JAVA
programming

데이터 타입 **변수 이름** = **초기화**;

데이터 타입
- int(정수), double(실수), Boolean(논리형), char(문자열) 등

변수의 이름 설정
- 영문으로 시작
- 영문, 숫자, 언더바(_), $ 사용 가능

변수의 선언
- 변수만 우선 선언 가능
- 선언과 동시에 초기화 가능

*** 변수 앞에 final을 붙이는 이유**
- 관리 용이 : 원하는 값을 상수로 고정시켜두면 값의 변경이 용이하다.
- 불변 객체 : 고정된 값이므로 새로운 값을 대입하거나 덧셈, 뺄셈 등이 불가하다.
  -> 원래 동작해야하는 동작의 무결성을 보장한다.

while (조건식) {수행문}

조건식이 ture이면 반복 수행, false이면 종료
- 조건의 결과나 변수가 논리형인 경우 주로 사용

무한반복
- while (true) {}

예시) num이 10이 될때까지 반복하여 num을 출력

```
int num = 1;
while (num <= 10) {
    num++;
    System.out.println(num);
}
```

for (초기화; 조건; 증감) {수행문}

초기화
- for문을 처음 만나는 순간에만 동작한다.

증감
- 변수의 값을 증가 혹은 감소시키는 것, 2차 반복부터 수행된다.

무한반복
- For (;;) {}

예시) num이 10이 될때까지 반복하여 num을 출력

```
for (int num = 1; num <= 10, num++) {
    System.out.println(num);
}
```

for문의 사용
- 특정 수의 범위, 횟수와 관련하여 반복될 때 주로 사용하며 배열과 함께 많이 사용된다.

switch (조건) {case의 조건과 수행문}

```
switch (num){
        case 0 :
        System.out.println("입금");
        break;

        case 1 :
        System.out.println("출금");
        break;

        default :
        System.out.println("none");
        break;
}
```

case
- 조건이 case에 해당하는 값과 일치할 때 코드를 수행

default
- case에 부합하는 값이 없을 때 수행

if (조건) {수행문}

```
if (score >= 90){
            System.out.println("A");
} else if {score < 90 && score >= 80
            System.out.println("B");
} else if {score < 80 && score >= 70
            System.out.println("C");
}
```

if / else if 구조

- if 조건 만족시 if 실행문을 수행하고,
- 아니라면 그 다음
- 첫번째 else if 조건 만족시 첫번째 else if 실행문을 수행
- 아니라면 그 다음
- 두번째 else if 조건 만족시 두번째 else if 실행문을 수행

if (조건) {수행문}

```
if (score >= 90){
            System.out.println("A");
}
if {score < 90 && score >= 80
            System.out.println("B");
}
if {score < 80 && score >= 70
            System.out.println("C");
}
```

if / if 구조

- 각 조건에 맞는 실행문을 수행
- if / else if 사용시
- 첫번째부터 두번째, 세번째 조건을 이어서 봐야한다.
- 깊이가 깊어질수록 코드를 파악하기 어렵다.

-> 따라서 직관적으로 볼 수 있는 if / if문을 사용하는 것이 좋다.

데이터 타입 [] 변수 이름 = {초기값};

배열
- 같은 타입의 데이터를 연속된 공간에 나열하고, 인덱스를 부여한다. 인덱스는 0부터 시작

데이터 타입 [] 변수 이름 = {1, 2, 3, 4, 5};
- 배열 선언 후 초기화
- Stack 메모리에 할당되며, 중괄호 내에서 사용된다.

데이터 타입 [] 변수 이름 = new 데이터 타입 [길이];
- 배열 선언
- Heap 메모리에 할당되며, new 이후 사용된다.
- 배열 선언 후 나중에 값이 결정되는 경우 사용한다.
- 예시) int [] diceArray = new int [5];
  diceArray라는 이름의 길이가 5인 배열을 선언

배열 이름.length
- 배열의 길이를 나타내는 것

객체 지향 프로그래밍(OOP)
: 특정한 속성을 갖는 독립적인 객체들을 상호작용하여 설계하는 방식

클래스
- 객체를 정의하는 설계도, 특정한 속성을 갖는 변수와 메소드의 집합

클래스의 생성
- class 클래스명 (){}

클래스의 구성 요소
- 필드 : 객체가 가지는 속성, 클래스 내부에서 사용하는 변수 등을 정리하는 곳
- 생성자 : 객체 생성시 초기화를 담당, 클래스명 public 클래스명 (){}
  () 외부에서 값을 입력받는 다면, 소괄호에 입력받을 형태를 작성
  {} new를 통해 객체화될 때 구동시키고 싶은 작업을 배치
- 메소드 : 특정 작업을 수행하기 위한 명령문, public 리턴 타입 메소드명 (){}
  리턴 타입 : int, double, string 등 / void : 리턴 값이 없다.
  {} 실제 메소드 이름에 해당하는 작업을 배치

객체 지향 프로그래밍(OOP)
: 특정한 속성을 갖는 독립적인 객체들을 상호작용하여 설계하는 방식

인스턴스(객체)
- 클래스로부터 만들어진 객체

인스턴스의 생성
- 클래스명 인스턴스명 = new 클래스명 ();

도메인 DDD(Domain Driven Design)
- 도메인 주도 설계
- 도메인 : 소프트웨어를 구현할 때 해결하고자 하는 문제 영역