String color) {
        System.out.println("버스는 "+color+" 입니다.");
    }

}

public class MyBus {
    public static void main(String[] args) {
        Bus bus = new Bus();

        bus.Color("초록색");
        bus.CurrentState();
    }
}
```

추상화 (abstract)

다양한
을 모아

의 추상화된

```
public class Car {
    String name;
    int speed;

    public void showInfo() {
        System.out.println("자동차 이름: " + name + ", 속도: " + speed);
    }
}

public class MyCar extends Car {
    int fuel;

    public void showInfo() {
        System.out.println("자동차 이름: " + name + ", 속도: " + speed + ", 연료: " + fuel);
    }
}
```

캡슐화

JAVA

상속 (extend)

```
public class Car {
    String name;
    int horsepower;

    public void displayInfo() {
        System.out.println("이름: " + name + " 마력: " + horsepower);
    }
}

public class MyCar extends Car {
    int price;
    int mileage;
}
```

기존의 **class**를 **이용**하여
변수와 메소드를 추가로 정의한
새로운 클래스를 만드는 것

추상화 (abstract)

```
abstract class Car {
    public void startEngine() {
        System.out.println("엔진 시동");
    }
}

class MyCar extends Car {
    String name;
    public void displayInfo() {
        System.out.println("이름: " + name + " 마력: " + horsepower);
    }
}

public class MyCar {
    public static void main(String[] args) {
        MyCar myCar = new MyCar();
        myCar.startEngine();
    }
}
```

현실 세계에 존재하는 다양한
객체들의 공통된 **특성**을 모아
일반화해 놓은 것

특징

```
public class Car {
    String name;
    int horsepower;

    public void displayInfo() {
        System.out.println("이름: " + name + " 마력: " + horsepower);
    }
}

public class MyCar extends Car {
    int price;
    int mileage;

    public void displayInfo() {
        System.out.println("이름: " + name + " 마력: " + horsepower + " 가격: " + price + " 주행거리: " + mileage);
    }
}
```

하나의 **인터페이스**를 이용하여
서로 다른 구현을 제공함
(메소드 오버로딩, 메소드 오버라이딩)

다형성

변수와 메소드를 하나의 추상화된
클래스로 **묶는 과정**

```
public class Car {
    public static void main(String[] args) {
        Car car = new Car();
        car.startEngine();
    }
}

public class MyCar extends Car {
    public void startEngine() {
        System.out.println("엔진 시동");
    }
}
```

캡슐화

JAVA

상속 (extend)

```
public class Car {
    String name;
    int currentSpeed;

    public void startEngine() {
        System.out.println("자동차의 엔진을 켜고 있습니다.");
    }

    public void accelerate() {
        currentSpeed = currentSpeed + 10;
    }

    public void getSpeed() {
        return currentSpeed;
    }
}
```

기존의 class
변수와 메소드
새로운 클래스

```
public class Car {
    String name;
    int currentSpeed;

    public void startEngine() {
        System.out.println("자동차의 엔진을 켜고 있습니다.");
    }

    public void accelerate() {
        currentSpeed = currentSpeed + 10;
    }

    public void getSpeed() {
        return currentSpeed;
    }
}
```

하나의 인터페이스
서로 다른 구현
(메소드 오버로딩)

다형성

```
public class Car {
    public String name;
    private int currentSpeed;

    public void startEngine() {
        System.out.println(name + "의 시동을 켜다.");
        currentSpeed = 1;
    }

    private int accelerate() {
        currentSpeed = currentSpeed * 10;
        return currentSpeed;
    }

    public String get_CurrentSpeed() {
        return name + "의 속도는 " + CurrentSpeed();
    }
}
```

정보 은닉을 위해
**private
Modifier** 를 사용
하였다!

추상화 (abstract)

는 다양한
성을 모아

```
abstract class Car {
    public void startEngine() {
        System.out.println("자동차의 엔진을 켜고 있습니다.");
    }
}

class Car {
    String name;
    public void startEngine() {
        System.out.println(name + "의 시동을 켜다.");
    }
}

public class Main {
    public static void main(String[] args) {
        Car car = new Car();
        car.startEngine();
    }
}
```

하나의 추상화된

캡슐화

JAVA

클래스

- 클래스: 현실 세계의 객체들을 추상화하여 만들어낸

```
public class Car {    // 클래스 선언부
    // 멤버 변수
    String name;
    int currentGear;

    // 메서드
    public void changeGear(int gear) {
        System.out.println("기어를 "+ gear + "로 변경합니다.");
        currentGear = gear;
    }

    public String get_CurrentState() {
        return name + "의 현재 기어 상태:" + currentGear;
    }
}
```

- 클래스에 사용되는 Modifier

- 접근 권한 예약어: public (모든 클래스에서 접근 가능)
- 활용 방법 예약어: final (자식 클래스를 가질 수 없는 클래스)

- 멤버 변수에 사용되는 Modifier

- 접근 권한: public (모든 클래스에서 접근 가능), private (변수가 선언된 클래스 내에서만 접근 가능)
- 활용 방법: final (변수를 상수로 이용할 때), static (클래스 변수)

종류	클래스	하위 클래스
private	○	×
(default)	○	×
protected	○	○
public	○	○

1. 객체 참조 변수 선언

```
Car red; // 클래스이름 객체참조변수이름;
```

2. 객체 생성

```
red = new Car(); // 객체참조변수이름 = new 클래스이름();
```

이 두가지를 한번에 쓸 수 있음

```
Car red = new Car();
```

객체 생성 ex)

```
class Car {                // 변수를 가진 클래스 선언
    String color;
    int number;
}

public class test {        // 실행을 위한 선언
    public static void main(String[] args) {
        Car mycar = new Car(); // Car클래스를 참조하는
        // 객체 생성

        mycar.color = "red";
        mycar.number = 1278;

        System.out.println(mycar.color + "의 뒷번호는 " + mycar.number);
    }
}
```

```
class Car {                // 변수를 가진 클래스 선언
    String color;
    int number;
}

public class test {
    public static void main(String[] args) {
        Car mycar = new Car(); // 객체1
        mycar.color = "red";
        mycar.number = 1278;
        System.out.println(mycar.color + "의 뒷번호는 " + mycar.number);
        Car bus = new Car(); // 객체2
        bus.color = "green";
        bus.number = 1278;
        System.out.println(bus.color + "의 뒷번호는 " + bus.number);
    }
}
```

2. 이차원 배열

```
String[][] name; // "or" String name[] []; // "or" String[] name = new String[배열의 배열길이][배열길이];
or
name = new String[2][];
name[0] = new String[4];
name[1] = new String[4]; // 2x4의 2차원 배열 생성
```

- 이동 제어문

- break
- continue
- return

- 변수의 중복

- 변수는 데이터 타입이 달라도 이름이 동일할 경우 허용하지 않음
- 메서드는 클래스 안에서 동일한 이름으로 여러개 정의 가능 (매개변수)

메서드 Overloading

- 하나의 클래스에 동일한 이름의 메서드가 중복되어 정의 되어 있는 경우
- 메서드를 호출할 때 동일한 이름의 메서드를 사용하므로 매개변수로 구분

```
import java.util.Scanner; // Scanner 사용을 위해 모듈을 선언

public class Main {

    public static void main(String args[]) {

        Scanner pr = new Scanner(System.in); // System.in = Input
        int a, b;
        a = pr.nextInt(); // Scanner메소드 next를 사용하여 입력받기
        b = pr.nextInt(); // nextInt : 정수형, next : 단어, nextLine : 문자열
        System.out.println(a + b);
        System.out.println(a - b);
        System.out.println(a * b);
        System.out.println(a / b);
        System.out.println(a % b);
    }
}
```

자바 시작

- 자바의 출력 (자바의 입출력 프로그램을 구현하기 위해선 java.io 패키지를 사용하면 됨.)

```
System.out.println("Hello World!");
```



- 예약어: 시스템에서 일정 특성을 가진 언어로 등록된 것으로 데이터 타입이나 프로그램을 정의함

abstract	assert	boolean	break	byte	cast	catch
char	class	const	continue	default	do	double
else	extends	false	final	finally	float	for
goto	if	implements	import	instanceof	int	interface
long	native	new	null	package	private	protected
public	return	short	static	super	switch	synchronized
this	throw	throws	transient	true	try	void
volatile	volatile					

- const, goto는 현재 사용되고 있지 않으며, 식별자로도 사용할 수 없음
- sizeof 예약어는 자바에서 더 이상 사용되지 않음

- 자료형: 변수 앞에 붙이는 데이터 타입 (데이터의 성격을 규정)

표현형태	데이터 타입	정의
논리값	boolean	참이나 거짓을 나타내는 값
단일 문자	char	16bit의 유니코드 문자 데이터
정수	byte	부호가 있는 8bit의 정수
	short	부호가 있는 16bit의 정수
	int	부호가 있는 32bit의 정수
	long	부호가 있는 64bit의 정수
실수	float	부호가 있는 32bit의 부동소수점 실수
	double	부호가 있는 64bit의 부동소수점 실수

My pick

백준

백준

<u>1000</u>	맞았습니다!!	14292 KB	104 ms	Java / 수정
2557	맞았습니다!!	12652 KB	64 ms	Java / 수정
1001	맞았습니다!!	14296 KB	116 ms	Java / 수정
10998	맞았습니다!!	14316 KB	108 ms	Java / 수정
1008	맞았습니다!!	14424 KB	108 ms	Java / 수정
10869	맞았습니다!!	14324 KB	128 ms	Java / 수정

간단한 백준 문제 풀이

백준

사칙연산

```
import java.util.Scanner; // Scanner 사용을 위해 모듈을 선언
```

10

```
public class Main {
```

/ 수정

```
    public static void main(String args[]) {
```

```
        Scanner pr = new Scanner(System.in); // System.in = InputStream의 in(변수)
```

```
        int a, b;
```

```
        a = pr.nextInt(); // Scanner메소드 next를 사용하여 입력받기
```

```
        b = pr.nextInt(); // nextInt : 정수형 , next : 단어 , nextLine : 문장
```

```
        System.out.println(a + b);
```

```
        System.out.println(a - b);
```

```
        System.out.println(a * b);
```

```
        System.out.println(a / b);
```

```
        System.out.println(a % b);
```

```
    }
```

```
}
```


앞으로의 계획 ...



안드로이드 스튜디오를 이용하여 자바를 사용한

앱 Frontend 개발 !

Thank You

감사합니다