1. 메소드(함수, 펑션)

1. 수학에서 함수는 어떠한 값들을 받아서 일련의 처리과정을 통해 결과 값을 반환하는 기능.
2. 프로그래밍 언어에서의 메소드(함수, 펑션)도 동일하게 어떠한 값들을 받아서(매개변수, 파라미터) 일련의 처리과정을 만드는 것을 메소드라고 한다.
3. 메소드는 결과 값을 전달해주는 메소드도 있고 결과 값이 없는 메소드도 존재한다. 메소드는 클래스(설계도) 안에 정의를 주로 하는데 설계도로 만들어지는 객체(부품)의 기능을 담당한다.

```
Class 타이어 {
    //속성값들. '변수'라고 한다.
    재질,
    가격,
    크기

    //기능들(기능 역할을 해주는 것을 '메소드'라고 한다.)
    타이어를 휠 크키게 맞춘다.
    모양을 잡는다.
    펑크난 곳을 수선한다.
}
```

5. 접근제어자
    - 접근제어자는 변수, 클래스, 메소드 앞에 지정하여 변수나 클래스, 메소드를 호출할 수 있는 범위를 지정해주는 역할.
    - 접근제어자의 종류
        - public: 모든 패키지에서 호출할 수 있는 범위를 갖는 접근제어자. (같은 클래스, 같은 패키지 클래스, 자식 클래스, 그 외의 모든 영역) 제일 넓은 범위를 갖는 접근제어자.
        - protected: 같은 클래스, 같은 패키지내의 클래스, 자식 클래스까지의 접근 범위를 허용하는 접근제어자.
        - default: 접근제어자를 명시하지 않았을 때 지정되는 접근제어자. 같은 클래스, 같은 패키지 내의 클래스에서만 접근이 허용되는 접근제어자. //나중에 인터페이스에서 default라는 키워드가 있다.
        - private: 제일 작은 범위를 갖는 접근제어자. 같은 클래스에서만 사용 가능.
        - public > protected > default > private (private이 보안성이 제일 좋다.)

6. 리턴타입(결과값의 타입) //기능이 동작하고 결과값의 타입.
    - 메소드가 (기능)가 종료된 후 반환(리턴, 전달)될 결과 값을 타입.
    - 결과 값이 없는 메소드와 결과 값이 있는 메소드로 구분할 수 있다.
    - 결과 값이 없는 메소드는 void라는 키워드를 사용해서 메소드의 리턴타입을 지정할 수 있다. void로 리턴타입이 지정된 메소드는 return 구문을 가지면 안된다. return을 사용하면 에러가 발생된다.

```
public void add() {
    //출력 기능만 하는 메소드
    System.out.println(10 +20);
    //return문을 사용할 수 없다.
}
```

    - 결과가 값이 있는 메소드는 결과 값이 어떤 타입인지 지정하면 된다. 만약에 결과값이 inf로 나오면 int형으로 지정, double로 나오면 double로 지정, 이외의 참조형 타입(배열, List, String, 개발자가 직접 생성한 클래스....) 등으로 지정할 수 있다. 결과 값이 있는 메소드는 항상 return 구문으로 리턴타입으로 지정된 타입의 값을 반환해줘야 한다.

```
public int add() {
    return 10 + 20;
}

public double add() {
    return 20.0 / 10;
}

public String hello() {
    return "hello java"; //만약 리턴으로 숫자를 넣으면
}
```

7. 매개변수(파라미터) : 어떠한 값들을 받는 것을 매개변수 혹은 파라미터라고 한다.
    - 매개변수(파라미터)는 메소드에서 받아서 처리해 줄 값들을 의미한다.
    - 매개변수가 있는 메소드와 매개변수가 없는 메소드로 구분할 수 있다.
    - 매개변수의 지정 위치는 메소드명 뒤에 잇는 소괄호() 안에 지정한다.
    - 매개변수의 지정은 변수의 타입과 변수명으로 지정한다.

- 매개변수로 받을 수 있는 타입은 기본타입(int, double, float, long, ...), 참조타입(String, 배열, List, 개발자가 직접 만든 클래스, ...) 모두 받을 수 있다.
- 매개변수의 개수에는 제한이 없다. 개발자가 만들기 나름이다.

```
public void add(int num1, double num2, long num3, St

}

public int add(int a, int b) {
    return a + b;
    //받아온 값들(a,b)을 만들어서(더하기) 던져줄 수 있[
}

public double div(int a, int b) {
    return (double)a / b;
}
```

8. 메소드의 선언방식(형태)

```
- 접근제어자 리턴타입 메소드명(매개변수) {
    기능처리
    return 구문;
}
```

9. 메소드 오버로딩과 오버라이딩
    - 기본적으로 메소드는 동일한 이름으로 여러 개를 생성할 수 없다.
    - 메소드 오버로딩과 오버라이딩을 이용하면 같은 이름의 메소드를 생성할 수 있다.
    - 메소드 오버로딩은 같은 이름의 메소드로 매개변수의 개수나, 타입을 변경해서 메소드를 생성하는 것

```
//메소드 오버로딩의 예

public int add(int a, int b) {
    return a + b;
}

//오버로딩된 메소드
//매개변수의 개수를 다르게 하여 다른 메소드로 인식
public int add(int a, int b, int c) {
    return a + b + c;
}

//매개변수의 타입을 다르게 하여 다른 메소드로 인식
public int add(int a, double b) {
    return (int)(a + b);
}
```

- 메소드 오버라이딩은 상속과 관련된 기능. 부모클래스에 존재하는 메소드를 자식클래스에서 재정의(다시한번 정의) 하는 것. 부모클래스에 존재하는 메소드와 재정의된 메소드의 형태(리턴타입, 매개변수 개수, 매개변수 타입 등)이 모두 동일해야 한다. 부모클래스의 똑같은 모양으로 똑같이 정의해야 한다.

10. 메소드의 장단점
    - 장점1 : 기능들을 메소드로 분리하면 코드가 모듈화되어 코드의 재사용성과 코드의 가독성이 좋아진다.
    - 장점2 : 에러가 발생한 메소드나 논리적으로 오류가 있는 메소드만 수정하면 되기 때문에 유지보수성과 오류 수정하기 쉬워진다.
    - 단점1 : 너무 많은 기능들을 메소드로 만들어 놓으면 오히려 코드가 더 복잡해진다.
    - 단점2 : 따라서 공통적으로 많이 사용되는 기능들만 메소드로 만들어서 사용하는 것이 권장된다. (예를들면 int타입 double타입별로 메소드를 만들면 복잡해 보이겠지)