메서드

💡 메서드란 자바프로그램에서 사용하는 최소한의 기능 단위로 자주 등장하는 코드를 **재사용 하기 위해** 그룹화 해놓은 단위라고 이해하면 좋습니다.

#01. 지금까지 작성한 메인 클래스 형식

```
public class HelloWorld {
    public static void main(String[] args) {
        // ... 구현내용 ...
    }
}
```

이 단원에서 설명 가능한 부분

- void 키워드의 기능.
- main 대신 다른 이름으로 직접 메서드 정의하기.
- 메서드 이름 옆의 괄호`()`의 의미.

#02. 메서드의 이해

1) 수학에서의 함수

> $f(x) = x + 1$

이 수식은 `x`라는 조건값에 따라 `x+1`이라는 계산식을 재사용한다.

> $f(5) = 6$
> $f(10) = 11$

2) 자바 프로그램에의 메서드

자바 프로그램도 특정 기능(=연산)을 **그룹화 해서 재사용하기 위한 단위**를 정의할 수 있는데 이를 메서드(=함수)라고 한다.

메서드 정의하기

메서드는 `class` 블록 안에서 다른 메서드와의 순서에 상관 없이 정의한다.

```
public static void 메서드이름() {
    ... 수행할 동작 ...
}
```

> 💡 `public`, `static` 키워드는 각각 별도의 기능을 갖는 예약어 입니다. 아직은 이 단어들의 기능을 소개할 기회가 없었기 때문에 이 단원에서는 지금껏 사용하던 `main()` 메서드와 동일하게 이 키워드들을 그

> 대로 적용한 상태로 진행합니다.

메서드의 호출

정의된 메서드는 다음의 형식으로 사용할 수 있으며, 이를 **호출한다**고 한다.

```
메서드이름();
```

메서드의 위치

메서드는 반드시 클래스 안에 포함되어야만 한다. 클래스 밖에는 절대 존재할 수 없다.

특정 조건이 충족되지 않는 한 하나의 클래스에 이름이 동일한 메서드가 공존할 수 없다.

> 💡 특정 조건은 뒤에서 다룹니다.

메서드 이름 규칙

- 영어, 숫자, 언더바, `$` 조합만 사용 가능.
- 첫 글자는 반드시 영어로만 지정 가능
- 두 개의 단어 이상을 조합할 경우
    1. 언더바 사용 : hello + world --> hello_world
    2. 대문자 변환 : hello + world --> helloWorld
        - 자바에서는 이 형태가 일반적

일반적으로 메서드는 프로그램이 수행해야하는 기능(동작)을 구현하는 것이기 때문에 **동사+목적어** 형태로 정의하며 첫 글자는 영어 소문자로 시작한다.

ex)

- 내용작성하기 : writeContent()
- 계정탈퇴확인: confirmAccountOut()
- 상품추가 : addProduct()
- 항목삭제 : removeItem()

메서드 정의하고 호출하기

메서드는 소스코드에서의 순서에 상관 없이 `main()` 이 시작점이 된다.

> Ex01_메서드사용.java

#03. 파라미터

1) 수학의 매개변수

> $f(x) = x + 1$

함수 `f(x)`는 주어지는 값 `x`에 따라 각각 다른 결과를 만들어낸다.

수학에서 함수 `f(x)`가 연산을 수행하기 위해 주어지는 **조건값** `x`를 매개변수라 한다.

2) 메서드의 파라미터

기본적으로 수학의 매개변수와 동일한 개념.

메서드는 자신이 실행되는데 필요한 **조건값**을 메서드 이름 뒤의 괄호`()`안에 **변수 형태로 선언**한다. 파라미터라 한다.

```
public static void foo(int bar) {
    ...
}
```

파라미터를 갖는 메서드는 반드시 호출시에 그 값을 전달해야 한다.

```
foo(100);   // foo라는 메서드를 호출하면서 bar 변수에 100을 할당함.
```

> Ex02_파라미터.java

#04. 다중 파라미터

1) 수학의 함수에서 여러 개의 매개변수

특정 함수가 연산을 수행하기 위해 두 개 이상의 조건값이 필요하다면 다음과 같이 콤마(,)로 구분하여 명시할 수 있다.

> $f(x_1, x_2) = x_1 + x_2 + 1$

매개변수가 여러 개 존재하는 함수는 호출시에 반드시 그에 맞는 값들을 전달해야만 한다.

> $f(5, 10) = 5 + 10 + 1 = 16$

2) 메서드의 다중 파라미터

메서드가 기능을 수행하는데 두 개 이상의 파라미터가 필요하다면 콤마(,)로 구분하여 선언할 수 있다.

```
public static void helloworld(int foo, int bar) {
    ...
}
```

메서드 `helloworld()`를 호출할 때는 반드시 정의된 파라미터값들을 전달해야만 한다.

```
helloworld(1, 2);
helloworld(3, 4);
```

> Ex03_다중파라미터.java

#05. 메서드의 결과값

1) 값의 반환

수학의 함수는 자신이 포함하고 있는 수식에 대한 결과를 반환한다.

> $f(x) = x + 1$

위와 같이 방정식이 정의되어 있을 경우

> $y = f(5)$

라고 호출하면 **함수 f(x)는 자신이 호출된 위치로 연산의 결과인 6을 반환**하여 아래와 같이 계산되어 진다.

> $y = 6$

2) 메서드의 리턴값

메서드가 수행 결과를 자신이 호출된 위치에 반환하는 것을 **리턴**이라고 하며, 이 때 반환되는 값을 **리턴값**이라고 한다.

예약어 return의 사용

메서드 안에서 값을 리턴하기 위해서는 `return`이라는 키워드가 사용되며 메서드 이름 옆에 `void` 대신 리턴하고자 하는 값의 데이터 타입을 반드시 명시해야 한다.

`void`는 리턴값이 없다는 의미이다.

```
public static int foo() {
    return 100;
}
```

3) 리턴값의 활용

함수가 호출되는 위치에 결과값을 돌려놓는 원리이므로 일반 변수와 같이 사용할 수 있다.

결과를 리턴하는 메서드는 리턴값을 받아 다른 변수에 대입할 수 있다

```
int bar = foo();    // --> int bar = 100;
```

일반 변수처럼 다른 연산에 활용할 수도 있다

```
int y = foo() + 5;  // --> int y = 100 + 5;
```

함수의 리턴값을 직접 출력할 수 도 있다

```
System.out.println( foo() );        // --> System.out.println( 100 );
System.out.printf("%d\n", foo());   // --> System.out.printf("%d\n", 100);
```

4) return 키워드의 기능

메서드가 수행되는 도중 특정 조건이 충족되어 return키워드를 만나게 되면 메서드는 그 즉시 수행을 중단한다.

x가 10보다 작은 경우 아무것도 하지 않고 종료

void 형은 리턴값이 없다는 의미이므로 별도의 리턴값 없이 return 키워드만 사용한다.

```
public void foo(int x) {
    if (x < 10) {
        return; // 이 구문을 만나는 즉시 메서드 실행 종료.
    }
    System.out.println("x=" + x);
}
```

x가 10보다 작은 경우 0을 리턴하고 종료

int 타입으로 리턴형을 정의하고 있기 때문에, 처리 중단을 위한 return 의 사용이라도 정수형의 값을 반환해야 한다. 이 경우 가장 일반적으로 사용되는 값은 0 이다.

```
public int bar(int x) {
    if (x < 10) {
        return 0;
    }
    return x + 10;
}
```

조건 없이 return 키워드를 사용하는 경우

return 0; 이후의 구문들 실행될 수 없으므로 불필요한 구문으로 인식되어 컴파일 시에 에러가 발생한다.

```
public static int some(int x) {
    return 0;
    int y = x + 10;  // <-- 컴파일시에 이 부분을 에러로 판단한다.
    return y;
}
```

위 코드는 컴파일시 아래와 같은 에러 메시지를 표시한다.

```
Test.java:4: error: unreachable statement
        int y = x + 10;
            ^
1 error
```

> Ex04_리턴값.java

#06. 메서드 간의 호출

1) 함수간의 호출

아래의 함수는 `f2`에서 `f1`을 호출하고 있다.

> $f_1(x_1) = x_1 + 1 \;\; f_2(x_2) = f1(x_2) * 2$

이 경우 `f2`에게 10을 전달하면 10이 다시 `f1`에게 전달된다.

> $f_2(10) = f_1(10)x2$
> $= (10 + 1)x2$
> $= 11 * 2$
> $= 22$

> Ex05_메서드간의_호출.java

2) 메서드의 호출순서와 종료순서 간의 관계

MyMethod5.java에서 메서드의 호출 순서는 다음과 같다.

> $main > f2 > f1$

종료되어 가장 먼저 결과를 만들어 내는 순서는 다음과 같다.

> $f1 > f2 > main$

메서드 `a()`가 실행하는 도중 그 안에서 다른 메서드 `b()`를 호출하게 되면 `b()`가 종료되기 전까지 `a()`는 종료하지 못하고 대기상태가 된다.

즉, 먼저 실행(First Input)된 메서드가 가장 나중(Last Output)에 종료된다.

이를 **FILO** 혹은 **선입후출** 관계라고 하는데 프로그래밍에서는 이러한 관계를 **스택(Stack)**이라고 한다.

> Ex06_호출스택.java

#07. 파라미터의 값 복사, 참조 복사

1) 파라미터의 값 복사

파라미터의 유효성 범위

파라미터는 자신이 정의된 메서드에 속한 변수이다. 그러므로 파라미터는 메서드에 대한 `{}` 을 벗어날 수 없다.

메서드가 서로 다르면 변수의 유효성 범위(스코프)가 서로 다르기 때문에 변수 이름이 동일하더라도 서로 다른 값으로 구분된다.

아래 두 개의 메서드 중, `foo()`에서 정의한 파라미터 `a`와 변수 `b`는 `bar()`메서드가 정의하는 파라미터 `a`나 변수 `b`와는 단지 이름만 같을 뿐 서로 다른 값이다.

> 💡 실제 생성되는 메모리의 위치가 다르기 때문에 구별됩니다.

기본 자료형 변수를 파라미터로 사용하는 경우

파라미터로 전달되는 변수는 값의 복사에 대한 개념이다. 그러므로 변수의 이름과 파라미터의 이름은 서로 아무런 연관이 없다.

결국 **기본 자료형 변수를 파라미터로 전달할 경우에만 값 복사가 발생**하기 때문에 파라미터를 전달받은 메서드 안에서 파라미터에 대한 수정이 발생하더라도 원래의 원본은 변하지 않는다.

> Ex07_파라미터_값복사.java

2) 파라미터의 참조 복사

배열을 파라미터로 받는 메서드

일반 배열끼리의 대입과 마찬가지로 파라미터로 전달되는 배열 역시 참조 형태로 전달된다.

즉, 배열 파라미터를 전달받은 `A()`라는 메서드에서 배열의 원소를 수정하면 `A()` 를 호출하기 전의 원본 값도 함께 변경된다.

> Ex08_파라미터_참조복사.java

#08. 메서드 오버로딩 (Overloading)

메서드 오버로딩(overloading)이란 같은 이름의 메서드를 중복하여 정의하는 것을 의미

자바에서는 원래 한 클래스 내에 같은 이름의 메서드를 둘 이상 가질 수 없지만 매개변수의 개수나 타입을 다르게 하면, 하나의 이름으로 메서드를 작성할 수 있다.

이러한 메서드 오버로딩을 사용함으로써 메서드에 사용되는 이름을 절약할 수 있다.

> 메서드를 호출할 때 전달해야 할 매개변수의 타입이나 개수에 대해 크게 신경을 쓰지 않고 호출할 수 있게 됨

1) 메서드 오버로딩의 조건

1. 메서드의 이름이 같아야 한다.
2. 매개변수의 개수 또는 타입이 달라야 한다.

메서드 오버로딩은 리턴 타입과는 관계가 없다. 만약 메서드의 이름과 파라미터는 같은데 리턴 타입만이 다른 경우에는 오버로딩이 성립하지 않는다.

2) 메서드 오버로딩의 예

println() 메서드

println() 메서드는 전달받는 매개변수의 타입에 따라 다음과 같이 다양한 원형 중에서 적절한 원형을 호출하게 된다.

```
System.out.println();
System.out.println(boolean x);
System.out.println(char x);
System.out.println(char[] x);
System.out.println(double x);
System.out.println(float x);
System.out.println(int x);
System.out.println(long x);
System.out.println(Object x);
System.out.println(String x);
```