즐거운 자바

강경미(carami@nate.com), 김성박 (urstory@gmail.com)

주석문

- 프로그래밍 실행과는 상관 없는 문장
- 프로그램에 설명을 붙이기 위해 사용

자바의 주석문

주석기호	설명
//	//부터 시작해서 줄 끝까지 주석처리된다.
/_ ..... _/	/_와 _/ 사이의 내용이 모두 주석처리 된다.
/*_ ..... _/	/*_ 와 _/ 사이의 내용이 모두 주석처리 된다. JavaDoc 주석문이라고 한다.

실습 1

```
public class Book {
    /*
        title
        price 를 필드로 선언하였습니다.
    */
    private String title;
    private int price; // field price

    // 필드의 값을 수정하고 얻기 위한 메소드를 만든다. setter, getter
    // setter, getter - 프로퍼티(property) - price 프로퍼티
    public int getPrice() {
        return this.price * 2; // this는 내 자신 인스턴스를 참조하는 예약어.
    }

    public void setPrice(int price) { // 지역변수 price
        this.price = price;
    }

    public String getName() {
        return title;
    }

    public void setName(String title) {
        this.title = title;
    }
}
```

JavaDoc 주석문에서 사용하는 태그들

annotaion	설명
@version	클래스나 메소드의 버전
@author	작성자
@deprecated	더이상 사용되지 않거나, 삭제될 예정
@since	언제 생성, 추가, 수정되었는가?
@see	외부 링크나 텍스트, 다른 필드나 메소드를 링크할 때 사용
@link	see와 동일한 기능. 링크 제공
@exception	발생할 수 있는 Exception 정의

기타 등등 있음.

실습 2

```
/**
 *  책 한권의 정보를 담기 위한 클래스
 *
 * @author urstory(<a href="mailto:urstory@gmail.com">김성박</a>)
 * @since 2022.03
 * @version 0.1
 *
 */
public class Book {
    private String title;
    private int price; // field price

    // 필드의 값을 수정하고 얻기 위한 메소드를 만든다. setter, getter
    // setter, getter - 프로퍼티(property) - price 프로퍼티
    public int getPrice() {
        return this.price * 2; // this는 내 자신 인스턴스를 참조하는 예약어.
    }

    public void setPrice(int price) { // 지역변수 price
        this.price = price;
    }

    /**
     * 책의 제목을 반환한다.
     * @return 책의제목
     */
    public String getName() {
        return title;
    }

    public void setName(String title) {
        this.title = title;
    }
}
```

IntelliJ 에서 JavaDoc 생성하기

- shift키를 2번 연속 누른다.
- generate javaDoc 을 입력한다.
- custom scope를 선택한 후 JavaDoc을 생성할 패키지, 클래스, 인터페이스 등을 선택한다. exclude를 선택해서 생성하지 않을 것들도 지정할 수 있다.
- output directory에서 JavaDoc이 생성할 경로를 지정한다.
- other command line arguments에는 다음을 입력한다.

```
-encoding UTF-8 -charset UTF-8 -docencoding UTF-8
```

- OK 버튼을 클릭한다.

JavaDoc을 명령으로 실행하기

- IntelliJ를 보면 아래와 같은 문장이 실행된 것을 알 수 있다.

```
javadoc -protected -splitindex -encoding
UTF-8 -charset UTF-8 -docencoding UTF-8 -d
/Users/urstory/Documents/MyJavaDoc
@/private/var/folders/kp/gbjcdk810dv1tyv7_dmm1vh00000gn
/T/javadoc_args
```

주석문은 잘 작성하는 방법은?

- 주석문이 없어도 이해할 수 있도록 클래스,메소드,변수 이름을 작성한다.
- 주석문은 최소한으로 작성한다.
- JavaDoc주석문을 잘 작성한다.

코드는 한줄씩 차례대로 실행된다.

이름, 이메일, 성별 출력하기

1. 다음과 같은 MyProfile 클래스를 작성한다. (본인의 이름, 이메일주소, 성별로 바꿔보세요.)

```
public class MyProfile {
    public static void main(String[] args){
        System.out.println("김성박");
        System.out.println("urstory@gmail.com");
        System.out.println("남자");
    }
}
```

2. 실행한다.

```
김성박
urstory@gmail.com
남자
```

main()메소드 안의 내용은 차례대로 실행된다.

- main() 메소드 안의 내용이 한줄씩 한줄씩 실행된다.
- System.out.println("김성박");
  - 위의 코드는 "김성박"이라는 이름을 출력한다.
  - 그리고 줄바꿈을 하게 된다. 그렇기 때문에 다음 줄에 이메일이 출력된다.

println()메소드를 print()로 바꿔보기

```
public class MyProfile {
    public static void main(String[] args){
        System.out.print("김성박");
        System.out.print("urstory@gmail.com");
        System.out.print("남자");
    }
}
```

- 위와 같이 코드를 수정한 후 실행한 결과는 다음과 같다.

```
김성박urstory@gmail.com남자
```

print()메소드는 줄바꿈을 하지 않는다.

- println()메소드는 괄호 안의 내용을 출력하고 줄바꿈을 하지만, print()메소드는 괄호 안의 내용만 출력한다. 줄바꿈을 하지 않는다.
- `System.out.println();` 는 아무것도 출력하지 않고 줄바꿈만 한다.

연습

- 다음과 같은 결과를 출력하는 Rectangle클래스를 작성한다. (너비가 별 10개, 높이가 별 10개인 사각형)

```
**********
**********
**********
**********
**********
**********
**********
**********
**********
**********
**********
```

Rectangle 클래스

- 너비가 별 10개, 높이가 별 10개인 사각형을 출력한다.

```
public class Rectangle {
    public static void main(String[] args){
        System.out.println("**********");
        System.out.println("**********");
        System.out.println("**********");
        System.out.println("**********");
        System.out.println("**********");
        System.out.println("**********");
        System.out.println("**********");
        System.out.println("**********");
        System.out.println("**********");
        System.out.println("**********");
    }
}
```

높이가 200인 사각형을 출력하려면?

- 만약 높이가 10,000인 사각형을 출력하라고 문제가 바뀐다면?
- 컴퓨터가 가장 잘하는 일은 반복하는 일이다.

while반복문을 이용해 높이가 10인 사각형 출력하기

- 다음과 같이 Rectangle2 클래스를 작성한다.

```
public class Rectangle2 {
    public static void main(String[] args){
        int i = 1;
        while(i <= 10){
            System.out.println("**********");
            i = i + 1;
        }
    }
}
```

- 실행하면? Rectangle클래스와 같다는 것을 알 수 있다.

while 반복문

- main메소드안의 내용은 한줄씩 실행해 나간다. JVM이 한줄씩 읽어나가면서 해석하고 실행한다. 우리도 한줄씩 코드를 읽어보도록 하자.
- 정수 타입(type) 변수 i에 1을 저장한다. 자바 프로그래밍에서 = 는 `같다.`라는 의미가 아니라, 우측의 값을 좌측에 저장한다는 의미이다.

  `int i = 1`
- while(i <=10)은 i가 10보다 작거나 같을때까지 중괄호 안의 내용을 반복하라는 의미이다. 이 부분을 while블록(block)이라고 읽는다. 중괄호 부분을 블록이라고 한다. 즉, i가 10보다 크다면 반복하지 말라는 이야기이다.
- 괄호안의 `i = i + 1`은 i에 저장된 값에 1을 더한 후 다시 i에 저장하라는 뜻이다.

```
while(i <= 10){
......
    i = i+1;
}
```

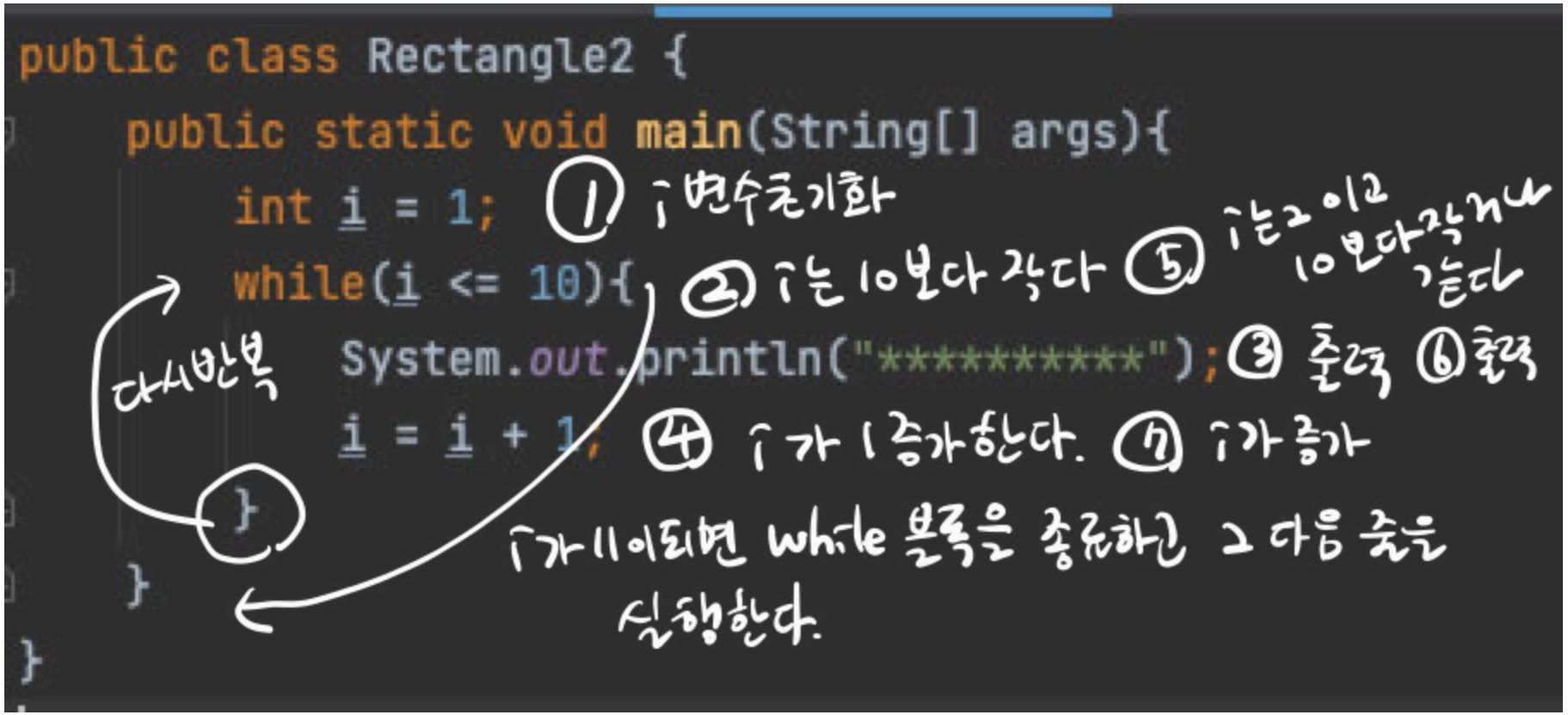
while 반복문

- 즉 아래의 코드는 i는 1부터 시작해서 while블록 안의 내용을 반복할 때마다 i를 1씩 증가시키라는 의미이다.
- 이때 i가 10보다 작거나 같을 경우는 계속 반복하게 된다. 어느 시점에 i가 11이 되는 경우가 발생하는데, 이때는 while블록을 종료하게 되고, 그 다음 줄을 실행하게 된다.

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

while 반복문

- 위에서 설명한 코드를 그림으로 표현하였다.



연습

- 높이가 1000인 사각형을 출력하자.

응용

- *(에스터리스크, Asterisk, 키보드 자판상으로 숫자 8 위)를 연속으로 20개 출력하는 AsteriskLine클래스 작성하기

```
public class AsteriskLine {
    public static void main(String[] args){
        int i = 1;
        while(i <= 20){
            System.out.print("*");
            i = i + 1;
        }
    }
}
```

실행 결과

```
********************
```

생각해보기

- while블록 안에는 while블록이 있을 수 있다.
- 아래와 같은 결과를 출력하려면 어떻게 해야할까?

```
*
**
***
****
*****
```

- 만약 높이가 100인 삼각형을 출력하라고 하면 어떻게 할까?
- 높이가 10,000 이라면?
- Hint : `System.out.print("*");`, `System.out.println()` 과 2개의 while블록을 이용해 문제를 해결할 수 있다.

변수와 리터럴

int i = 1;

- "한줄씩 실행되는 코드"에서 위와 같은 코드가 등장했다.
- 해당 코드의 구성 요소에 대해서 설명하면 다음과 같다.
  - int : 정수 타입(type)을 나타내는 키워드(keyword). 키워드란 java언어에서 정한 예약어를 말한다. 앞에서 등장했던 class, public, static, while 과 같은 단어들을 키워드라고 한다.
  - i : "변수 i" 라고 말한다. 변수는 `하나의 값을 가질 수 있는 공간` 이라고 말한다. 여기서 공간이란 메모리의 어떤 영역을 말한다. i는 변수의 이름, 즉 `변수명` 이라고 말합니다.
  - 1 : "정수 리터럴(literal) 1" 이라고 말한다. 리터럴이란 변수에 입력되는 값을 말한다.
- `int i = 1` ; 은 정수 타입 변수 i를 선언함과 동시에 1로 초기화 하였다고 한다.

메모리로 표현한 `int i = 1;`

- 정수 타입은 4byte의 메모리를 사용합니다. 이 4byte메모리에 숫자 1이 2진수로 저장된다.

1byte	2byte	3 byte	4 byte
00000000	00000000	00000000	00000001

변수명은 아무 이름이나 될 수 없습니다.

- 하나 이상의 글자로 이루어져야 한다.
- 첫 번째 글자는 문자 이거나 '$', '_'이어야 한다.
- 두번재 이후의 글자는 숫자, 문자, '$', '_'이어야 한다.
- '$', '_' 이외의 특수문자 사용은 불가능하다.
- 길이 제한이 없다.
- 키워드는 변수명으로 사용할 수 없다.
- 상수 값을 표현하는 단어 true, false, null은 변수명으로 사용할 수 없다.

다음 중 변수 명으로 사용할 수 있는 것은?

1. abc
2. $abc
3. _abc
4. @abc
5. abcdefghijklmnopqrstuvwxyz
6. while
7. public
8. true

정수타입 변수 x, y, z를 선언하려면?

```
int x;
int y;
int z
```

또는

```
int x , y, z
```

- 정수 타입 변수를 선언할 때 초기값을 넣어주지 않으면 모두 0을 가지게 된다.

정수타입 변수 x, y, z를 선언하는데 각각 5, 10, 15 값을 가지도록 선언한다.

```
int x = 5;
int y = 10;
int z = 15;
```

또는

```
int x = 5, y = 10, z = 15;
```

정수 타입의 변수를 선언하고 출력하기

- System.out.println(정수); 는 정수값을 화면에 출력한다.
- IntPrint클래스를 다음과 같이 작성한 후 실행한다.

```
public class IntPrint {
    public static void main(String[] args){
        int i = 1;
        System.out.println(i);

        System.out.println(100);
    }
}
```

- 실행결과

```
1
100
```

정수 타입의 변수를 선언하고 출력하기

- 정수타입 변수 i를 선언하고 정수 리터럴 1을 저장한다. 간단하게 변수 i에 정수 1을 저장한다고 말해도 된다.
- `System.out.println(i);` 는 i에 저장된 값을 출력한다. 즉 화면에 정수 1이 출력된다.

```
        int i = 1;
        System.out.println(i);
```

- 정수 리터럴 100을 화면에 출력한다.

```
    System.out.println(100);
```

연습

- 정수 1부터 100까지 출력하시오.
- 힌트 : 앞에서 배운 while문장을 이용한다.

논리형 타입과 연산자 (비교, 논리)

자바의 타입에는 `기본형 타입(primitive type)`과 `레퍼런스 타입(reference type)` 2가지 종류가 있다.

- 기본형 타입은 논리형, 문자형, 정수형, 실수형이 존재한다.
- 기본형 타입 중 논리형 타입은 boolean 키워드를 사용한다.
- 기본형 타입 중 정수형 타입은 byte, short, int, long 키워드를 사용한다.
- 기본형 타입 중 실수형 타입은 float, double 키워드를 사용한다.
- 기본형 타입 중 문자형 타입은 char 키워드를 사용한다.
- 이를 제외한 모든 타입은 레퍼런스 타입이다.

참인가? 거짓인가?

- 프로그래밍은 어떤 조건에 따라서 다르게 동작하게 코드를 작성하는 경우가 많다.
- 온도가 30도 이상일 때 동작한다. 가격이 5000원 이하일 때 동작한다. 등의 표현을 보자. 여기에서 "30도 이상", "5000원 이하"가 조건이라고 말할 수 있다. 이러한 조건의 결과는 참(true)이나 거짓(false)이 나오게 된다.
- 이러한 참이나 거짓을 저장하는 변수를 논리형 타입 변수라고 한다.

논리형 타입 boolean 사용하기

- boolean 타입은 true, false 2가지 값중에 하나를 가질 수 있다.
- 초기화 하지 않으면 기본적으로 false값을 가진다. 참고로 메소드 안에서는 변수를 선언한 후 초기화하지 않고 사용하면 `java: variable 변수명 might not have been initialized` 와 같은 오류가 발생한다.

논리형 타입은 언제 사용할까?

- 논리형 타입 변수는 논리연산의 결과를 저장할 때 사용한다.
- 논리연산이란 무엇일까? 예를 들어 `a라는 변수가 10보다 크고 30보다 작을 경우` 라는 조건이 있을 때, 이러한 조건의 결과가 저장되는 변수가 논리형 타입 변수이다.
- a가 만약 20이라면 true가 되겠고, a가 5라면 false가 위의 조건의 결과가 된다.

참(true)과 거짓(false)이 나오는 식

- 비교연산자와 논리연산자를 이용한 식의 결과는 참이나 거짓이 결과로 나온다.
- 비교 연산자 : == , != , < , > , <= , >=
- 논리 연산자 : && , || , & , | , ^ , !
- 비교 연산자, 논리 연산자와 함께 산술 연산자도 함께 사용될 수 있다. 산술 연산자는 덧셈, 뺄셈등을 말한다. 다음에 설명한다.

비교 연산자란?

- 10은 5보다 크다. 참인가? 거짓인가? 자바에서 표현하면 다음과 같다.

```
10 > 5
```

- 위의 식의 결과를 변수에 저장하고 싶다면? 아래와 같이 표현한다. flag는 논리형 타입 변수로 false로 초기화 된다. 그리고 나서 10이 5보다 크다가 참이라면 flag는 true 값을 가지게 된다.

```
boolean flag = false;
flag = 10 > 5;
```

예제 1

- 실행해 보기전에 결과를 예상해보자. System.out.println(boolean)은 논리형 값을 출력한다.

```
public class BooleanTest1 {
    public static void main(String[] args){
        boolean flag1 = false;
        boolean flag2 = false;
        boolean flag3 = false;
        boolean flag4 = false;
        boolean flag5 = false;

        flag1 = 10 > 5;
        flag2 = 10 < 5;
        flag3 = 10 >= 10;
        flag4 = 10 <= 10;
        flag5 = 10 == 10;

        System.out.println(flag1);
        System.out.println(flag2);
        System.out.println(flag3);
        System.out.println(flag4);
        System.out.println(flag5);
    }
}
```

논리연산자 and(&&) 와 or (||)

- `a는 20보다 크고, a는 50보다 작다.` 라는 표현은 자바에서 다음과 같이 표현한다.
  `a > 20 && a < 50`
- 'a는 20보다 크거나, a는 50보다 작다. `라는 표현은 자바에서 다음과 같이 표현한다.`
  a > 20 || a < 50`
- 2가지 조건을 모두 만족해야 하는 경우엔 and연산자인 &&를 사용하고, 2가지 조건 중 하나만 만족해도 될 경우에는 || 를 사용한다.
- && 대신에 &를 || 대신에 | 를 사용해도 같은 결과가 나온다. && 와 & , ||와 | 의 차이는 다음에 설명한다.

논리 연산자 ^ (exclusive-or 또는 XOR 라고 말한다.)

- 2개의 식의 논리 값이 서로 다를 경우 참이 된다. 아래의 식이 참이려면 한쪽은 참이고, 한쪽은 거짓이 나와야 한다. 예를 들어 a가 31이라는 값을 가질 경우 `true & false` 가 되니 이 때 결과는 참이 나온다.

  ```
  a > 10 ^ a < 20
  ```

부정 연산자 !

- 논리형 값을 부정한다. true는 false로 바꾸고, flase는 true로 바꾼다. 10은 5보다 크다는 참이다. 하지만 그 결과를 부정하게 되니 논리형 변수 a에는 false가 저장되게 된다.

```
boolean a = !(10 > 5);
```

논리 연산자 정리

x	y	X && y	x & y	X || y	x | y	x ^ y	!x
true	true	true	true	true	true	false	false
true	false	false	false	true	true	true	false
false	true	false	false	true	true	true	true
false	false	false	false	false	false	false	true

예제 2

```
public class BooleanTest2 {
    public static void main(String[] args){
        boolean flag1 = false;
        boolean flag2 = false;
        boolean flag3 = false;
        boolean flag4 = false;
        boolean flag5 = false;
        boolean flag6 = false;

        flag1 = 10 > 5 && 5 < 20;
        flag2 = 10 > 5 & 5 < 20;
        flag3 = 10 >= 10 || 5 > 6;
        flag4 = 10 >= 10 | 5 > 6;
        flag5 = 10 == 10 ^ 5 == 4;
        flag6 = !flag5;

        System.out.println(flag1);
        System.out.println(flag2);
        System.out.println(flag3);
        System.out.println(flag4);
        System.out.println(flag5);
        System.out.println(flag6);
    }
}
```

불린형 타입은 메모리를 얼마나 사용할까?

- 불린형 타입은 1byte(8bit)를 사용한다. 사실 1비트로도 참과 거짓은 표현할 수 있다. 0은 거짓, 1은 참으로 표현하면 된다. 하지만 컴퓨터는 자료를 표현하는 최소 단위가 1byte이다. 그렇기 때문에 불린형 타입은 메모리에 1byte를 사용하게 된다.

정수와 실수 그리고 산술 연산자

정수형 타입 byte, short, int, long

- byte타입은 1byte크기의 정수 값을 가질 수 있다.
- short타입은 2byte크기의 정수 값을 가질 수 있다.
- int타입은 4byte크기의 정수값을 가질 수 있다.
- long타입은 8byte크기의 정수값을 가질 수 있다.
- 리터럴 값인 숫자 5는 int타입이다.
- 리터럴 값인 숫자 5L은 long타입이다. (숫자 뒤에 L또는 l이 붙는다.)

각 타입별 값의 범위

정수형 타입	크기	값의 범위
byte	1바이트	-128 ~ 127
short	2바이트	$-2^{15} \sim (2^{15} - 1)$
		-32,768 ~ 32,767
int	4바이트	$-2^{31} \sim (2^{31} - 1)$
		-2,147,483,648 ~ 2,147,483,647
long	8바이트	$-2^{63} \sim (2^{63} - 1)$
		-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807

실수형 타입 float, double

- float타입은 4byte크기의 실수 값을 가질 수 있다.
- double타입은 8byte크기의 실수 값을 가질 수 있다.
- 리터럴 값인 숫자 5.2는 double타입이다.
- 리터럴 값인 숫자 5.2f는 float타입이다. (숫자 뒤에 F또는 f가 붙는다.)

각 타입별 값의 범위

실수형 타입	크기	값의 범위	리터럴 타입 접미사
float	4바이트	$(3.4 \times 10^{-38}) \sim (3.4 \times 10^{38})$	F 또는 f
double	8바이트	$(1.7 \times 10^{-308}) \sim (1.7 \times 10^{308})$	D 또는 d (생략 가능함)

float와 double의 유효 자릿수 차이

실수형 타입	지수의 길이	가수의 길이	유효 자릿수
float	8 비트	23 비트	소수 부분 6자리까지 오차없이 표현할 수 있음.
double	11 비트	52 비트	소수 부분 15자리까지 오차없이 표현할 수 있음.

산술 연산자

- 정수와 실수는 덧셈(+), 뺄셈(-), 곱셈(*), 나눗셈(/), 나머지구하기(%)를 할 수 있다.

```
a = 5 + 4;
b = 4 - 3;
c = 3 * 2;
d = 5 / 2;
e = 4 % 3;
```

- a는 9, b는 1, c는 6, d는 2, e는 1의 값을 가진다.
- 정수를 정수로 나누면 잘림 현상이 발생한다. 2.5가 아닌 2가 d에 저장된다.
- 나머지 연산자는 앞의 숫자를 나누고 나눈 나머지 값을 구한다.

증가 연산자와 감소 연산자

- 증가 연산자(++)를 사용하면 1이 증가된다.
- 감소 연산자(--)를 사용하면 1이 감소된다.
- 증가 연산자와 감소 연산자는 변수의 앞에 붙으면 전위 증가 연산자와 전위 감소 연산자라 부르고, 변수의 뒤에 붙으면 후위 증가 연산자와 후위 감소 연산자라 말한다.

  `++a , a++, --a, a--`

```
a = 5;
a++;

b = 4;
b--;
```

- a++는 `a = a + 1`과 같은 뜻이다. a는 6이 된다.
- b--는 `a = a - 1`과 같은 뜻이다. b는 3이 된다.

산술 대입 연산자

- 산술 대입 연산자는 +=, -=, *=, /=, %= 가 있다.

```
a = 5;
a += 3;

b = 4;
b -= 2;

c = 8
c *= 2;

d = 4
d /= 2;

e = 5
e %= 2;
```

- `a += 3`은 `a = a + 3`을 줄인 식이다. a는 8이 된다.
- `b -= 2`는 `b = b -2`를 줄인 식이다. 나머지도 결과를 생각해보자.

괄호 ()

- 괄호가 있을 경우 괄호 안의 내용부터 계산한다.

```
a = 3;
b = 5;

c = a * ( b + 5 );
d = a * b + 5;
```

- c와 d에 저장된 값은?

정수와 실수의 최솟값과 최댓값

- int타입과 double타입이 표현할 수있는 최댓값과 최솟값을 출력한다.

```
public class NumberExam01 {
    public static void main(String[] args){
        int maxInt = Integer.MAX_VALUE;
        int minInt = Integer.MIN_VALUE;

        double maxDouble = Double.MAX_VALUE;
        double minDouble = Double.MIN_VALUE;

        System.out.println(maxInt);
        System.out.println(minInt);

        System.out.println(maxDouble);
        System.out.println(minDouble);
    }
}
```

오버플로우(overflow)

- 아래의 프로그램은 결과가 얼마가 나올까?
- 계산 결과가 최댓값을 넘거나, 최솟값보다 작을 경우 음수는 양수로, 양수는 음수로 바뀌는 문제가 발생한다. 이를 오버플로우라고 한다.

```
public class NumberOverflow {
    public static void main(String[] args){
        int value = 10;
        int maxInt = Integer.MAX_VALUE;

        System.out.println(value + 1);
        System.out.println(maxInt + 1);
    }
}
```

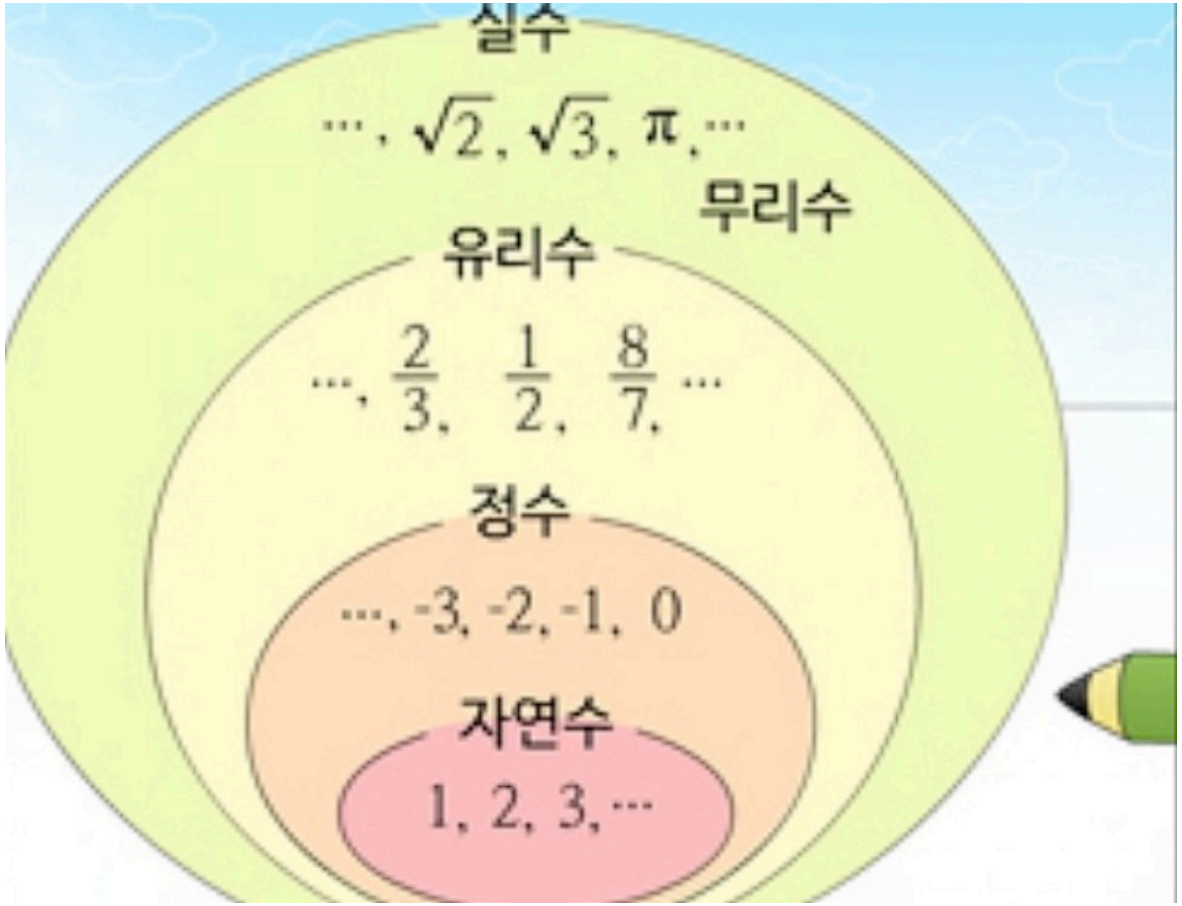
생각해볼 문제

- 정수와 실수는 타입별로 메모리에 어떻게 저장될까?
- 메모리에 저장되는 형태를 알아야 오버플로우에 대해 이해할 수 있다. `자료구조` 라는 내용을 학습하면 이러한 데이터 타입에 대한 저장형식에 대해서 배우게 되는데, 비전공자도 학습을 잘 이해하기 위해서는 이러한 내용을 별도로 공부할 필요가 있다. 하지만, 이 부분이 너무 어렵게 느껴지면 프로그래밍의 흥미를 잃을 수 있다. 그렇기 때문에 이 설명은 건너뛰려고 한다.

타입의 변환

수학시간에 배웠던 내용을 기억해보자.

- 실수는 정수를 포함한다.



dobule형 타입은 정수값이 잘 대입된다.

- 다음은 가능하다.
- 이를 묵시적 타입 변환(자동 타입 변환, implicit conversion)이라고 한다.

```
double d1 = 50;
double d2 = 500L;
```

- int형 리터럴 50, long형 리터럴 500L이 모두 d1, d2에 저장된다.

예제 1

- double타입의 변수에 정수가 잘 저장된다.

```
public class PrimitiveCastExam1 {
    public static void main(String[] args){
        double d1 = 50;
        double d2 = 500L;

        System.out.println(d1);
        System.out.println(d2);
    }
}
```

int형 타입에 실수를 대입하면 오류가 발생한다.

- 실수는 정수를 포함하지만, 정수는 실수를 포함할 수 없기 때문에 아래의 코드는 컴파일 오류가 발생한다.

```
int i1 = 50.0;
int i2 = 25.4f;
```

실수 값을 정수에 저장하려면 형 변환을 해야한다.

- 실수 값을 정수타입의 변수에 저장하려면 정수타입으로 형변환 해야한다. 변환하고자 하는 값 앞에 (int)를 붙인다.
- 주의해야할 점은 소수점 이하 부분은 잘린다.
- 이를 명시적 타입 변환(강제 타입 변환, explicit conversion)이라고 한다.

```
int i1 = (int)50.0;
int i2 = (int)25.6f;
```

예제 2

```
public class PrimitiveCastExam2 {
    public static void main(String[] args){
        int i1 = (int)50.0;
        int i2 = (int)25.6f;

        System.out.println(i1);
        System.out.println(i2);
    }
}
```

크기가 큰 타입은 작은 타입을 저장할 수 있다.

- long 타입의 변수는 byte, short, int 타입의 값을 저장할 수 있다.
- int 타입의 변수는 byte, short타입의 값을 저장할 수 있다.
- short 타입의 변수는 byte타입의 값을 저장할 수 있다.

```
short s = 5;
int i = s;
long x = i;

System.out.println(i);
System.out.println(x);
```

- 오류가 발생하지 않고 잘 실행된다.

정수형의 자동 타입 변환

byte형 → short형 → int형 → long형 → float형 → double형

크기가 작은 타입은 큰 타입을 저장할 수 없다.

```
        long x2 = 50;
        int i2 = x2;
```

- `int i2 = x2;` 부분에서 컴파일 오류가 발생한다.

크기가 큰 타입을 작은 타입에 저장하려면 형 변환을 해야한다.

```
long x2 = 50;
int i2 = (int)x2;
```

예제 : 형변환시 주의할 점

- 크기가 큰 타입을 작은 타입에 저장할 때는 오버플로우를 조심해야 한다.

```
public class PrimitiveCastExam4 {
    public static void main(String[] args){
        long x2 = 50;
        int i2 = (int)x2;
        System.out.println(x2);
        System.out.println(i2);

        long x3 = Long.MAX_VALUE;
        int i3 = (int)x3;
        System.out.println(x3);
        System.out.println(i3);
    }
}
```

구글에서 `JAVA에서 강제 형변환시 주의할 점`이라고 검색해보자.

문자(char) 타입

문자타입

- 문자는 작은 따옴표로 묶인 문자 하나를 말한다.
- 문자는 2byte크기를 가지며 유니코드 값을 가진다.
- 아래의 위키 페이지를 보면 유니코드 값을 확인할 수 있다.
    - https://ko.wikipedia.org/wiki/유니코드_0000~0FFF

유니코드표 보기

- 16진수 0041이 문자 A를 표현한다.
- 16*4 + 1 = 65

C0 Controls and Basic Latin (로마자 기본) [편집]

이 부분의 본문은 로마자 기본입니다.

U+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
0010	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
0020		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0030	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0040	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0050	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0060	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0070	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

예제

```
public class CharExam1 {
    public static void main(String[] args){
        char c1 = 'a';
        char c2 = 'z';
        char h1 = '한';
        char h2 = '글';

        System.out.println(c1);
        System.out.println(c2);
        System.out.println(h1);
        System.out.println(h2);
    }
}
```

문자 타입은 정수 타입이기도 하다.

- 문자 타입은 0부터 65535까지 저장할 수 있는 정수 타입이기도 하다.

예제 - 문자를 정수형으로, 정수를 문자로 변환

- 유니코드 97번째 값은 문자 'a'이다. (유니코드 표를 확하자.)

```
public class CharExam2 {
    public static void main(String[] args){
        char c1 = 'a';

        System.out.println((int)c1);

        char c2 = (char)97;

        System.out.println(c2);
    }
}
```

비트 연산자

비트(bit)와 바이트(byte)

- 비트는 컴퓨터가 처리하는 정보의 최소 단위인데, 한 개만으로는 많은 양의 데이터를 나타내기에 턱없이 부족하기 때문에 정보를 표현하는 기본단위로는 8개의 비트를 묶은 바이트(Byte)를 사용한다.
- 1byte는 `00000000` 부터 `11111111` 까지 값을 표현할 수 있다.
- 1byte는 정수로 표현하면 0부터 254까지 표현가능하다.
- 1byte를 16진수로 표현하면 `00` 부터 `FF` 까지 표현가능하다. 4비트는 0부터 15까지 표현가능하기 때문이다.

비트 연산자는 논리 연산자와 비슷하지만, 비트(bit) 단위로 논리 연산을 할 때 사용하는 연산자이다.

- 비트 연산자의 종류로는 &, |, ^ , ~ , << , >>, >>> 이 있다.
- &는 and, | 는 or, ^는 xor, ~는 not을 의미한다.
- << 는 좌측 시프트(shift), >>는 우측 시프트(shift), >>>는 우측 양수화 시프트라고 한다.
- &, |, ^ 는 논리연산자로도 사용된다.

and, or, xor, not 비트 연산 결과

AND

대응되는 비트가 1로 같으면 1

A	B	A & B
1	1	1
1	0	0
0	1	0
0	0	0

OR

대응되는 비트 중에서 하나라도 1이면 1

A	B	A | B
1	1	1
1	0	1
0	1	1
0	0	0

XOR

대응되는 비트가 서로 다르면 1

A	B	A ^ B
1	1	0
1	0	1
0	1	1
0	0	0

NOT

비트를 반전함

A	~A
1	0
0	1

비트연산자는 바이트를 구성하고 있는 bit를 연산하는 연산자이다.

```
   00000000
&  11111111
------------
   00000000
```

- 두개의 바이트에 각각 `00000000`과 `11111111`이 저장되어 있을 때 2개의 바이트에 대한 and(&)연산의 결과는? 각각의 자리수에 해당하는 비트끼리 and연산을 한 결과가 구해진다.

>> 와 <<

- `<<` 는 명시된 수만큼 비트들을 전부 왼쪽으로 이동시킨다.
- `>>` 는 부호를 유지하면서 지정한 수만큼 비트를 전부 오른쪽으로 이동시킨다.
    - 정수형 타입을 비트로 표현했을 때, 맨 좌측의 비트를 부호화 비트라고 한다. 맨 좌측의 비트가 1이면 음수, 0이면 양수를 나타낸다.

예제 shift연산자

```
public class BitOperatorExam1 {
    public static void main(String[] args){
        int a = 4;
        int b = a >> 1;
        System.out.println(b);

        int c = 4;
        int d = c << 1;
        System.out.println(d);
    }
}
```

왜 결과가 저렇게 나올까?

```
00000000 00000000 00000000 00000100   (int 타입 4를 bit로 표현)
 >> 1 을 하게 되면 우측으로 비트가 하나씩 밀리고, 맨 좌측엔 0이 채워진다.
 00000000 00000000 00000000 00000010
```

```
00000000 00000000 00000000 00000100   (int 타입 4를 bit로 표현)
 << 1 을 하게 되면 좌측으로 비트가 하나씩 밀리고, 맨 좌측 비트는 사라지고 우측엔 0이 채워진
 00000000 00000000 00000000 00001000
```

- 시프트하는 숫자의 값을 n이라고 한다면, 좌측시프트는 2^n으로 나눈 결과가, 우측시프트는 2^n으로 곱한 결과가 나온다.

>>> 는 지정한 수만큼 비트를 전부 오른쪽으로 이동시키며, 새로운 비트는 전부 0이 된다.

- `>>>` 는 그 결과 무조건 양수가 나온다.
- `10000000` >>> 2 를 하게 되면 `00100000` 가 된다.
- `10000000` 는 음수, `00100000` 는 양수이다.

생각해볼 문제

- 1byte에 `10000000` 이 저장되어 있다. 정수 값 얼마인가?
- 정수가 어떻게 저장되는지 알아야 한다. 너무 어려우면 건너뛰는 것도 공부 방법이다.

삼항 연산자와 instanceof 연산자

삼항 연산자는 조건문을 배울때 배운다.

- 자바의 유일한 삼항 연산자.

```
항(項)은 수학에서 쓰인다.
항은 다항식을 이루는 각각의 단항식이다.
항은 분수의 분모, 분자이다.
항은 비례식의 각 부분이다.
항은 수열 또는 급수를 이루는 각 수다.
```

instanceof 연산자는 '상속'문법을 배울때 배운다.

- 영어로 되어있지만 덧셈(+), 뺄셈(-)처럼 연산자의 한 종류이다.

조건문 if

if

- if는 제어문(control flow statements) 중에 하나이다. 순차적인 흐름안에서 조건에 따라 제어를 할 필요가 있을 경우 if를 사용한다.

if 첫번째 사용법

- 중괄호 안의 내용을 블록이라고 한다.

```
if(조건문){
    조건문이 참일 경우 실행되는 블록
}
```

예제 1

```
public class IfExam1 {
    public static void main(String[] args){
        int a = 5;

        if(a > 4){
            System.out.println("a는 4보다 큽니다.");
        }
    }
}
```

if 두번째 사용법

```
if(조건문){
    조건문이 참일 경우 실행되는 블록
}else{
    조건문이 거짓일 경우 실행되는 블록
}
```

예제 2

- a의 값을 변경하면서 실행한다.

```
public class IfExam2 {
    public static void main(String[] args){
        int a = 3;

        if(a > 4){
            System.out.println("a는 4보다 큽니다.");
        }else{
            System.out.println("a는 4이하입니다.");
        }
    }
}
```

if 세번째 사용법

- else if 는 여러줄 추가될 수 있다.

```
if(조건문1){
    조건문1이 참일 경우 실행되는 블록
}else if(조건문2){
    조건문2가 참일 경우 실행되는 블록
}else{
    조건문1이나 조건문2에 해당되지 않을 경우 실행되는 블록
}
```

예제 3

```
public class IfExam3 {
    public static void main(String[] args){
        int score = 70;

        if(score >= 90){
            System.out.println("A");
        }else if(score >= 80){
            System.out.println("B");
        }else if(score >= 70){
            System.out.println("C");
        }else if(score >= 60){
            System.out.println("D");
        }else{
            System.out.println("F");
        }
    }
}
```

if문장에 중괄호가 없을 경우?

- if문장에 중괄호, 즉 블록이 없을 경우는 if문장 다음 문장만 조건에 만족할 경우 실행된다.
- "hello"는 무조건 출력된다. (들여쓰기를 잘못한 안좋은 코드의 예)

```
public class IfExam4 {
    public static void main(String[] args){
        int a = 10;

       if(a > 5)
           System.out.println("a는 10보다 큽니다.");
           System.out.println("hello");
    }
}
```

삼항연산자

- 자바에는 항이 3개인 연산자가 하나 있다. 조건식이 참일 경우 반환값 1이 사용되고, 거짓일 경우 반환값2가 사용된다.

```
조건식 ? 반환값1 : 반환값2
```

예 - 삼항연산자

- a의 값을 10, 4 등으로 바꿔가면서 실행해보자.

```
public class IfExam5 {
    public static void main(String[] args){
        int a = 10;

       int value = (a > 5) ? 20 : 30;

        System.out.println(value);
    }
}
```

조건문 switch

switch

- switch는 제어문(control flow statements) 중에 하나이다. switch문은 경우에 따라 if 문보다 가독성이 좋을 수 있다.
- 이론적으로는 switch문이 if문보다 속도가 빠르다고 하지만 의미가 없는 수준이다.

switch 사용법

- switch블록 안에는 여러개의 case가 올 수 있다.
- switch블록 안에는 하나의 default가 올 수 있다.
- break문은 생략할 수 있다.

```
switch (변수) {
    case 값1:
        변수가 값1일때 실행된다.
        break;
    case 값2:
        변수가 값2일때 실행된다.
        break;
    ...
    default:
        변수의 값이 어떤 case에도 해당되지 않을 경우 실행된다.
}
```

예제 1

- num의 값을 1,2,3,4 로 바꿔가면서 실행해보자.

```
public class SwitchExam1 {
    public static void main(String[] args){
        int num = 1;
        switch(num){
            case 1 :
                System.out.println("1입니다.");
            case 2 :
                System.out.println("2입니다.");
            case 3 :
                System.out.println("3입니다.");
        }
    }
}
```

예제 2

- num의 값을 1,2,3,4 로 바꿔가면서 실행해보자.

```
public class SwitchExam2 {
    public static void main(String[] args){
        int num = 1;
        switch(num){
            case 1 :
                System.out.println("1입니다.");
                break;
            case 2 :
                System.out.println("2입니다.");
                break;
            case 3 :
                System.out.println("3입니다.");
                break;
            default :
                System.out.println("1,2,3이 아닙니다.");
        }
    }
}
```

예제 3

- 변수 ch의 값을 'a', 'b', 'c', 'd', 'A', 'B', 'C', 'D'로 바꿔가면서 실행한다.

```
public class SwitchExam3 {
    public static void main(String[] args){
        char ch = 'a';
        switch(ch){
            case 'a' :
            case 'A' :
                System.out.println("A입니다.");
                break;
            case 'b' :
            case 'B' :
                System.out.println("B입니다.");
                break;
            case 'c' :
            case 'C' :
                System.out.println("C입니다.");
                break;
            default :
                System.out.println("A,B,C가 아닙니다.");
        }
    }
}
```

예제 4

- str의 값을 "감자", "고구마", "사과"로 변경하며 실행한다.
- JDK 7이상에서만 switch에서 String타입을 사용할 수 있다.

```
public class SwitchExam4 {
    public static void main(String[] args){
        String str = "감자";
        switch(str){
            case "감자" :
                System.out.println("감자입니다.");
                break;
            case "고구마" :
                System.out.println("고구마입니다.");
                break;
            default :
                System.out.println("감자와 고구마가 아닙니다.");
        }
    }
}
```

찾아보기

- JDK 14, JDK 17에서 switch와 관련된 새로운 문법이 추가되었다. 어떤 내용이 추가되었는가?

반복문 while

while

- while은 반복문(iteration statements) 중에 하나이다.
- 컴퓨터가 잘하는 일은 반복하면서 일을 처리하는 것이다.

while 사용법

- while문은 `탈출 조건식`이 false를 반환할 때 while문을 종료하게 된다.

```
변수의 초기화
while (탈출 조건식) {
    탈출 조건식이 참일 경우 실행되는 코드;
    변수의 증감식;
}
```

예제 1

- 1부터 5까지 출력하시오.

```
public class WhileExam1 {
    public static void main(String[] args){
        int i = 1;
        while(i <= 5){
            System.out.println(i);
            i++;
        }
    }
}
```

while문과 break

- while문에서 break를 만나면, 더이상 반복하지 않는다. break는 보통 조건문 if와 함께 사용된다.

예제 2

- while(true){ .... } 는 무한 루프(loop, 반복문)이라고 한다. 끝없이 반복한다.
- i가 11일 경우 while블록을 빠져나간다.

```
public class WhileExam2 {
    public static void main(String[] args){
        int i = 1;
        while(true){
            if(i == 11) break;
            System.out.println(i);
            i++;
        }
    }
}
```

while문과 후위 증감식

- 변수 뒤에 후위 증가식이 붙을 경우 변수가 사용된 이후에 값이 증가된다.
- i와 10이 비교를 한 후 i가 1증가한다.

```
public class WhileExam3 {
    public static void main(String[] args){
        int i = 0;
        while(i++ < 10){
            System.out.println(i);
        }
    }
}
```

while문과 continue

- while문에서 continue를 만나면, continue이하 문장을 실행하지 않고 반복한다.

예제 3

- 1부터 10사이에 있는 짝수만 출력하시오.

```
public class WhileExam4 {
    public static void main(String[] args){
        int i = 0;
        while(i++ < 10){
            if(i % 2 != 0)
                continue;
            System.out.println(i);
        }
    }
}
```

반복문 do / while

do / while

- do / while은 반복문(iteration statements) 중에 하나이다.
- do / while문은 while문과 비슷하지만, 무조건 한번은 실행된다는 특징이 있다.

do / while 사용법

- do/ while 문은 `탈출 조건식`이 false를 반환할 때 do / while문을 종료하게 된다.

```
변수의 초기화
do {
    탈출 조건식이 참일 경우 실행되는 코드;
    변수의 증감식;
}while (탈출 조건식);
```

예제 1

- 1부터 10까지 정수를 출력한다.

```
public class DoWhileExam1 {
    public static void main(String[] args){
        int i = 1;
        do{
            System.out.println(i);
            i++;
        }while(i <= 10);
    }
}
```

예제 2

- do / while 문장은 무조건 한번은 실행된다.

```
public class DoWhileExam2 {
    public static void main(String[] args){
        int i = 1;
        do{
            System.out.println(i);
            i++;
        }while(i < 1);
    }
}
```

반복문 for

for

- for문은 반복문(iteration statements) 중에 하나이다.
- while문은 변수 선언, 탈출 조건식, 증감식이 3줄로 이뤄지지만, for문은 한 줄에 모두 표현한다.

for 사용법

```
for (변수의 초기화; 탈출조건식 ; 증감식) {
    탈출 조건식이 참인 경우 실행되는 부분.
}
```

예제 1

- "*"을 10번 출력한다.

```
public class ForExam1 {
    public static void main(String[] args){
        for(int i = 0; i < 10; i++){
            System.out.println("*");
        }
    }
}
```

예제 2

- 1부터 10까지 출력한다.

```
public class ForExam2 {
    public static void main(String[] args){
        for(int i = 1; i <= 10; i++){
            System.out.println(i);
        }
    }
}
```

중첩 반복문

- 반복문 안에 조건문이 올 수 있는 것처럼, 반복문 안에 반복문이 올 수 있다.

중첩 반복문을 이용한 구구단 출력 1/4

- 문자열과 더해지면 문자열이 된다.
- 문자열 + 정수
  "hello" + 1 ---> "hello1"
- 문자열 + 불린
  "hello" + true ----> "hellotrue"
- 문자열 + 실수
  "hello" + 50.4 ----> hello50.4

중첩 반복문을 이용한 구구단 출력 2/4

```
public class StringExam1 {
    public static void main(String[] args){
        String str1 = "hello" + 1;
        String str2 = "hello" + true;
        String str3 = "hello" + 50.4;

        System.out.println(str1);
        System.out.println(str2);
        System.out.println(str3);
    }
}
```

중첩 반복문을 이용한 구구단 출력 3/4

```
public class Gugudan1 {
    public static void main(String[] args){
        for(int i = 1; i <= 9; i++){
            System.out.println( "1 * " + i + " = " + (1 * i));
        }
    }
}
```

중첩 반복문을 이용한 구구단 출력 4/4

```
public class Gugudan2 {
    public static void main(String[] args){
        for(int k = 1; k <= 9; k++) {
            for (int i = 1; i <= 9; i++) {
                System.out.println(k + " * " + i + " = " + (k * i));
            }
            System.out.println();
        }
    }
}
```

for each(Enhanced for) 문법은 배열을 배울때 설명한다.

반복문과 label

break와 continue문의 한계

- break는 현재 반복문을 빠져나가는데 사용한다.
- continue는 continue문 아래 부분을 실행하지 않고 다시 반복한다.
- 그렇다면 중첩반복문을 한번에 빠져나가려면? continue이하를 실행하지 않고 한번에 중첩 반복문을 반복하려면 어떻게 해야할까?
- 이럴때 label을 사용한다.

예제 1

```
public class LabelExam1 {
    public static void main(String[] args){
        outter:
        for(int i = 0; i< 3; i++){
            for(int k = 0; k < 3; k++){
                if( i == 0 && k == 2)
                    break outter;
                System.out.println(i + ", " + k);
            }
        }
    }
}
```

예제 2

```
public class LabelExam2 {
    public static void main(String[] args){
        outter:
        for(int i = 0; i< 3; i++){
            for(int k = 0; k < 3; k++){
                if( i == 0 && k == 2)
                    continue outter;
                System.out.println(i + ", " + k);
            }
        }
    }
}
```

배열

배열이란?

- 참조 타입
- 같은 타입의 변수가 여러개 필요할 때 사용한다.

기본형 배열

- 배열은 기본형 배열과 참조형 배열로 나뉜다.
- 기본형 배열이란 boolean, byte, short, char, int, long, float, double 타입의 변수를 여러개 선언할 필요가 있을 때 사용한다.

기본형 배열 선언

```
기본형타입[]  변수명;
기본형타입   변수명[];
```

예제

```
public class Array01 {
     public static void main(String[] args) {
          int[] array1;
          int array2[];
          int array3[];

          array1 = new int[5];
          array2 = new int[5];
          array3 = new int[0];
     }
 }
```

예제

```
public class Array02 {
     public static void main(String[] args) {
         int[] array1, array2;
         int array3[], array4;
     }
}
```

초깃값 있이 선언하기

```
기본형타입[] 변수명 = new 기본형타입[배열의크기];
변수명[index값] = 값;
기본형타입[] 변수명 = new 기본형타입[]{ 값1, 값2, .... };
기본형타입[] 변수명 = {값1, 값2, 값3.... };
```

예제

```
public class Array03 {
     public static void main(String[] args) {
         int[] array1 = new int[5];
         array1[0] = 1;
         array1[1] = 2;
         array1[2] = 3;
         array1[3] = 4;
         array1[4] = 5;
         int[] array2 = new int[]{1,2,3,4,5};
         int[] array3 = {1,2,3,4,5};

         System.out.println("array1 의 값 출력");
         for(int i = 0; i < 5; i++){
             System.out.println(array1[i]);
         }

         System.out.println("array2 의 값 출력");
         for(int i = 0; i < 5; i++){
             System.out.println(array2[i]);
         }

         System.out.println("array3 의 값 출력");
         for(int i = 0; i < 5; i++){
             System.out.println(array3[i]);
         }
     }
 }
```

참조형 배열

- 참조형 배열이란 배열의 타입이 기본형이 아닌 경우를 말한다.
- 배열 변수가 참조하는 배열의 공간이 값을 저장하는 것이 아니라 값을 참조한다는 것을 의미한다.

예제 1

```
public class ItemForArray {
    private int price;
    private String name;
    public ItemForArray(int price, String name){
        this.price = price;
        this.name = name;
    }

    public int getPrice() {
        return price;
    }

    public String getName() {
        return name;
    }
}
```

예제 1

```
public class Array04 {
    public static void main(String[] args) {
        ItemForArray[] array1;
        ItemForArray array2[];

        array1 = new ItemForArray[5];
        array2 = new ItemForArray[5];
    }
}
```

예제 2

```
public class Array05 {
    public static void main(String[] args) {
        ItemForArray[] array1 = new ItemForArray[3];
        array1[0] = new ItemForArray(500, "사과");
        array1[1] = new ItemForArray(300, "바나나");
        array1[2] = new ItemForArray(900, "수박");

        ItemForArray[] array2 = new ItemForArray[]{new ItemForArray(500, "사과"), new ItemForArray(300, "바나나"), new ItemForArray(900, "수박")};
        ItemForArray[] array3 = {new ItemForArray(500, "사과"), new ItemForArray(300, "바나나"), new ItemForArray(900, "수박")};

        System.out.println(array1[0].getName());
        System.out.println(array1[0].getPrice());
        System.out.println(array1[1].getName());
        System.out.println(array1[1].getPrice());
        System.out.println(array1[2].getName());
        System.out.println(array1[2].getPrice());
    }
}
```

배열의 길이 구하기

- 배열은 length 필드를 가진다.

```
public class Array06 {
    public static void main(String[] args) {
        double[] array1 = new double[5];
        double[] array2 = {1.5, 2.4, 3.5};
        double[] array3;
        double[] array4 = null;

        System.out.println(array1.length);
        System.out.println(array2.length);
//        System.out.println(array3.length);
//        System.out.println(array4.length);
    }
}
```

ArrayIndexOutOfBoundsException

```
public class Array07 {
    public static void main(String[] args) {
        double[] array = {1.5, 2.4, 3.5};

        System.out.println(array[0]);
        System.out.println(array[1]);
        System.out.println(array[2]);
        System.out.println(array[3]);
    }
}
```

이차원 배열

```
타입[][] 변수명 = new 타입[행의수][열의수];
변수명[행인덱스][열인덱스] = 값;
```

예제

```
public class Array08 {
    public static void main(String[] args) {
        int[][] array = new int[2][3];
        array[0][0] = 0;
        array[0][1] = 1;
        array[0][2] = 2;

        array[1][0] = 3;
        array[1][1] = 4;
        array[1][2] = 5;

        for(int i = 0; i < array.length; i++){
            for(int j = 0; j < array[i].length; j++){
                System.out.print(array[i][j] + "\t");
            }
            System.out.println();
        }
    }
}
```

이차원 배열 선언과 초기화

```
public class Array09 {
    public static void main(String[] args) {
        int[][] array = {{0,1,2}, {3,4,5}};

        for(int i = 0; i < array.length; i++){
            for(int j = 0; j < array[i].length; j++){
                System.out.print(array[i][j] + "\t");
            }
            System.out.println();
        }
    }
}
```

이차원 가변 배열의 선언과 초기화

```
타입[][] 변수명 = new 타입[행의수][];
변수명[행의인덱스] = new 타입[열의수];
```

이차원 가변 배열의 선언과 초기화

```
int[][] koreanScoreArray = new int[3][];
koreanScoreArray[0] = new int[20];
koreanScoreArray[0] = new int[19];
koreanScoreArray[0] = new int[21];
```

예

```
public class Array10 {
    public static void main(String[] args) {
        int[][] array = new int[2][];
        array[0] = new int[2];
        array[1] = new int[3];

        array[0][0] = 0;
        array[0][1] = 1;
        array[1][0] = 2;
        array[1][1] = 3;
        array[1][2] = 4;

        for(int i = 0; i < array.length; i++){
            for(int j = 0; j < array[i].length; j++){
                System.out.print(array[i][j] + "\t");
            }
            System.out.println();
        }
    }
}
```

예제

```
public class Array11 {
    public static void main(String[] args) {
        int[][] array = {{0,1}, {2,3,4}};

        for(int i = 0; i < array.length; i++){
            for(int j = 0; j < array[i].length; j++){
                System.out.print(array[i][j] + "\t");
            }
            System.out.println();
        }
    }
}
```

for each 문

```
for(타입 변수명 : 배열명){
    ........
}
```

예제

```
public class Array12 {
    public static void main(String[] args) {
        int[] array = {1,2,3,4,5};

        for(int i : array){
            System.out.println(i);
        }
    }
}
```

예제

```
public class Array13 {
    public static void main(String[] args) {
        ItemForArray[] array = new ItemForArray[3];
        array[0] = new ItemForArray(500, "사과");
        array[1] = new ItemForArray(300, "바나나");
        array[2] = new ItemForArray(900, "수박");

        for(ItemForArray item : array){
            System.out.println(item.getName());
            System.out.println(item.getPrice());
        }
    }
}
```

Arrays

- 배열을 다룰때 사용하는 유틸리티

예제

```
public class Array14 {
    public static void main(String[] args) {
        int[] copyFrom = {1,2,3};

        int[] copyTo = java.util.Arrays.copyOf(copyFrom, copyFrom.length);

        for(int c : copyTo){
            System.out.println(c);
        }
        System.out.println("----------------------------------");
        int[] copyTo2 = java.util.Arrays.copyOf(copyFrom, 5);

        for(int c : copyTo2){
            System.out.println(c);
        }
    }
}
```

예제

```
public class Array15 {
    public static void main(String[] args) {
        char[] copyFrom = {'h', 'e', 'l', 'l', 'o', '!'};

        char[] copyTo = java.util.Arrays.copyOfRange(copyFrom, 1, 3);

        for(char c : copyTo){
            System.out.println(c);
        }
    }
}
```

예제

```
import java.util.Arrays;

public class Array16 {
    public static void main(String[] args) {
        int[] array1 = {1,2,3,4,5};
        int[] array2 = {1,2,3,4,5};
        //int[] array2 = {1,2,3,4,6};
        //int[] array2 = {1,2,3,4,4};

        int compare = Arrays.compare(array1, array2);

        System.out.println(compare);
    }
}
```

예제

```
import java.util.Arrays;

public class Array17 {
    public static void main(String[] args) {
        int[] array = {5,4,3,1,2};

        Arrays.sort(array);

        for(int i : array){
            System.out.println(i);
        }
    }
}
```

예제

```
import java.util.Arrays;

public class Array18 {
    public static void main(String[] args) {
        int[] array = {5,4,3,1,2};

        Arrays.sort(array);

        int i = Arrays.binarySearch(array, 4);
        System.out.println(i);
    }
}
```

명령 행 아규먼트(Command-Line Arguments)

- 강좌에서 많이 사용된 배열이 무엇인지 물어본다면 그것은 바로 main메소드에 있는 String[] args
- main메소드는 JVM이 실행하는 메소드이다.
- JVM이 main메소드를 실행할 때 String[]을 아규먼트로 넘겨 준다는 것을 의미합한다.

예제

```
public class EmptyCommandLineArgumentExam {
    public static void main(String[] args){
        System.out.println(args.length);
    }
}
```

예제

```
public class CommandLineArgumentExam {
    public static void main(String[] args){
      if(args.length == 0){
          System.out.println("사용법 : CommandLineArgumentExam 값 값 ....");
          System.exit(0); // return; 으로 변경 가능
      }

      for(String arg : args){
          System.out.println(arg);
      }
    }
}
```

제한 없는 아규먼트(unlimited arguments)

- 경우에 따라서 메소드 아규먼트를 가변적으로 전달하고 싶은 경우가 있다.
- 메소드에 정수값을 경우에 따럿 3개, 어떤 경우엔 5개를 넘기고 싶다면 어떻게 해야 할까?

제한 없는 아규먼트(unlimited arguments)

```
리턴타입 메소드이름(타입... 변수명){
.......
}
```

예제

```
public class UnlimitiedArgumentsExam {
    public static void main(String[] args){
        System.out.println(sum(5,10));
        System.out.println(sum(1,2,4,2));
        System.out.println(sum(3,1,2,3,4,1));
    }

    public static int sum(int... args){
        System.out.println("print1 메소드 - args 길이 : " + args.length);
        int sum = 0;
        for(int i = 0; i < args.length; i++){
            sum += args[i];
        }
        return sum;
    }
}
```

감사합니다.