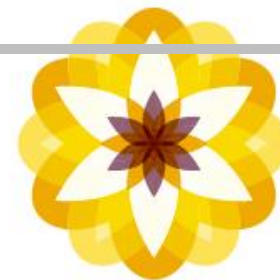
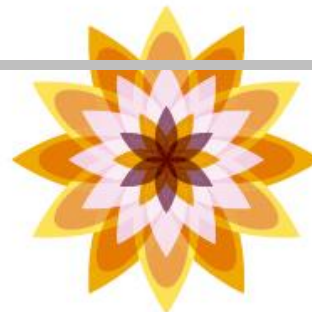


Chapter 04

숫자야구 게임



1. 이번에 만들 코드

- 숫자 야구 게임은 두 사람이 각각 3개의 숫자를 숨겨놓고 먼저 상대방의 숫자를 맞추는 편이 이기는 숫자 야구 놀이를 컴퓨터와 할 수 있도록 옮긴 것입니다.
- 물론 컴퓨터는 숫자를 맞히지 않고 숨겨두기 만하고, 플레이어(사람)가 제한된 횟수 내에 컴퓨터가 숨긴 숫자 3개를 모두 맞히면 적절한 칭찬을 해주도록 구현 하겠습니다.
- 규칙
 - 컴퓨터가 숨기는 숫자 3개는 1부터 9까지의 수로 0은 포함되지 않는다.
 - 3개의 숫자가 모두 다르다.
 - 플레이어는 3개의 숫자를 맞혀야 하는 것뿐만 아니라 그 위치까지 정확히 맞혀야 한다.
 - 예를 들어 컴퓨터가 1, 2, 3 의 3개의 숫자를 숨겨두었다면, 플레이어는 2, 3, 1 또는 3, 2, 1 이라고 해서는 안 되고 반드시 1, 2, 3 이라고 해야 한다.

1. 이번에 만들 코드

- 플레이어가 정답을 추측할 수 있도록 컴퓨터는 매 번 볼카운트를 알려줍니다.
- 이 볼카운트가 정답을 맞힐 수 있는 중요한 힌트가 됩니다.
- 볼카운트의 규칙은 플레이어가 입력한 숫자와 컴퓨터가 숨겨놓은 숫자가 같고 위치만 다르면 Ball 이고, 위치도 같으면 Strike으로 간주하는 것입니다.
- 예를 들어, 컴퓨터가 숨겨 놓은 숫자가 1, 2, 3 이고 플레이어가 입력한 숫자가 2, 1, 3 이면, 1과 2는 위치가 다르기 때문에 Ball 이고 3은 위치까지 같기 때문에 Strike 이어서, 볼카운트는 1 Strike, 2 Ball 이 됩니다.

컴퓨터가 숨긴 숫자	플레이어가 입력한 숫자	볼카운트
1, 2, 3	2, 5, 3	1Strike, 1Ball
4, 2, 1	4, 2, 7	2Strike, 0Ball
4, 5, 9	1, 3, 8	0Strike, 0Ball
2, 5, 8	5, 8, 2	0Strike, 3Ball
3, 6, 7	6, 7, 4	0Strike, 2Ball
3, 7, 2	3, 7, 2	3Strike, 0Ball→Game Over

1. 이번에 만들 코드

- 이 프로그램에서 컴퓨터가 숨겨두는 3개의 숫자와 플레이어가 입력하는 3개의 숫자는 배열에 저장되고, 규칙에 어긋나지 않는 숫자를 만들기 위해서 반복문을 사용합니다.
- 또 앞에서 배운 프로그램과는 달리 메서드를 정의하고 호출하기 때문에, 이 프로그램을 잘 이해하면 자바의 배열과 반복문, 메서드의 사용법을 익힐 수 있습니다.

2. 메서드 호출

- 프로그램을 개발하는 방법 중의 하나는 커다란 문제를 간단히 해결할 수 있는 작은 부분으로 쪼개어 각 부분을 구현함으로써 전체 문제를 해결하는 프로그램을 만드는 것입니다.
- 이러한 작은 부분을 모듈(module) 또는 서브루틴(subroutine) 이라고 부르는데, 모듈식으로 프로그램을 만들면, 개발하기에 용이할 뿐만 아니라 만든 프로그램을 이해하기 쉬워지고 반복적으로 나오는 부분을 하나의 모듈로 만들어서 계속 부를 수 있기 때문에 전체 프로그램의 크기가 줄어 드는 이점이 있습니다.
- C 언어나 C++ 언어에서는 이러한 모듈을 함수(function) 라고 부르고 파스칼 언어에서는 프로시저(procedure) 라고 하지만 자비에서는 메서드(method) 라고 부릅니다.

2. 메서드 호출

- 자바의 메서드는 반환하는 값의 데이터형(반환형), 메서드 이름, 인수인 매개 변수의 리스트를 정의하는 헤더 (header)와 처리할 일을 정의하는 바디 (body) 로 구성됩니다.

```
반환형 메서드이름([매개변수, 매개변수, ..., 매개변수])  → 메서드 헤더
{
    // 명령어들                                         → 메서드 바디
}
```

- 예를 들어 int형인 두 수 x와 y를 매개 변수로 받아서 두 수의 합을 int형으로 돌려주는 메서드 add는 다음처럼 정의할 수 있습니다.

```
반환형  메서드이름  매개변수  매개변수
  ↓         ↓         ↓         ↓
int      add      ( int x,   int y ) ← 메서드 헤더
{
    return ( x + y );                ← 메서드 바디
}
```

2. 메서드 호출

- 이 메서드 add를 호출하는 명령어는 다음과 같습니다.

```
z    =    add    ( 10  , 20 );
```

↑ ↑ ↑ ↑

반환값 메서드이름 인수 인수

- 위 명령어에서 10과 20은 변수가 아닌 상수지만, add의 매개변수 x와 y의 위치에 전달되어 각각 x와 y가 됩니다.
- 따라서 add가 돌려주는 값은 30 이 되고 z 에 저장되게 됩니다.
- 이때 Z는 add의 반환형인 int형이거나 int 형 값이 저장될 수 있는 데이터형이어야 합니다.
- 만일 z가 int형이 저장될 수 없는 데이터형인 경우는 형변환을 해야 합니다.

2. 메서드 호출

- 메서드를 호출하면 괄호 안에 주어진 매개 변수의 리스트가 해당 메서드에게 인수로 전달되고, 메서드 바디의 실행 결과는 메서드를 호출한 자리에 치환됩니다.
- 만일 인수가 필요 없는 메서드라면 매개 변수 리스트를 생략할 수 있습니다.
- 그러나 반환형은 반드시 명시해야 하는데, 만일 돌려줄 반환형이 없는 경우에도 생략해서는 안되고 void라고 표시해야 합니다.
- C 언어에서는 반환형이 int형일 때는 종종 반환형 선언을 생략하기도 하고, 바디에서 return을 빠뜨려도 됩니다만, 자바에서는 프로그래머의 실수를 방지하기 위해서 반드시 표시하도록 정해져 있습니다.
- 다음은 인수도 없고 반환값도 없는 메서드의 예입니다.

```
void printHello()  
{  
    System.out.println("Hello!");  
}
```


2. 메서드 호출

- 자바에서 인수로 매개 변수를 전달하는 방식은 크게 두 가지가 있는데, 기본 데이터형은 모두 Call by Value로 처리되고, 클래스의 객체는 Call by Reference로 처리됩니다.
- 두 방식의 차이점을 분명히 알아야 프로그램을 만들 수 있기 때문에 잘 알아두어야 합니다.
- Call by value
 - 자바에서 인수로 기본 데이터 형을 사용하면 모두 Call by Value가 됩니다.
 - Call by Value는 주어진 값을 복사하여 처리하는 방식입니다.
 - 즉 메서드 내에서 인수로 전달되는 데이터형과 동일한 종류의 데이터형 변수를 만들어 값을 복사한 후, 메서드 내의 변수만을 가지고 수행하는 방식입니다.
 - 따라서 메서드 내의 처리 결과는 메서드 밖의 변수에는 영향을 미치지 않습니다.
 - 다음은 Call by Value로 두 변수의 값을 바꾸려고 한 예제입니다.

2. 메서드 호출

■ Call by value

```
1 public class CallByValueTest {
2     public static void swap(int x, int y) {
3         int temp = x;
4         x = y;
5         y = temp;
6     }
7     public static void main(String[] args) {
8         int a = 10;
9         int b = 20;
10        System.out.println("swap() 메서드 호출 전: " + a + ", " + b);
11        swap(a, b);
12        System.out.println("swap() 메서드 호출 후: " + a + ", " + b);
13    }
14 }
```

2. 메서드 호출

■ Call by value

▪ 결과



```
<terminated> CallByValueTest [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe (2021. 6. 26. 오후 3:12:58 - 오후 3:13:01)
swap() 메서드 호출 전: 10, 20
swap() 메서드 호출 후: 10, 20
```

- `swap(a, b);`로 호출했으므로 `x` 값은 `a`, `y` 값은 `b`의 값이 복사되지만, `swap()` 메서드가 끝난 후에는 돌려받지 못합니다.
- 위의 예제에서 `main()` 메서드의 `a`와 `b`는 `swap` 메서드 내의 `x`와 `y`에 각각 값이 복사되고, `swap()` 메서드에서는 `x`와 `y`만 다루어지기 때문에, `swap()` 내에서 `x`와 `y`의 값을 서로 바꾸지만 `main()` 메서드의 `a`와 `b`에는 아무런 영향을 미치지 않게 되는 것입니다.
- 만일 위 예제에서 `a`와 `b`의 값이 바뀌도록 하고 싶다면, 이어서 배울 `Call by Reference`를 쓰거나 다음처럼 `a`와 `b`를 전역변수로 선언하여 사용하면 됩니다.

2. 메서드 호출

■ Call by value

```
1 public class CallByValueTest2 {  
2     static int a;  
3     static int b;  
4  
5     public static void swap( ) {  
6         int temp = a;  
7         a = b;  
8         b = temp;  
9     }  
10 }
```

2. 메서드 호출

■ Call by value

```
11 public static void main(String[] args) {  
12     a = 10;  
13     b = 20;  
14  
15     System.out.println("swap() 메서드 호출 전: " + a + ", " + b);  
16     swap( );  
17     System.out.println("swap() 메서드 호출 후: " + a + ", " + b);  
18 }  
19 }
```

2. 메서드 호출

■ Call by value

▪ 결과



```
<terminated> CallByValueTest2 [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe (2021. 6. 26. 오후 3:35:34 - 오후 3:35:35)
swap() 메서드 호출 전: 10, 20
swap() 메서드 호출 후: 20, 10
```

- swap();으로 호출했으므로 값의 전달은 일어나지 않고, swap() 메서드에서도 인수를 받지 않습니다.
- 하지만 swap() 메서드에서 전역 변수에 바로 값을 저장하기 때문에, 전역변수 a, b를 사용하는 모든 메서드에 영향을 미치게 됩니다.

■ Call by Reference

- Call by Value가 주어진 매개 변수의 값을 복사해서 처리하는데 비해 Call by Reference는 매개 변수의 원래 주소에 값을 저장하는 방식입니다.
- 따라서 Call by Reference로 인수를 전달하면, 메서드의 실행에 따라 인수로 전달한 변수의 값이 영향을 받게 됩니다.
- 자바에서는 클래스 객체를 인수로 전달한 경우에만 Call by Reference로 처리합니다.
- 다음은 Call by Reference로 두 변수의 값을 바꾼 예제입니다.

```
1 public class CallByReferenceTest {  
2     public static void swap(Number z) {  
3         int temp = z.x;  
4         z.x = z.y;  
5         z.y = temp;  
6     }  
7 }
```

2. 메서드 호출

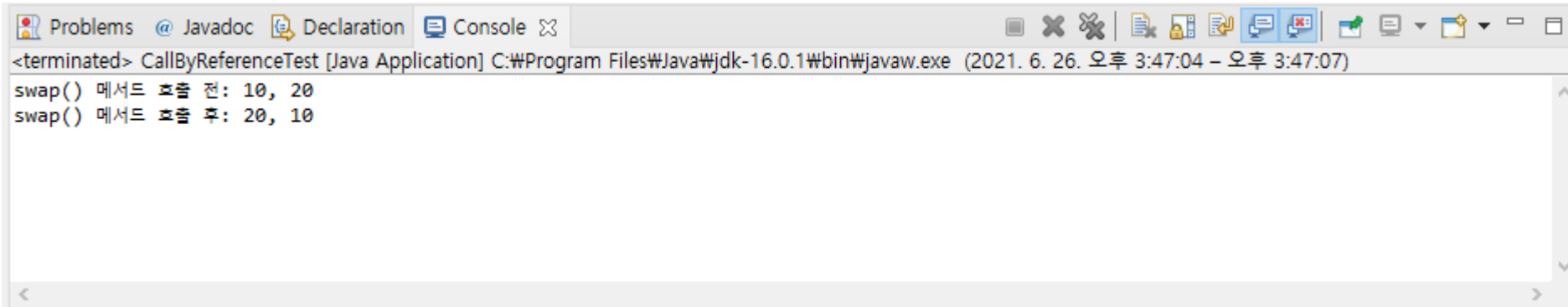
■ Call by Reference

```
8      public static void main(String[] args) {
9          Number n = new Number();    // Number 클래스로 n 생성
10         n.x = 10;
11         n.y = 20;
12
13         System.out.println("swap() 메서드 호출 전: " + n.x + ", " + n.y);
14         swap(n);
15         System.out.println("swap() 메서드 호출 후: " + n.x + ", " + n.y);
16     }
17 }
18 class Number{
19     public int x;
20     public int y;
21 }
```


2. 메서드 호출

■ Call by Reference

▪ 결과



```
<terminated> CallByReferenceTest [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe (2021. 6. 26. 오후 3:47:04 - 오후 3:47:07)
swap() 메서드 호출 전: 10, 20
swap() 메서드 호출 후: 20, 10
```

- Number 클래스의 객체를 생성하여 값을 전달하게 되면 객체가 저장한 값이 주소 값이기 때문에, swap() 메서드에서 객체에 저장한 결과가 main() 메서드로 돌려지게 됩니다.
- 위의 예제에서는 main() 메서드에서 Number 클래스의 객체인 n을 만들어 인수로 전달하기 때문에, swap() 메서드 내에서 Number 클래스 내의 x와 y 값을 바꾼 결과가 main() 메서드에 영향을 미치게 됩니다.
- 이렇게 다른 메서드에서 현재의 메서드 내의 변수 값을 바꾸는 현상을 사이드 이펙트(side effect)라고 합니다.
- 사이드 이펙트는 메서드 간의 값 전달을 쉽게 하기 때문에 편리하지만, 실수로 프로그래머가 모르는 사이에 값이 바뀌면 심각한 문제를 일으킬 수 있기 때문에 위험하다고 알려져 있습니다.
- 그래서 자바는 모든 기본 데이터형은 Call by Value로 값을 주고받아 사이드 이펙트가 일어나지 않도록 했고, Call by Reference가 필요한 경우는 명시적으로 클래스 객체를 주고받도록 정해둔 것입니다.

3. 메서드 오버로딩

- 프로그래밍을 할 때 각각의 메서드를 모두 다른 이름으로 만드는 것은 당연하지만, 때때로 같은 이름의 메서드를 여러 개 정의하고 싶을 때가 있을 수 있습니다.
- c 언어에서는 모든 함수가 다른 이름이어야 하지만, 자바에서는 인수의 개수나 종류가 다르다면 같은 이름의 메서드를 얼마든지 정의할 수 있습니다.
- 이렇게 같은 이름의 메서드를 여러 개 정의할 수 있도록 해주는 것을 메서드 오버로딩(method over loading) 이라고 합니다.
- 메서드 오버로딩이 필요한 경우가 어떤 때일지 생각해봅시다.
- 예를 들어 int형의 두 수를 인수로 받아 합을 돌려주는 다음과 같은 add() 메서드를 만들었습니다.

```
int add(int x, int y){  
    return x + y;  
}
```

3. 메서드 오버로딩

- 그런데 나중에 `int`형이 아니고 `double`형의 두 수를 인수로 받아 돌려주는 메서드가 필요하게 되었다고 한다면, 메서드 오버로딩이 불가능한 경우에는 이미 `add()` 메서드가 있기 때문에 다음처럼 `d_add()` 메서드를 만들어야 합니다.

```
double d_add(double x, double y){  
    return x + y;  
}
```

- 비슷한 일을 하는 메서드인데도 같은 이름을 쓸 수 없다면 매번 다른 이름으로 정의해야 하고, 프로그래머는 다음과 같은 여러 개의 이름을 기억해야 합니다.

<code>add(int x, int y)</code>	→ <code>int</code> 형인 두 수를 더하는 <code>add</code>
<code>d_add(double x, double y)</code>	→ <code>double</code> 형인 두 수를 더하는 <code>add</code>
<code>f_add(float x, float y)</code>	→ <code>float</code> 형인 두 수를 더하는 <code>add</code>
<code>l_add(long x, long y)</code>	→ <code>long</code> 형인 두 수를 더하는 <code>add</code>

3. 메서드 오버로딩

- 그러나 자바의 경우는 메서드 오버로딩을 지원하기 때문에 모두 같은 이름으로 정의하는 것이 가능합니다.
- 예를 들어, 위의 여러 `add()` 메서드들도 다음처럼 같은 이름으로 정의할 수 있습니다.

<code>add(int x, int y)</code>	→ int형인 두 수를 더하는 add
<code>add(double x, double y)</code>	→ double형인 두 수를 더하는 add
<code>add(float x, float y)</code>	→ float형인 두 수를 더하는 add
<code>add(long x, long y)</code>	→ long형인 두 수를 더하는 add

- 이렇게 하면, `add()` 메서드를 호출하는 쪽에서는 여러 메서드 이름을 외울 필요 없이 `add(10, 3)`, `add(2.5, 4.2)`, `add(3F, 1.2F)`, `add(100L, 2000L)`, .. 등으로 부를 수 있습니다.

3. 메서드 오버로딩

- 그런데, 메서드의 이름이 같은데 자바는 어떻게 적절한 메서드를 호출할 수 있는 걸까요?
- 자바는 메서드의 이름 뿐만 아니라 인수를 함께 보고 판단하는 것입니다.
- 같은 이름의 메서드가 2 개 이상 있다면, 주어진 인수가 몇 개인지, 종류가 무엇 인지로 판단하게 됩니다.
- 예를 들어 `add(4, 5)` 는 2 개의 `int`형 인수가 든 `add()` 메서드이기 때문에, 자바 는 `add(int x, int y)`를 호출해줍니다.
- 따라서 같은 이름의 메서드를 정의하는 것은 괜찮지만 인수까지 동일한 메서드를 정의해서는 안됩니다.

3. 메서드 오버로딩

- 다음의 두 `add()`는 함께 정의 될 수 없습니다.

```
int add(int x, int y){  
    return x + y;  
}
```

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

3. 메서드 오버로딩

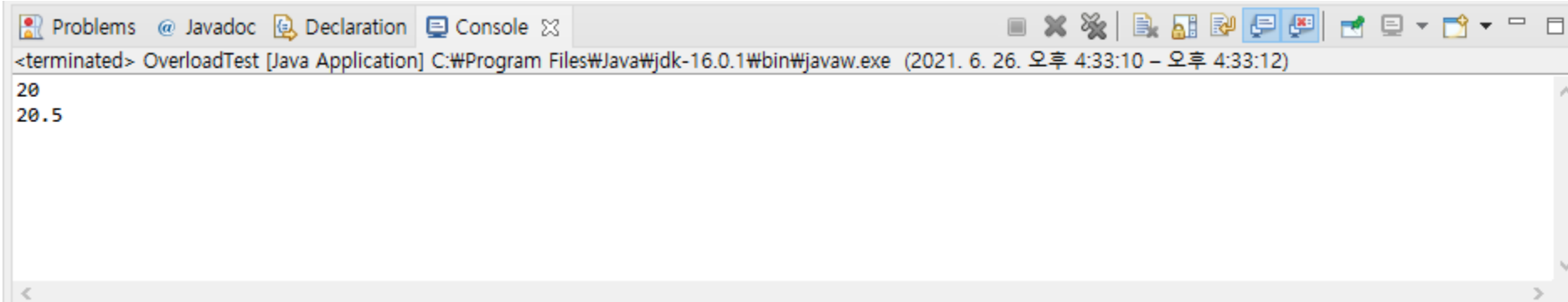
```
1 public class OverloadTest {  
2     public static int max(int x, int y) {  
3         if (x > y) {  
4             return x;  
5         }else {  
6             return y;  
7         }  
8     }  
9  
10    public static double max(double x, double y) {  
11        if (x > y) {  
12            return x;  
13        }else {  
14            return y;  
15        }  
16    }
```

3. 메서드 오버로딩

```
17     public static void main(String[] args) {
18         int a = 10;
19         int b = 20;
20
21         System.out.println(max(a, b)); // int형 인수 2개를 받는 max() 메서드
호출
22
23         double c = 10.5;
24         double d = 20.5;
25         System.out.println(max(c, d)); // double형 인수 2개를 받는 max() 메서드
호출
26     }
27 }
```


3. 메서드 오버로딩

■ 결과



```
<terminated> OverloadTest [Java Application] C:\Program Files\Java\jdk-16.0.1\bin\javaw.exe (2021. 6. 26. 오후 4:33:10 - 오후 4:33:12)
20
20.5
```

- 메서드의 이름이 같을 경우에는 전달하는 인수의 데이터형과 수로 메서드를 결정합니다.
- 만약 애매하게 인수를 주게 되면 메서드를 잘지 못하는 경우도 발생합니다.

4. 반복문

■ 반복문이란?

- 1부터 10까지 더하여 그 합을 계산해 볼까요?
- 지금까지 우리가 배운 것만으로 코드를 작성한다면 다음과 같을 것입니다.
- Ex) 1부터 10까지 더하기

```
1 package loopexample;
2
3 public class BasicLoop {
4     public static void main(String[] args) {
5         int num = 1;
6         num += 2;
7         num += 3;
8         num += 4;
9         num += 5;
10        num += 6;
11        num += 7;
```

4. 반복문

■ 반복문이란?

- Ex) 1부터 10까지 더하기

```
12         num += 8;
13         num += 9;
14         num += 10;
15
16         System.out.println("1부터 10까지의 합은 " + num + "입니다.");
17     }
18 }
```

- 그냥 보기에 별로 효율적이지 않은 것 같죠?
- 이렇게 반복되는 일을 처리하기 위해 사용하는 것이 '반복문'입니다.
- 자바 프로그램에서 사용하는 반복문의 종류에는 while문, do-while문, for문 이렇게 세 가지가 있습니다.
- 모두 반복 수행을 한다는 것은 동일하지만, 사용 방법에 조금씩 차이가 있습니다.

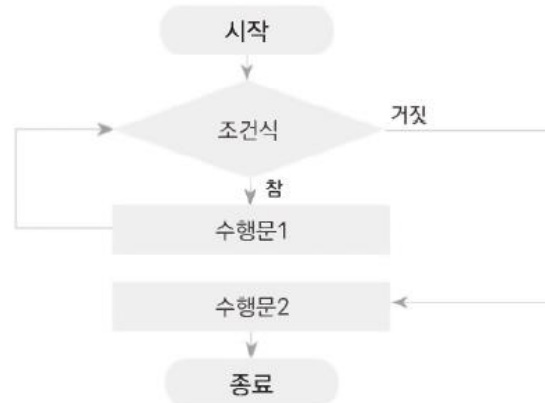
4. 반복문

■ while 문

- 반복문 중 먼저 while문을 살펴보겠습니다.
- while문은 조건식이 참인 동안 수행문을 반복해서 수행합니다.
- while문의 문법을 살펴보면 다음과 같습니다.

```
while(조건식) {  
    수행문1;  
    ...  
}  
수행문2;  
...
```

조건식이 참인 동안
반복 수행

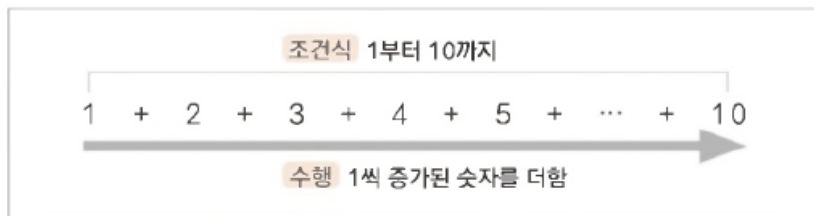


- 어떠한 조건식을 만족하는 동안 종괄호 } 안의 수행문을 반복해서 처리합니다.
- 조건문과 마찬가지로 수행문이 하나인 경우에는 { }를 사용하지 않을 수 있습니다.

4. 반복문

■ while 문

- 그러면 우리가 앞에서 만든 1부터 10까지 더하는 프로그램을 while문으로 만들어 보겠습니다.
- 반복문은 조건식을 만족하는 동안에 수행문을 반복해서 처리한다고 했습니다.
- 그러면 조건식을 어떻게 만들면 될까요?

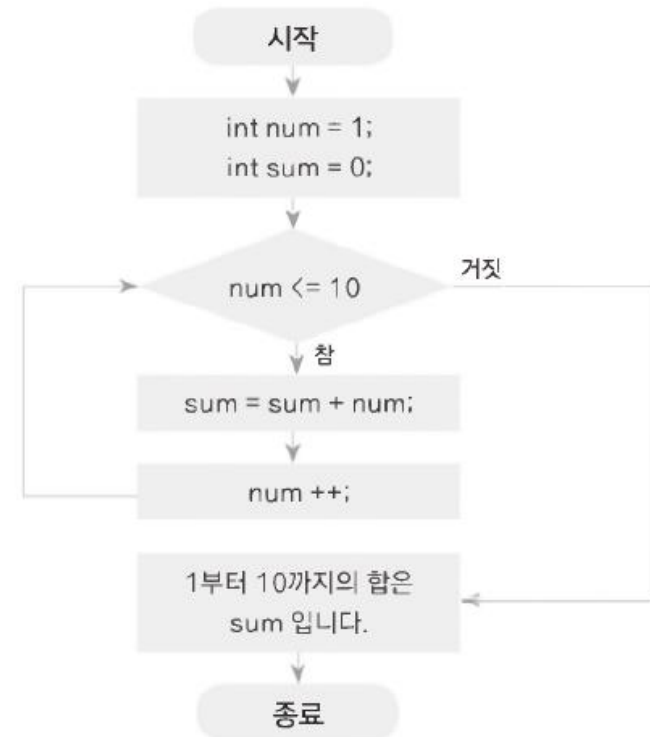


- '1부터 10까지 숫자가 커지는 동안'을 조건으로 하고, 1씩 증가한 숫자를 더하는 작업을 합니다.
- 1씩 올려 나갈 변수를 하나 선언하고, 증가한 숫자를 모두 더한 결과 값은 다른 변수에 저장하겠습니다.

4. 반복문

■ while 문

- 이 내용을 순서도로 보면 다음과 같습니다.
- num이 1 씩 증가하다가 숫자가 10을 넘어가는 순간 while문이 끝납니다.
- 즉 num이 10일 때까지 1씩 더한 값이 sum에 저장됩니다.



- 다음은 while문이 반복되는 과정을 보여 주는 표입니다.

num	num = 1	num = 2	num = 3	num = 4	num = 5
sum =	sum =	sum =	sum =	sum =	sum =
sum + num	0 + 1	1 + 2	3 + 3	6 + 4	10 + 5
sum	sum = 1	sum = 3	sum = 6	sum = 10	sum = 15

...

num = 9	num = 10	num = 11
sum =	sum =	
36 + 9	45 + 10	while문
sum = 45	sum = 55	종료

4. 반복문

■ while 문

- 전체 코드는 다음과 같습니다.
- Ex) while문 활용하여 1부터 10까지 더하기

```
1 package loopexample;
2
3 public class WhileExample {
4     public static void main(String[] args) {
5         int num = 1;
6         int sum = 0;
7
8         while(num <= 10) { // num값이 10보다 작거나 같을 동안
9             sum += num;    // 합계를 뜻하는 sum에 num을 더하고
10            num++;         // num에 1씩 더해 나감
11        }
```

4. 반복문

■ while 문

- 전체 코드는 다음과 같습니다.
- Ex) while문 활용하여 1부터 10까지 더하기

```
12         System.out.println("1부터 10까지의 합은 " + sum + "입니다.");  
13     }  
14 }
```

- 위의 예제에서 5~6행을 보면 num 변수와 sum 변수를 선언하면서 동시에 초기값을 저장했습니다.
- 변수를 항상 초기화해야 하는 것은 아니지만, 이 예제에서는 반드시 초기화를 해야 합니다.
- 만약 변수를 초기화하지 않고 프로그램을 실행하면 오류가 납니다.
- 왜 그럴까요?
- while문 내부를 보면 sum에 num 값을 더해 줍니다.
- 그런데 num 값이 먼저 정해져 있지 않다면 sum에 무엇을 더해야 할지 알 수 없습니다.
- 또 sum 값도 정해져 있지 않다면 어떤 값에 num 값을 더해야 할지 알 수 없겠죠.
- 즉 변수를 사용하여 연산을 하거나 그 값을 가져다 사용하려면 변수는 반드시 어떤 값을 가지고 있어야 합니다.
- 따라서 이 예제에서는 num과 sum을 먼저 초기화해야 합니다.

4. 반복문

■ while 문

■ while문이 무한히 반복되는 경우

- 앞에서 살펴본 while문은 특정 조건을 만족하는 동안 반복되는 명령을 수행하고, 그렇지 않으면 수행을 중단한 후 while문을 빠져나옵니다.
- 그런데 어떤 일을 수행할 때는 멈추면 안 되고 무한 반복해야 하는 경우도 있습니다.
- 가장 쉬운 예로 여러분이 자주 사용하는 인터넷 쇼핑몰을 생각해 봅시다.
- 인터넷 쇼핑몰이 24시간 서비스하기 위해서는 쇼핑몰의 데이터를 저장하고 있는 웹 서버가 멈추지 않고 끊임없이 돌아가야 합니다.
- 웹 서버가 멈추면 고객들의 항의가 많을 겁니다.
- while문의 구조를 보면 조건식이 참이면 반복합니다.
- 따라서 while문을 다음과 같이 사용하면 조건이 항상 '참'이 되어 '무한 반복'하겠죠?

```
while(true){  
    ...  
}
```

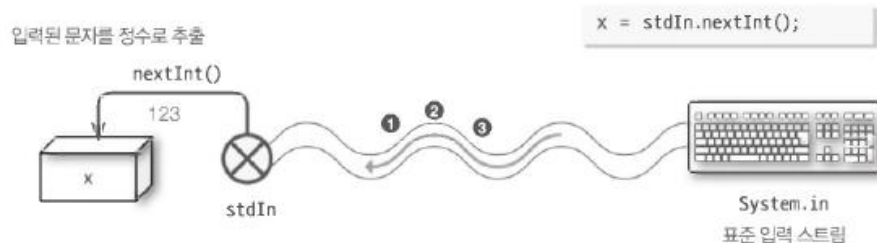
- 이렇게 끊임없이 돌아가는 시스템을 데몬(daemon)이라고 부릅니다.
- 데몬 시스템은 반복문을 이용하여 멈추지 않는 서비스를 구현할 수 있습니다.

4. 반복문

■ while 문

■ 연습문제

- 입력받은 정숫값부터 0까지 카운트다운하는 프로그램을 작성하라. 카운트다운 종료 후의 변수값을 확인할 수 있게 할 것.
- 키보드로 입력받는 것은 다음과 같이 수행한다.



```
import java.util.Scanner;          ---①
...
Scanner stdIn = new Scanner(System.in); ---②
...
int x = stdIn.nextInt( );          ---③
```

- 키보드로 값을 입력하려면 ①, ②, ③ 순서를 따른다.
- ②에서, System.in은 키보드와 연결된 표준 입력 스트림(standard input stream; STDIN)이다.
- 키보드와 연결된 표준 입력 스트림인 System.in에서 문자나 숫자를 꺼내는 '추출 장치'가 stdin이다.
- stdin은 다른 이름으로 변경할 수 있다

4. 반복문

■ while 문

■ 연습문제 4-1

- 입력받은 정숫값부터 0까지 카운트다운하는 프로그램을 작성하라. 카운트다운 종료 후의 변숫값을 확인할 수 있게 할 것.

```
1 package loopexample;
2
3 import java.util.Scanner;
4
5 public class Countdown {
6     public static void main(String[] args) {
7         Scanner stdIn = new Scanner(System.in);
8         System.out.println("카운트다운 합니다.");
9         int x = 0;
10    }
```

4. 반복문

■ while 문

■ 연습문제 4-1

- 입력받은 정숫값부터 0까지 카운트다운하는 프로그램을 작성하라. 카운트다운 종료 후의 변숫값을 확인할 수 있게 할 것.

```
11         while(x <= 0){
12             System.out.print("양의 정숫값: ");
13             x = stdIn.nextInt();
14         };
15
16         while(x >= 0)
17             System.out.println(x--);
18         System.out.println("x의 값이 " + x + "이 되었습니다.");
19     }
20 }
```

4. 반복문

■ while 문

■ 연습문제 4-2

- 입력한 값의 개수만큼 '*'를 표시하는 프로그램을 작성하자. 마지막에는 줄 바꿈 문자를 출력할 것. 단, 읽은 값이 1 미만이면 줄 바꿈 문자를 표시해서는 안 된다.

```
1 package loopexample;
2
3 import java.util.Scanner;
4
5 public class PutAsterisk {
6     public static void main(String[] args) {
7         Scanner stdIn = new Scanner(System.in);
8         System.out.print( " 몇 개의 *를 표시할까요?");
9         int n = stdIn.nextInt();
10
```

■ while 문

■ 연습문제 4-2

- 입력한 값의 개수만큼 '*'를 표시하는 프로그램을 작성하자. 마지막에는 줄 바꿈 문자를 출력할 것. 단, 읽은 값이 1 미만이면 줄 바꿈 문자를 표시해서는 안 된다.

```
11         if(n > 0) {  
12             int i = 0;  
13             while(i < n) {  
14                 System.out.print("*");  
15                 i++;  
16             }  
17             System.out.println();  
18         }  
19     }  
20 }
```

4. 반복문

■ do-while 문

- while문은 조건을 먼저 검사하기 때문에 조건식에 맞지 않으면 반복 수행이 한 번도 일어나지 않습니다.
- 하지만 do-while문은 { } 안의 문장을 무조건 한 번 수행한 후에 조건식을 검사합니다.
- 즉 조건이 만족하는지 여부를 마지막에 검사하는 것입니다.
- 따라서 중괄호 안의 문장을 반드시 한 번 이상 수행해야할 때 while문 대신 do-while문을 사용합니다.
- do-while문의 구조는 다음과 같습니다.



4. 반복문

■ do-while 문

- while문으로 만든 1부터 10까지 더하는 프로그램을 do-while문으로 바꿔 봅시다.
- Ex) do-while문 예제

```
1 package loopexample;
2
3 public class DoWhileExample {
4     public static void main(String[] args) {
5         int num = 1;
6         int sum = 0;
7
8         do{
9             sum += num;
10            num++;
11        } while(num <= 10);
```


4. 반복문

■ do-while 문

- while문으로 만든 1부터 10까지 더하는 프로그램을 do-while문으로 바꿔 봅시다.
- Ex) do-while문 예제

```
12         System.out.println("1부터 10까지의 합은 " + sum + "입니다.");  
13     }  
14 }
```

■ do-while 문

■ 연습문제 4-3

- 3자리의 양의 정숫값(100~999)을 읽는 프로그램을 작성하라

```
1 package loopexample;
2
3 import java.util.Scanner;
4
5 public class ThreeDigit {
6     public static void main(String[] args) {
7         Scanner stdIn = new Scanner(System.in);
8         int x;
9
10        do{
11            System.out.print("세 자리의 정숫값: ");
12            x = stdIn.nextInt();
13        } while(x < 100 || x > 999);
```

4. 반복문

■ do-while 문

■ 연습문제 4-3

- 3자리의 양의 정숫값(100~999)을 읽는 프로그램을 작성하라

```
14  
15         System.out.print("입력한 값은 "+ x + "입니다");  
16     }  
17 }
```

■ for 문

- 반복문 중에서 가장 많이 사용하는 반복문이 for문입니다.
 - for문은 while문이나 do-while 문보다 구조가 조금 더 복잡합니다.
 - 왜냐하면 반복문을 구현하는 데 필요한 여러 요소(변수의 초기화식, 조건식, 증감식)를 함께 작성하기 때문이지요. 처음에는 for문이 좀 낯설겠지만, 익숙해지면 어떤 조건부터 어떤 조건까지 반복 수행하는지 한눈에 알아볼 수 있어 편리합니다.
-
- for문의 기본 구조
 - for문의 구조를 살펴보면서 반복문의 요소도 함께 알아보시다.
 - 초기화식은 for문이 시작할 때 딱 한 번만 수행하며 사용할 변수를 초기화합니다.
 - 조건식에서 언제까지 반복 수행할 것인지 구현합니다.
 - 증감식에서 반복 횟수나 for문에서 사용하는 변수 값을 1 만큼 늘리거나 줄입니다.

```
for(초기화식; 조건식; 증감식){  
    명령어;  
}
```

4. 반복문

■ for 문

■ for문의 기본 구조

- for문의 수행순서를 이해하기 쉽도록 간단한 예를 들어 보겠습니다.
- 1부터 5까지 출력하는 프로그램을 for문으로 만들어 볼까요?
- 화살표와 번호는 이 예제가 수행되는 순서입니다.
- 조건식이 참인 동안 순서로 반복문을 계속 수행합니다.
- for 문은 증감식에서 사용한 변수가 조건식의 참·거짓 여부를 결정합니다.

```
int num;
for(num = 1; num <= 5; num++)
{
    System.out.println(num);
}
```

```
graph LR
    1((1)) --> 2((2))
    2 --> 3((3))
    3 --> 4((4))
    4 --> 2
```

4. 반복문

■ for 문

▪ for문의 기본 구조

① 처음 for문이 시작할 때 출력할 숫자인 num을 1로 초기화합니다.



- ② 조건식 `num <= 5`를 검사했을 때 num은 1이므로 참입니다.
③ 조건식이 참이기 때문에 for문의 `System.out.println(num);`을 수행하고 1을 출력합니다.
④ 증감식 `num++`를 수행하여 num 값은 2가 됩니다.



- ② 조건식 `num <= 5`를 검사했을 때 num은 2이므로 참입니다.
③ 조건식이 참이기 때문에 for문의 `System.out.println(num);`을 수행하고 2를 출력합니다.
④ 증감식 `num++`를 수행하여 num 값은 3이 됩니다.



...



② 조건식 `num <= 5`를 검사했을 때 num은 6이므로 거짓입니다. for문이 끝납니다.

4. 반복문

■ for 문

▪ for문의 기본 구조

- 1부터 10까지 더하는 과정을 for문으로 구현한 전체 프로그램은 다음과 같습니다.
- Ex) for문 예제

```
1 package loopexample;
2
3 public class ForExample1 {
4     public static void main(String[] args) {
5         int i;
6         int sum;
7         for(i = 1, sum = 0; i <= 10; i++) {
8             sum += i;
9         }
10    }
```

4. 반복문

■ for 문

▪ for문의 기본 구조

- 1부터 10까지 더하는 과정을 for문으로 구현한 전체 프로그램은 다음과 같습니다.
- Ex) for문 예제

```
11         System.out.println("1부터 10까지의 합은 " + sum + "입니다.");
12     }
13 }
```

- 초기화 부분과 증감식 부분도 콤마(,)로 구분하여 여러 문장을 사용할 수 있습니다.
- 예를 들어 7행을 보면 $i = 1$, $sum = 0$ 으로 두 개의 변수를 초기화한 것을 볼 수 있습니다.
- 연습문제 : for 문 연습하기
 - for문과 변수를 사용하여 '안녕하세요1, 안녕하세요2..., 안녕하세요10'까지 차례로 출력하는 프로그램을 작성해 보세요.

■ for 문

■ For문을 자주 사용하는 이유

- for문을 가장 많이 사용하는 이유는 반복 횟수를 관리할 수 있기 때문입니다.
- 물론 while문에서도 반복 횟수에 따라 구현할 수 있습니다.
- 1부터 10까지 더하는 프로그램을 while문과 for문으로 만들어 비교해 보겠습니다.

```
int num = 1;           //초기화
int sum = 0;
while(num <= 10) {     //조건 비교
    sum += num;
    num++;             //증감식
}
```

while문으로 구현



```
int sum = 0;
for(int num = 1; num <= 10; num++) {
    sum += num;
}
```

for문으로 구현

- while문으로 작성한 코드를 살펴보면 변수 num의 초기화와 조건 비교, 증감식을 따로 구현했습니다.
- 하지만 for문을 사용하여 구현하면 초기화, 조건 비교, 증감식을 한 줄에 쓸 수 있을 뿐더러 가독성도 좋습니다.

■ for 문

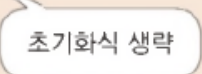
■ For문을 자주 사용하는 이유

- 또 for문은 배열과 함께 자주 사용됩니다.
- 배열은 같은 자료형이 순서대로 모여 있는 구조인데, 배열 순서를 나타내는 색인은 항상 0부터 시작합니다.
- 따라서 배열의 전체 요소 개수가 n 개일 때, 요소 위치는 $n-1$ 번째로 표현할 수 있습니다.
- 이러한 배열의 특성과 증감에 따른 반복을 표현하는 데 적합한 for문의 특성 때문에 for문과 배열을 함께 자주 사용하는 것입니다.

■ for문 요소 생략하기

- for문을 구성하는 요소는 코드가 중복되거나 논리 흐름상 사용할 필요가 없을 때 생략할 수 있습니다.
- 초기화식 생략
 - 이미 이전에 다른 곳에서 변수가 초기화되어 중복으로 초기화할 필요가 없을 때 초기화 부분을 생략할 수 있습니다.

```
int i = 0;
for( ; i < 5; i++) {
    ...
}
```



■ for 문

■ for문 요소 생략하기

• 조건식 생략

- 어떤 연산 결과 값이 나왔을 때 바로 for문의 수행을 멈추려면 조건식을 생략하고 for문 안에 if문을 사용하면 됩니다.
- 예를 들어 1부터 시작해 수를 더해 나갈 때 더한 결과 값이 200을 넘는지 검사하려면 for문 안에 if문을 사용합니다.

조건식 생략

```
for(i = 0; ; i++) {  
    sum += i;  
    if(sum > 200) break;  
}
```

• 증감식 생략

- 증감식의 연산이 복잡하거나 다른 변수의 연산 결과 값에 좌우된다면 증감식을 생략하고 for문 안에 쓸 수 있습니다.

증감식 생략

```
for(i = 0; i < 5; ) {  
    ...  
    i = (++i) % 10;  
}
```

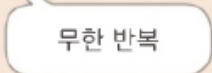
4. 반복문

■ for 문

■ for문 요소 생략하기

- 요소 모두 생략
 - 모든 요소를 생략하고 무한 반복하는 경우에 사용합니다.

```
for( ; ; ) {  
    ...  
}
```



A callout bubble with the text "무한 반복" (Infinite repetition) points to the empty for loop structure.

■ 중첩된 반복문

- 반복문 안에 또 다른 반복문을 중첩해서 사용하는 경우가 종종 있습니다.
- 간단한 예로 구구단을 출력해 보겠습니다.

4. 반복문

■ for 문

- 중첩된 반복문

- Ex) 중첩된 반복문

```
1 package loopexample;
2
3 public class NestedLoop {
4     public static void main(String[] args) {
5         int dan;
6         int times;
7
8         for(dan = 2; dan <= 9; dan++) {
9             for(times = 1; times <= 9; times++) {
10                 System.out.println(dan + " X " + times + " = " + dan * times);
11             }
```

4. 반복문

■ for 문

▪ 중첩된 반복문

- Ex) 중첩된 반복문

```
1      System.out.println();
2      }
3      }
11     }
```

- 반복문을 중첩해서 사용할 때 외부 for문과 내부 for문이 어떤 순서로 실행되는지 잘 이해해야 합니다.
- 구구단은 2단부터 9단까지 단이 증가합니다.
- 그리고 각 단은 1부터 9까지 곱하는 수가 증가하죠.

4. 반복문

■ for 문

■ 중첩된 반복문

- 그러면 '단이 증가'하는 부분과 '곱하는 수가 증가'하는 부분 중 무엇을 먼저 반복 수행 해야 할까요?



- 먼저 외부 for문의 초기화 값이 `dan = 2`이므로 구구단 2단부터 시작합니다.
- 그리고 내부 for문으로 들어가면 초기화 값 `times = 1`부터 시작해 1씩 증가하면서 9보다 작거나 같을 때까지 곱합니다.
- `times` 값이 10이 되면 내부 for문은 끝나고 외부 for문으로 돌아갑니다.
- 외부 for문에서 `dan++`를 수행하고 증가한 단의 값이 9보다 작은 지 확인합니다.
- 9보다 작으므로 다시 내부 for문으로 들어와 1부터 9까지 곱합니다.
- 정리하자면, 중첩 반복문을 쓸 때는 어떤 반복문을 먼저 수행해야 하는지 그리고 내부 반복문을 수행하기 전에 초기화해야 할 값을 잘 초기화했는지를 살펴야 합니다.
- for문 외의 다른 반복문도 중첩해서 사용할 수 있습니다.
- 연습문제 : 조금 전에 실습한 중첩 반복문 예제를 수정해 구구단을 3단부터 7단까지만 출력해 보세요.

■ for 문

■ 중첩된 반복문

- 우리는 지금까지 세 가지 반복문(while문, do-while문, for문)을 살펴보았습니다.
- 그러면 각 반복문을 언제, 어떤 경우에 사용하는 것이 가장 좋을까요?
- 반복 횟수가 정해진 경우에는 for문을 사용하는 것이 좋습니다.
- 그리고 수행문을 반드시 한 번 이상 수행해야 하는 경우에는 do-while문이 적합합니다.
- 이 두 경우 외에 조건의 참·거짓에 따라 반복문이 수행하는 경우에는 while문을 사용합니다.
- 물론 반복 횟수가 정해진 반복문을 while문으로 구현할 수도 있습니다.
- 그리고 조건의 참·거짓에 따른 반복문을 for문으로 구현할 수도 있죠.
- 하지만 좋은 프로그래밍 습관을 가지고 싶다면, 상황에 맞는 적절한 문법을 사용하는 것이 중요합니다.

■ for 문

▪ continue 문

- continue문은 반복문과 함께 쓰입니다.
- 반복문 안에서 continue문을 만나면 이후의 문장은 수행하지 않고 for문의 처음으로 돌아가 증감식을 수행합니다.
- 다음 예제를 봅시다.
- 1부터 100까지 수를 더할 때 홀수일 때만 더하고 짝수일 때는 더하지 않는 프로그램을 continue문으로 작성해 보겠습니다.
- Ex) continue문 예제

```
1 package loopexample;
2
3 public class ContinueExample {
4     public static void main(String[] args) {
5         int total = 0;
6         int num;
7     }
```

4. 반복문

■ for 문

▪ continue 문

- Ex) continue문 예제

```
8      for(num = 1; num <= 100; num++) {  
9          if(num % 2 == 0)  
10             continue;  
11             total += num;  
12         }  
13         System.out.println("1부터 100까지의 홀수의 합은: " + total + "입니다.");  
14     }  
15 }
```

- 그러면 continue문은 언제 사용할까요?
- 예제를 보면 반복문 안의 조건문에서 변수 num이 짝수일 때는 이후 수행을 생략하고 for문의 증감식으로 돌아가서 num에 1을 더합니다.
- num이 홀수일 때는 계속 진행(continue)해서 total += num ; 문장을 수행합니다.
- 이렇듯 continue문은 반복문을 계속 수행하는데, 특정 조건에서는 수행하지 않고 건너뛰어야 할 때 사용합니다.

4. 반복문

■ for 문

▪ break 문

- switch-case문에서 break문을 사용할 때 조건을 만족하면 다른 조건을 더 이상 비교하지 않고 switch문을 빠져 나왔지요?
- 반복문에서도 마찬가지입니다.
- 반복문에서 break문을 사용하면 그 지점에서 더 이상 수행문을 반복하지 않고 반복문을 빠져 나옵니다.
- 다음 예제를 살펴보겠습니다.
- 0부터 시작해 숫자를 1 씩 늘리면서 합을 계산할 때 숫자를 몇까지 더하면 100이 넘는지 알고 싶습니다.
- 지금까지 배운 반복문을 사용해 봅시다.
- Ex) break문 예제

```
1 package loopexample;
2
3 public class BreakExample1 {
4     public static void main(String[] args) {
5         int sum = 0;
6         int num = 0;
```

4. 반복문

■ for 문

▪ break 문

- Ex) break문 예제

```
7      for(num = 0; sum < 100; num++) {  
8          sum += num;  
9      }  
10     System.out.println("num: " + num);  
11     System.out.println("sum: " + sum);  
12 }  
13 }
```

- 이 코드를 실행해 보면 합은 105가 되었고 이 때 num 값은 15가 출력되었습니다.
- 그렇다면 1부터 15까지 더 했을 때 100이 넘는 걸까요?
- 그렇지 않습니다.
- 합이 105가 되는 순간 num 값은 14였습니다.
- 즉 1부터 14까지 더해져서 105가 되었고 num 값이 1씩 증가하여 15가 되었을 때 조건을 비교해 보니 합이 100보다 커서 반복문이 끝난 것입니다.

■ for 문

▪ break 문

- 따라서 우리가 원하는 정확한 값인 14를 얻으려면 증감이 이루어지기 전에 반복문을 끝내야 하죠.
- 그러면 반복문 안에 break문을 사용하여 수행을 중단해 보겠습니다.
- Ex) break문 예제

```
1 package loopexample;
2
3 public class BreakExample2 {
4     public static void main(String[] args) {
5         int sum = 0;
6         int num = 0;
7
8         for(num = 0; ; num++) {
9             sum += num;
```

4. 반복문

■ for 문

▪ break 문

- 따라서 우리가 원하는 정확한 값인 14를 얻으려면 증감이 이루어지기 전에 반복문을 끝내야 하죠.
- 그러면 반복문 안에 break문을 사용하여 수행을 중단해 보겠습니다.
- Ex) break문 예제

```
10         if(sum >= 100)           // sum이 100보다 크거나 같을 때(종료 조건)
11             break;                // 반복문 중단
12     }
13     System.out.println("num : " + num);
14     System.out.println("sum : " + sum);
15 }
16 }
```

- 위 예제는 0부터 시작해 1씩 늘린 숫자를 sum에 더합니다.
- 그리고 sum 값이 100보다 크거나 같으면 반복문을 바로 빠져 나옵니다.
- 프로그램을 실행하면 num 값이 14일 때 합이 105가 되는 것을 알 수 있습니다.

■ for 문

▪ break 문

- Ex) break문 예제
- 종료 조건을 for문 안에 사용하면 num 값을 늘리는 증감식을 먼저 수행하므로 num 값이 15가 됩니다.
- 따라서 프로그램 실행 중에 반복문을 중단하려면 break문을 사용해야 정확한 결과 값을 얻을 수 있습니다.

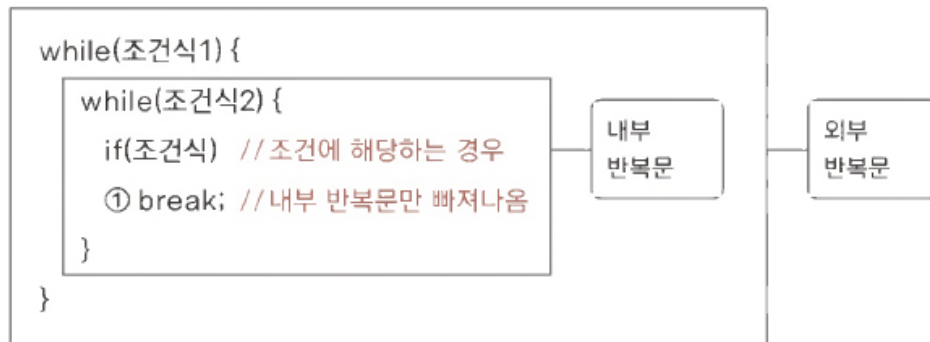
4. 반복문

■ for 문

▪ break 문

- break문의 위치

- 앞의 예제에서 봤듯이 반복문이 중첩된 경우가 있습니다.
- 이 경우에 break문을 사용하면 모든 반복문을 빠져 나오는 것이 아니고 break문을 감싸고 있는 반복문만 빠져나옵니다.



- 위 코드의 ① 위치에서 break문을 사용하면 if 조건문만 빠져나온다고 생각할 수도 있는데, 반복문 안의 break문은 해당 반복문 수행만 중지한다는 것을 기억하기 바랍니다.
- 즉 이러한 경우에는 내부 반복문만 빠져나오고 외부 반복문은 계속 수행합니다.
- 정리하자면, continue문은 반복문을 계속 수행하지만 특정 조건에서 수행문을 생략하는 경우에 사용하고, break문은 반복문을 더 이상 수행하지 않고 빠져 나올 때 사용합니다.

4. 반복문

■ for 문

■ 연습문제 4-4

- 기호 문자를 임의의 개수만큼 출력했던 [문제 4-2]를 for문으로 작성하자.

```
1 package loopexample;
2
3 import java.util.Scanner;
4
5 public class PutAsteriskFor {
6     public static void main(String[] args) {
7         Scanner stdIn = new Scanner(System.in);
8         System.out.print("몇 개의 *를 표시할까요? ");
9         int n = stdIn.nextInt();
10    }
```

4. 반복문

■ for 문

■ 연습문제 4-4

- 기호 문자를 임의의 개수만큼 출력했던 [문제 4-2]를 for문으로 작성하자.

```
11         if (n > 0) {  
12             for(int i = 0; i < n; i++)  
13                 System.out.print("*");  
14         }  
15         System.out.println();  
16     }  
17 }
```

■ 자료를 순차적으로 관리하는 구조, 배열

- 학교에 학생이 100명 있습니다.
- 이 학생들 100명의 학번을 어떻게 관리할 수 있을까요?
- 학번의 자료형을 정수라고 하면 학생이 100명일 때 `int studentID1, int studentID2, int studentID3, ..., int studentID100` 이렇게 변수 100개를 선언해서 사용해야겠죠.
- 그런데 학번에 대한 여러 개 변수들을 일일이 쓰는 것은 너무 귀찮고 번거롭습니다.
- 이때 사용하는 자료형이 배열(array)입니다.
- 배열은 자료 구조의 가장 기초 내용입니다.



- 배열을 사용하면 자료형이 같은 자료 여러 개를 한 번에 관리할 수 있습니다.
- 위 그림으로 알 수 있듯이 배열은 자료가 연속으로 나열된 자료 구조입니다.

■ 배열 선언과 초기화

- 배열을 사용하려면 먼저 배열을 선언해야 합니다.
- 배열도 변수와 마찬가지로 자료형을 함께 선언합니다.
- 배열은 선언하는 문법은 오른쪽과 같습니다.
- 배열을 이루는 각각의 자료를 배열 요소라고 합니다.
- 배열 요소는 자료형이 모두 같습니다.
- 먼저 저장하려는 자료의 성격에 맞게 자료형을 정하고 선언하려는 배열 요소 개수만큼 [] 안에 적습니다.
- new 예약어는 배열을 새로 만들라는 의미입니다.

```
자료형[ ] 배열이름 = new 자료형[개수];  
자료형 배열이름[ ] = new 자료형[개수];
```

8. 배열

■ 배열 선언과 초기화

- 앞에서 이야기한 학생들의 학번을 배열로 선언해 봅시다.

```
int[ ] studentIDs = new int[10];           // int형 요소가 10개인 배열 선언
```

- 위 문장은 int형 요소가 10개인 배열을 선언한 것입니다.
- 이렇게 선언했을 때 메모리 상태를 그림으로 나타내면 다음과 같습니다.



- 배열을 선언하면 선언한 자료형과 배열 길이에 따라 메모리가 할당됩니다.
- 위 그림을 보면 자료형이 int형이므로 배열 요소를 저장할 수 있는 공간의 크기는 전부 4바이트로 동일합니다.
- 배열 요소를 저장할 수 있는 공간이 총 10개이므로 이 배열을 위해 총 40바이트의 메모리가 할당되는 것입니다.

■ 배열 선언과 초기화

■ 배열 초기화하기

- 자바에서 배열을 선언하면 그와 동시에 각 요소의 값이 초기화됩니다.
- 배열의 자료형에 따라 정수는 0, 실수는 0.0, 객체 배열은 null로 초기화되며, 다음처럼 배열 선언과 동시에 특정 값으로 초기화할 수도 있습니다.
- 배열이 초기화 요소의 개수만큼 생성되므로 [] 안의 개수는 생략합니다.

```
int[ ] studentIDs = new int[ ] {101, 102, 103};           // 개수는 생략함
```

- 다음과 같이 값을 넣어 초기화할 때 안에 개수를 쓰면 오류가 발생합니다.

```
int[ ] studentIDs = new int[3] {101, 102, 103};          // 오류 발생
```

- 선언과 동시에 초기화할 때 다음과 같이 new int[] 부분을 생략할 수도 있습니다.
- int형 요소가 3개인 배열을 생성한다는 의미이므로 new int[]를 생략해도 됩니다.

```
int[ ] studentIDs = {101, 102, 103};                     // int형 요소가 3개인 배열 생성
```

- 하지만 다음과 같이 배열의 자료형을 먼저 선언하고 초기화하는 경우에는 new int[]를 생략할 수 없습니다.

```
int[ ] studentIDs;                                         // 배열 자료형 선언  
studentIDs new int[ ] {101, 102, 103};                   // new int[ ]를 생략할 수 없음
```

■ 배열 사용하기

- 선언한 배열의 각 요소에 값을 넣을 때나 배열 요소에 있는 값을 가져올 때는 []를 사용합니다.
- 만약 배열의 첫 번째 요소에 값 10을 저장한다면 다음처럼 코드를 작성합니다.

```
studentIds[0] = 10;           // 배열의 첫 번째 요소에 값 10을 저장
```

- 첫 번째 요소에 값을 저장했는데 [] 안에는 0이 있네요.
- 위 코드에 대해 자세히 살펴봅시다.

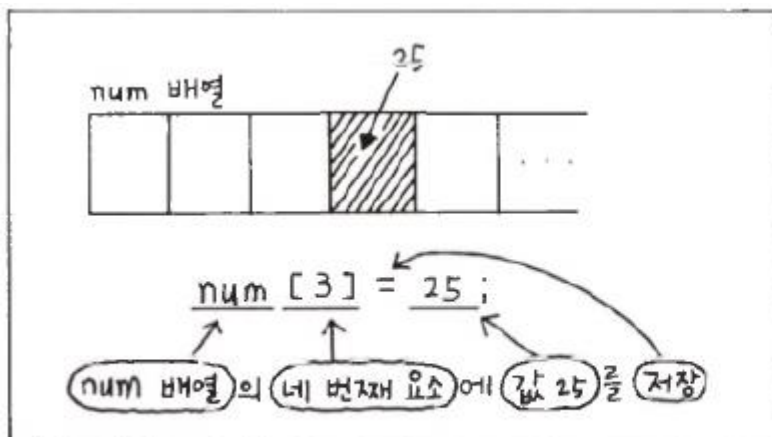
▪ 인덱스 연산자 []

- []는 배열을 처음 선언할 때 사용한 연산자입니다.
- 배열 이름에 []를 사용하는 것을 인덱스 연산이라고 합니다.
- 인덱스 연산자의 기능은 배열 요소가 저장된 메모리 위치를 찾아 주는 역할입니다.
- 변수 이름으로 변수가 저장된 메모리 위치를 찾는 것처럼, 배열에서 [i] 인덱스 연산을 하면 i번째 요소의 위치를 찾아 해당 위치의 메모리에 값을 넣거나 이미 저장되어 있는 값을 가져와서 사용할 수 있습니다.

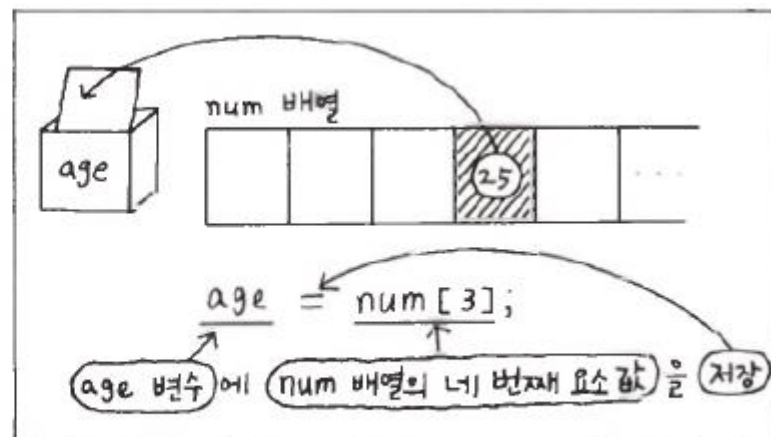
■ 배열 사용하기

■ 인덱스 연산자 []

- 예를 들어 int형으로 선언한 num 배열의 네 번째 요소에 값 25를 저장하고, 그 값을 가져와 int형 int형 변수 age에 저장한다면 다음 그림과 같습니다.



배열의 요소에 값 저장하기



배열 요소의 값 가져오기

■ 배열 사용하기

■ 배열 순서는 0번부터

- 배열 길이(처음에 선언한 배열 전체 요소 개수)가 n 이라고 하면, 배열 순서는 0번부터 $n-1$ 번까지 입니다.
- 0번 요소를 배열의 첫 번째 요소라고 합니다.
- 이해를 돕기 위해 정수 10개를 저장할 배열을 선언하고 각 요소를 값 1부터 10까지 초기화한 후 for 반복문을 사용하여 배열 요소 값을 하나씩 출력해 보겠습니다.
- Ex) 배열을 초기화하고 출력하기

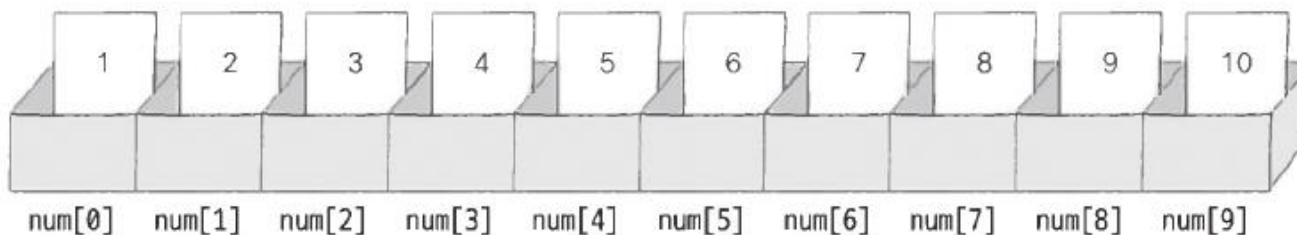
```
1 package array;
2 public class ArrayTest {
3     public static void main(String[] args) {
4         int[] num = new int[] {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
5         for(int i = 0; i < num.length; i++) {
6             System.out.println(num[i]);
7         }
8     }
9 }
```

8. 배열

■ 배열 사용하기

■ 배열 순서는 0번부터

- 4행에서 int형 배열 num을 선언하고 1부터 10까지의 값으로 초기화하였습니다.
- 초기화가 끝난 num 배열은 다음 그림과 같습니다.



- 배열 요소를 하나씩 가져와 출력하기 위해 7행에서 for 반복문을 사용했습니다.
- 배열의 첫 번째 요소 인덱스는 0부터 시작합니다.

■ 배열 사용하기

■ 배열 순서는 0번부터

- 자바의 배열은 배열 길이를 나타내는 length 속성을 가집니다.
- 자바에서 배열 길이는 처음에 선언한 배열의 전체 요소 개수를 의미합니다.
- 전체 길이를 알고 싶은 배열 이름 뒤에 도트(.) 연산자를 붙이고 length 속성을 쓰면 배열 길이를 반환합니다.
- for문의 조건에서 얼마나 반복할지 결정해야 하는데, 배열 요소 끝까지 반복하기 위해 배열 전체 길이(length)를 넣습니다.
- 따라서 num.length 값은 10이 됩니다.
- 이렇게 배열 전체 길이만큼 수행문을 반복해야 할 때는 숫자를 직접 사용하는 것보다 length 속성을 사용하는 것이 좋습니다.
- 연습문제 : 조금 전 실습한 예제의 main() 함수에 `int sum = 0 ;` 을 작성하고, 6행 코드를 수정하여 배열의 모든 요소 합을 계산하는 프로그램을 만들어 보세요.

■ 배열 사용하기

■ 전체 배열 길이와 유효한 요소 값

- 우리가 배열을 사용할 때 처음 선언한 배열 길이만큼 값을 저장해서 사용하는 경우는 많지 않습니다.
- 따라서 전체 배열 길이와 현재 배열에 유효한 값이 저장되어 있는 배열 요소 개수가 같다고 혼동하면 안 됩니다.
- 다음 예제를 한번 살펴보겠습니다.
- Ex) 배열 길이만큼 출력하기

```
1 package array;
2
3 public class ArrayTest2{
4     public static void main(String[] args) {
5         double[] data = new double[5];
6
7         data[0] = 10.0;           // 첫 번째 요소에 값 10.0 대입
8         data[1] = 20.0;           // 두 번째 요소에 값 20.0 대입
9         data[2] = 30.0;           // 세 번째 요소에 값 30.0 대입
```

■ 배열 사용하기

■ 전체 배열 길이와 유효한 요소 값

- 우리가 배열을 사용할 때 처음 선언한 배열 길이만큼 값을 저장해서 사용하는 경우는 많지 않습니다.
- 따라서 전체 배열 길이와 현재 배열에 유효한 값이 저장되어 있는 배열 요소 개수가 같다고 혼동하면 안 됩니다.
- 다음 예제를 한번 살펴보겠습니다.
- Ex) 배열 길이만큼 출력하기

```
10
11     for(int i = 0; i < data.length; i++) {
12         System.out.println(data[i]);
13     }
14 }
15 }
```

■ 배열 사용하기

■ 전체 배열 길이와 유효한 요소 값

- double형으로 길이가 5인 배열을 선언했습니다.
- 자바에서 정수 배열과 실수 배열을 별도로 초기화하지 않고 선언하면 배열의 요소 값은 0으로 초기화됩니다.
- 7~9행을 보면 배열의 첫 번째 요소(data[0])부터 세 번째 요소(data[2])까지만 값을 저장했습니다.
- 11행 for문에서 i가 0부터 배열 길이인 data.length 미만까지 반복하며 배열에 저장된 요소 값을 출력합니다.
- 배열의 네 번째 요소와 다섯 번째 요소에는 값을 저장하지 않았기 때문에 0이 출력되는 것을 알 수 있습니다.
- 즉 배열의 세 번째 요소까지만 유효한 값이 저장된 것이죠.
- 만약 위 코드에서 유효한 값이 저장된 배열 요소만 정확히 출력하려면 새로운 변수를 선언하고 배열 요소 순서대로 값을 저장할 때마다 그 변수 값을 증가시킵니다.
- 그리고 반복문 종료 조건으로 배열의 length 속성이 아닌 해당 변수를 사용하면 됩니다.

■ 배열 사용하기

■ 전체 배열 길이와 유효한 요소 값

- 그러면 유효한 값이 저장된 배열 요소까지만 출력하는 프로그램을 만들어 봅시다.
- 배열의 유효한 요소 값 출력하기

```
1 package array;
2
3 public class ArrayTest3{
4     public static void main(String[] args) {
5         double[] data = new double[5];
6         int size = 0;
7
8         data[0] = 10.0; size++;           // 첫 번째 요소에 값 10.0 대입
9         data[1] = 20.0; size++;           // 두 번째 요소에 값 20.0 대입
10        data[2] = 30.0; size++;           // 세 번째 요소에 값 30.0 대입
11    }
```

■ 배열 사용하기

■ 전체 배열 길이와 유효한 요소 값

- 그러면 유효한 값이 저장된 배열 요소까지만 출력하는 프로그램을 만들어 봅시다.
- 배열의 유효한 요소 값 출력하기

```
12         for(int i = 0; i < size; i++) {  
13             System.out.println(data[i]);  
14         }  
15     }  
16 }
```

- 6행에 유효한 값이 저장된 배열 요소 개수를 저장할 size 변수를 선언했습니다.
- 배열 요소에 순서대로 값을 저장할 때마다 size 변수의 값을 하나씩 증가시킵니다.
- 즉 유효한 값을 저장하고 있는 배열 요소 개수를 알 수 있는 것이죠.
- 따라서 12행 반복문은 전체 배열 길이만큼 반복하는 게 아니라 유효한 요소 개수만큼만 반복합니다

■ 문자 저장 배열 만들기

- 이번에는 문자를 저장하는 배열도 한번 생각해 봅시다.
- 문자 자료형 배열을 만들고 알파벳 대문자를 A부터 Z까지 저장한 후 각 요소 값을 알파벳 문자와 정수 값(아스키 코드 값)으로 출력해 보겠습니다.
- 문자 자료형 배열은 `char[]`로 선언해야 합니다.
- Ex) 알파벳 문자와 아스키 코드 값 출력하기

```
1 package array;
2
3 public class CharArray{
4     public static void main(String[] args) {
5         char[] alphabets = new char[26];
6         char ch = 'A';
7
8         for (int i = 0; i < alphabets.length; i++, ch++) {
9             alphabets[i] = ch; // 아스키 값으로 각 요소에 저장
10        }
```

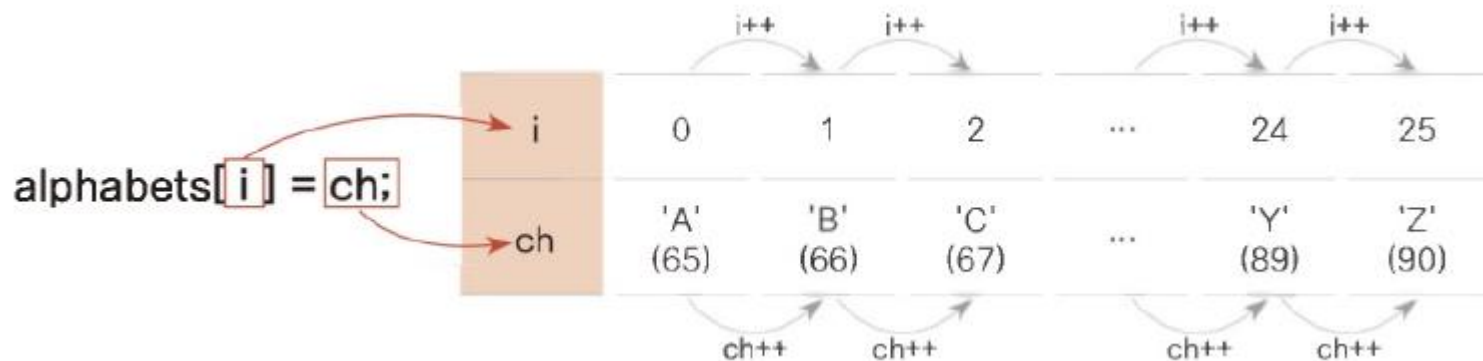
■ 문자 저장 배열 만들기

- 이번에는 문자를 저장하는 배열도 한번 생각해 봅시다.
- 문자 자료형 배열을 만들고 알파벳 대문자를 A부터 Z까지 저장한 후 각 요소 값을 알파벳 문자와 정수 값(아스키 코드 값)으로 출력해 보겠습니다.
- 문자 자료형 배열은 `char[]`로 선언해야 합니다.
- Ex) 알파벳 문자와 아스키 코드 값 출력하기

```
11
12     for(int i = 0; i < alphabets.length; i++) {
13         System.out.println(alphabets[i] + ", " + (int)alphabets[i]);
14     }
15 }
16 }
```

■ 문자 저장 배열 만들기

- 5행에서 대문자 알파벳 26개를 저장하기 위해 문자형 배열을 선언하고, 8행에서 for문을 사용해 각 배열 요소에 알파벳 문자를 저장하였습니다.
- 각 알파벳 문자는 실제 메모리에 아스키 코드 값으로 저장되기 때문에 `ch` 값에 1을 더하면(`ch++`) 1만큼 증가한 값이 배열에 저장됩니다.



- 12행의 for문은 `alphabets` 배열에 저장된 알파벳 문자와 그 문자에 해당하는 아스키 코드 값을 반복하여 출력합니다.
- 13행의 `(int)alphabets[i]` 문장에서 형 변환 연산자 `(int)`는 배열에 저장된 `char`형 문자를 `int`형 정수로 변환합니다.

■ 향상된 for문과 배열

- 자바 5부터 제공되는 향상된 for문(enhanced for loop)은 배열의 처음에서 끝까지 모든 요소를 참조할 때 사용하면 편리한 반복문입니다.
- 향상된 for문은 배열 요소 값을 순서대로 하나씩 가져와서 변수에 대입합니다.
- 따로 초기화와 종료 조건이 없기 때문에 모든 배열의 시작 요소부터 끝 요소까지 실행합니다.

```
for(변수 : 배열){  
    반복 실행문;  
}
```

- 다음 예제를 따라 하며 향상된 for문을 연습해 봅시다.

■ 향상된 for문과 배열

■ Ex) 향상된 for문 사용하기

```
1 package array;
2
3 public class EnhancedForLoop {
4     public static void main(String[] args) {
5         String[] strArray = {"Java", "Android", "C", "JavaScript", "Python"};
6
7         for(String lang: strArray) {
8             System.out.println(lang);
9         }
10    }
11 }
```

■ 연습문제 8-1

- int형 배열의 각 요소에 1 ~ 10의 난수를 대입하고 각 요소의 값을 막대그래프(기호 문자 *를 나열)로 표시하는 프로그램을 작성하자.
- 요소 수는 키보드로 입력한다.
- 마지막에는 인덱스를 10으로 나눈 나머지를 표시할 것.

```
1 package array;
2 import java.util.Random;
3 import java.util.Scanner;
4
5 public class ColumnChart {
6     public static void main(String[] args) {
7         Random rand = new Random();
8         Scanner stdIn = new Scanner(System.in);
9
10        System.out.print("요소 수: ");
11
```

■ 연습문제 8-1

- int형 배열의 각 요소에 1 ~ 10의 난수를 대입하고 각 요소의 값을 막대그래프(기호 문자 *를 나열)로 표시하는 프로그램을 작성하자.

```
12         int n = stdIn.nextInt();           // 요소수 읽기
13         int[] a = new int[n];               // 배열 생성
14
15         for(int i = 0; i < n; i++)
16             a[i] = 1 + rand.nextInt(10); // 1~10 난수
17
18         for(int i = 10; i >= 1; i--) {
19             for(int j = 0; j < n; j++) {
20                 if (a[j] >= i)
21                     System.out.print("* ");
22                 else
23                     System.out.print(" ");
24             }
```

■ 연습문제 8-1

- int형 배열의 각 요소에 1 ~ 10의 난수를 대입하고 각 요소의 값을 막대그래프(기호 문자 *를 나열)로 표시하는 프로그램을 작성하자.

```
25         System.out.println( );
26
27         for(int i = 0; i < 2 * n; i++)
28             System.out.print('-');
29         System.out.println( );
30
31         for(int i = 0; i < n; i++)
32             System.out.print(i % 10 + " ");
33         System.out.println();
34     }
35 }
```

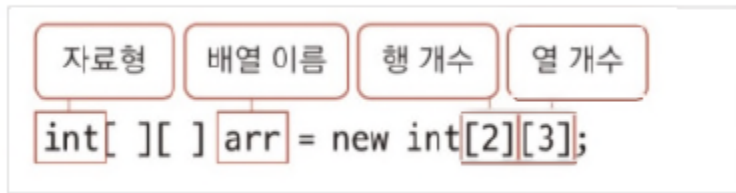

■ 다차원 배열이란?

- 지금까지 배운 배열은 모두 행 하나로 이루어진 '일차원 배열'입니다.
- 수학에서 평면을 나타내기 위해 x , y 좌표를 쓰는 것처럼 프로그램에서도 평면을 구현하기 위해 이차원 배열을 사용할 수 있습니다.
- 예를 들어 바둑이나 체스 게임, 네비게이션 지도 등을 구현할 때 이차원 배열을 활용합니다.
- 삼차원 이상의 배열도 가능합니다.
- 삼차원 배열은 주로 공간을 나타내는 프로그램에서 활용합니다.
- 이렇게 이차원 이상으로 구현한 배열을 '다차원 배열' 이라고 합니다.
- 다차원 배열은 평면이나 공간 개념을 구현하는데 사용합니다.
- 여기에서는 이차원 배열을 살펴보겠습니다.

9. 다차원 배열

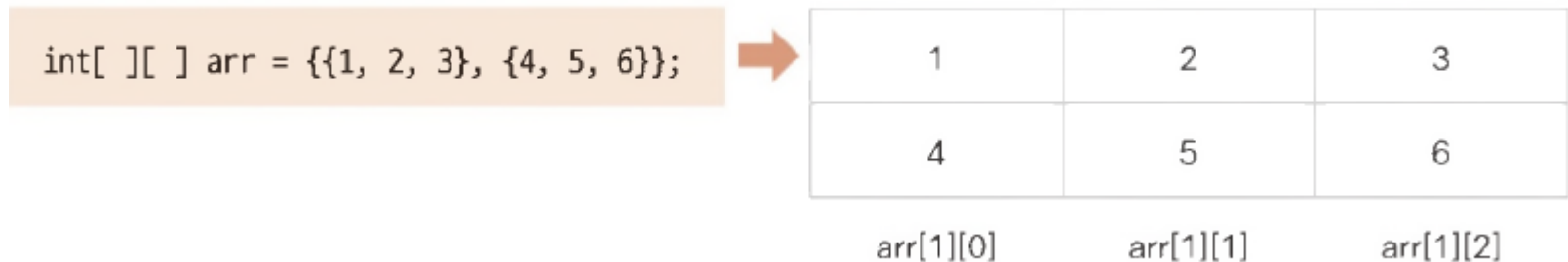
■ 이차원 배열

- 다음은 2행 3열의 이차원 배열을 선언하는 코드와 논리 구조입니다.



arr[0][0]	arr[0][1]	arr[0][2]
arr[1][0]	arr[1][1]	arr[1][2]

- 배열의 모든 요소를 참조하려면 각 행을 기준으로 열 값을 순회하면 됩니다.
- 이차원 배열을 초기화하려면 다음처럼 행과 열 개수에 맞추어서 중괄호 { } 안에 콤마(,)로 구분해 값을 적습니다.
- 이렇게 이차원 배열을 초기화하면 괄호 안에 적은 6개 값이 순서대로 arr 배열의 각 요소에 저장됩니다.



■ 이차원 배열

■ Ex) 이차원 배열 초기화하기

```
1 package array;
2
3 public class TwoDimension {
4     public static void main(String[] args) {
5         int[ ][ ] arr = {{1, 2, 3}, {4, 5, 6}};
6
7         for(int i = 0; i < arr.length; i++) {
8             for(int j = 0; j < arr[i].length; j++) {
9                 System.out.println(arr[i][j]);
10            }
11            System.out.println();    // 행 출력 끝난 후 한 줄 띄움
12        }
13    }
14 }
```

9. 다차원 배열

■ 이차원 배열

- 7~10행의 중첩 for문은 배열 인덱스용으로 i, j 두 변수를 사용하는데 i 는 행을, j 는 열을 가리킵니다.
- 전체 배열 길이인 `arr.length`는 행의 개수를 각 행의 길이 `arr[i].length`는 열의 개수를 나타냅니다.



■ 연습문제 이차원 배열 연습하기

- 알파벳 소문자를 2글자씩 13줄 (13행 2열) 로 출력하는 프로그램을 이차원 배열로 구현해 보세요.

9. 다차원 배열

■ 이차원 배열

- 다음 예제에서 이차원 배열 각행의 길이와 열의 길이를 다시 살펴보겠습니다.
- Ex) 이차원 배열의 길이 출력하기

```
1 package array;
2
3 public class TwoDimension2 {
4     public static void main(String[] args) {
5         int[][] arr = new int[2][3];        // 2행 3열 이차원 배열 선언
6
7         for(int i = 0; i < arr.length; i++) {
8             for(int j = 0; j < arr[i].length; j++) {
9                 System.out.println(arr[i][j]);
10            }
11            System.out.println();           // 행 출력 끝난 후 한 줄 띄움
12        }
```

9. 다차원 배열

■ 이차원 배열

- 다음 예제에서 이차원 배열 각행의 길이와 열의 길이를 다시 살펴보겠습니다.
- Ex) 이차원 배열의 길이 출력하기

```
13         System.out.println(arr.length);
14         System.out.println(arr[0].length);
15     }
16 }
```

- 위 코드를 보면 이차원 배열을 선언만 하고 초기화를 따로 하지 않았기 때문에, 모두 0으로 자동 초기화된 것을 알 수 있습니다.

■ 연습문제 9-1

- 6명의 두 과목 점수(국어, 수학)를 읽어 과목별 평균과 학생별 평균을 구하자.

```
1 package array;
2
3 import java.util.Scanner;
4
5 public class PointTotalization {
6     public static void main(String[] args) {
7         Scanner stdIn = new Scanner(System.in);
8         final int NUMBER = 6;          // 사람수
9         int[ ][ ] point = new int[NUMBER][2];    // 점수
10        int[ ] sumStudent = new int[NUMBER]; // 학생별 점수 합계
11        int[ ] sumSubject = new int[2];          // 각 과목의 점수 합계
12
13        System.out.printf("%d명의 국어, 수학 점수를 입력하세요.\n", NUMBER);
```

9. 다차원 배열

■ 연습문제 9-1

- 6명의 두 과목 점수(국어, 수학)를 읽어 과목별 평균과 학생별 평균을 구하자.

```
14         for(int i = 0; i < NUMBER; i++) {
15             System.out.printf("%2d번 국어: ", i + 1);
16             point[i][0] = stdIn.nextInt();
17             System.out.print("    수학: ");
18             point[i][1] = stdIn.nextInt();
19
20             sumStudent[i] = point[i][0] + point[i][1];           // 학생의 합계
21             sumSubject[0] += point[i][0];                       // 국어 합계
22             sumSubject[1] += point[i][1];                       // 수학 합계
23         }
24
25         System.out.println("No. 국어    수학    평균");
26
```


9. 다차원 배열

■ 연습문제 9-1

- 6명의 두 과목 점수(국어, 수학)를 읽어 과목별 평균과 학생별 평균을 구하자.

```
27         for(int i = 0; i < NUMBER; i++)
28             System.out.printf("%2d%6d%6d%6.1f\n", i + 1, point[i][0],
29                                 point[i][1], (double)sumStudent[i] / 2);
30         System.out.printf("평균%6.1f%6.1f\n",
31                             (double)sumSubject[0] / NUMBER,
32                             (double)sumSubject[1] / NUMBER);
33     }
34 }
```

■ 연습문제 9-2

- 학급 수, 각 학급의 학생 수, 그리고 모든 학생의 점수를 읽어서 합계와 평균을 구하는 프로그램을 작성하자.

```
1 package array;
2
3 import java.util.Scanner;
4
5 public class PointClass {
6     public static void main(String[] args) {
7         Scanner stdIn = new Scanner(System.in);
8
9         System.out.print("학급 수: ");
10        int classNum = stdIn.nextInt( );
11        int[ ][ ] point = new int[classNum][ ];
12        int totNumber = 0;           // 모든 학급의 총 학생 수
13
```

■ 연습문제 9-2

- 학급 수, 각 학급의 학생 수, 그리고 모든 학생의 점수를 읽어서 합계와 평균을 구하는 프로그램을 작성하자.

```
14         for(int i = 0; i < point.length; i++) {
15             System.out.printf("\n%d반의 학생 수: ", i + 1);
16             int num = stdIn.nextInt( );
17             point[i] = new int[num];
18             totNumber += num;
19             for(int j = 0; j < point[i].length; j++) {
20                 System.out.printf("%d반%d번의 점수: ", i + 1, j + 1);
21                 point[i][j] = stdIn.nextInt();
22             }
23         }
24
25         System.out.println("  반 | 합계   평균");
26         System.out.println("-----+-----");
```

■ 연습문제 9-2

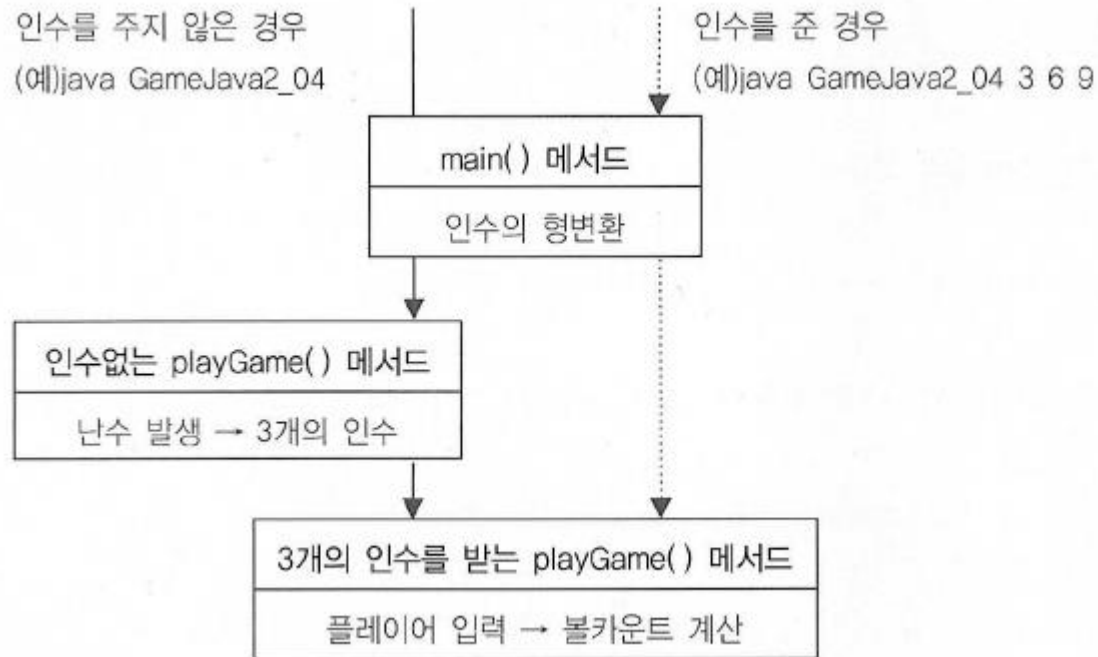
- 학급 수, 각 학급의 학생 수, 그리고 모든 학생의 점수를 읽어서 합계와 평균을 구하는 프로그램을 작성하자.

```
27         int total = 0;
28         for(int i = 0; i < point.length; i++) {
29             int sum = 0;
30             for(int j = 0; j < point[i].length; j++)
31                 sum += point[i][j];
32             total += sum;
33             System.out.printf("%2d반 |%7d%7.1fWn", i + 1, sum,
34                               (double)sum / point[i].length);
35         }
36         System.out.println("-----+-----");
37         System.out.printf("  합 |%7d%7.1fWn", total, (double)total / totNumber);
38     }
39 }
```

10. 숫자 야구 게임 만들기

- 숫자 야구 게임은 `main()` 메서드와 두 개의 `playGame()` 메서드로 구성되어 있습니다.
- `playGame()` 메서드는 3개의 `int`형 변수를 인수로 받는 것과 인수가 없는 두 종류로, 프로그램을 실행시킬 때 3개의 숫자를 인수로 주변 3개의 `int`형 변수를 인수로 받는 `playGame()` 메서드가 호출되고, 인수 없이 프로그램이 실행되면 인수 없는 `playGame()` 메서드가 호출됩니다.
- 3개의 인수를 받는 `playGame()` 메서드는 주어진 값을 컴퓨터가 숨겨둔 숫자 3개로 간주하고 게임을 실행하고, 인수 없는 `playGame()` 메서드는 난수를 발생해서 3개의 숫자를 정한 후 3 개의 인수를 받는 `playGame()` 메서드에 3개의 숫자를 전달합니다.

10. 숫자 야구 게임 만들기



10. 숫자 야구 게임 만들기

- 인수가 없는 playGame() 메서드에서 난수로 숫자 3개를 만들 때는 do-while문을 사용하여 3개의 숫자가 모두 다르도록 조정합니다.
- 먼저 1부터 9 사이의 숫자 하나를 난수로 만들어서 x에 할당합니다.
- 같은 방법으로 y 값을 구한 후 이미 구한 x와 y의 값을 비교하여 같은 경우에는 다시 y 값을 구하는 일을 반복합니다.
- 결국 x와 y 값이 달라질 때까지 난수를 구하는 일을 반복하는 셈이 됩니다.
- 세 번째 숫자인 x의 경우도 마찬가지입니다.
- 다만, z의 경우는 이미 구한 숫자가 x와 y 두 개이기 때문에 x와도 비교하고 y와도 비교해야 합니다.
- 이처럼 어떤 일을 일단 한 번 한 후에 조건을 비교해서 반복 여부를 결정할 때는 do-while문이 편리합니다.

10. 숫자 야구 게임 만들기

```
int x, y, z;  
Random r = new Random( );  
x = Math.abs(r.nextInt( ) % 9) + 1;  
  
do{  
    y = Math.abs(r.nextInt( ) % 9 ) + 1;  
}while(y == x);  
  
do{  
    z = Math.abs(r.nextInt( ) % 9) + 1;  
}while((z == x) || (z == y);
```

- 3개의 인수를 받는 playGame() 메서드에서는 주어진 인수를 com 배열에 저장하고, 사용자가 입력한 3개의 수를 입력받아 usr 배열에 저장합니다.
- 이때 플레이어(사람)가 입력한 값이 0 또는 9보다 큰 숫자나 같은 숫자가 없도록 앞의 난수 발생 때와 비슷한 방법으로 do-while문을 사용해서 반복하도록 합니다.

10. 숫자 야구 게임 만들기

```
do{
    // 키보드로부터 3개의 숫자를 입력받아 각각 usr[0], usr[1], usr[2]에 저장
}while((usr[0] == 0) || (usr[1] == 0) || (usr[2] == 0) ||    ← 입력받은 수가 0인 경우
        (usr[0] > 9) || (usr[1] > 9) || (usr[2] > 9) ||    ← 입력받은 수가 9보다 큰 경우
        (usr[0] == usr[1]) || (usr[1] == usr[2]) || (usr[0] == usr[2])); ← 입력받은 수가 같은 경우
```

- 무사히 3개의 값을 모두 입력받으면, com 배열의 수와 usr 배열의 수를 비교해서 위치와 값이 같으면 strike 값을 증가시키고 값은 같지만 위치가 다르면 ball 값을 증가시키는 방법으로 볼카운트를 구합니다.
- strike 값이 3개면 게임이 종료되고, 그렇지 않은 경우엔 볼카운트를 보여줘서 플레이어에게 다시 한 번 숨겨진 숫자를 추측할 수 있도록 합니다.
- 총 11회의 기회를 주고 그 안에 답을 못 맞히면 적절한 메시지를 출력하고 프로그램을 끝냅니다.

10. 숫자 야구 게임 만들기

```
1  import java.util.*;
2  import java.io.*;
3
4  public class GameJava2_05 {
5      public static int playGame() throws IOException{
6          int x, y, z;
7          Random r = new Random();
8          x = Math.abs(r.nextInt() % 9) + 1;
9          do {
10              y = Math.abs(r.nextInt() % 9) + 1;
11          }while(y == x);          // x값과 y값이 같지 않도록(다를 때까지) 반복
12          do {
13              z = Math.abs(r.nextInt() % 9) + 1;
14          }while((z == x) || (z == y));    // x, y, z 값이 같지 않도록 반복
15          System.out.println(x + ", " + y + ", " + z);
16          return playGame(x, y, z);
17      }
```

10. 숫자 야구 게임 만들기

```
18 public static int playGame(int x, int y, int z) throws IOException{
19     int count;           // 문제를 푼 횟수
20     int strike, ball;
21     int[] usr = new int[3];      // 사용자가 입력한 숫자 3개
22     int[] com = {x, y, z};      // 컴퓨터가 숨긴 숫자 3개
23
24     System.out.println("숫자 야구 게임");
25
26     count = 0;
27
28     do {
29         count++;
30         do {
31             System.out.println("\n카운트: " + count);
32             BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
33             String user;
```

10. 숫자 야구 게임 만들기

```
34         System.out.print("1번째 숫자: ");
35         user = in.readLine();                // 키보드로 1번째 수 입력
36         usr[0] = new Integer(user).intValue();    // 입력받은 문자를
int형 숫자로 변환
37
38         System.out.print("2번째 숫자: ");
39         user = in.readLine();                // 키보드로 2번째 수 입력
40         usr[1] = Integer.valueOf(user).intValue();    // 입력받은 문자를
int형 숫자로 변환
41
42         System.out.print("3번째 숫자: ");
43         user = in.readLine();                // 키보드로 3번째 수 입력
44         usr[2] = new Integer(user).intValue();    // 입력받은 문자를
int형 숫자로 변환
45
```

10. 숫자 야구 게임 만들기

```
46         if((usr[0] == 0) || (usr[1] == 0) || (usr[2] == 0)) {
47             System.out.println("0은 입력하지 마세요. 다시 입력해주세요.");
48         }else if((usr[0] > 9) || (usr[1] > 9) || (usr[2] > 9)) {
49             System.out.println("1부터 9까지의 숫자 중 하나를 입력해주세요. 다시 입력해주세요.");
50         }
51     }while((usr[0] == 0) || (usr[1] == 0) || (usr[2] == 0) ||
52           (usr[0] > 9) || (usr[1] > 9) || (usr[2] > 9) ||
53           (usr[0] == usr[1]) || (usr[1] == usr[2]) || (usr[2] == usr[0]));
54     // 입력받은 답에 이상이 없을 때까지 반복
55
56     strike = ball = 0;           // 볼카운트 초기화
57
58     if(usr[0] == com[0]) strike++;
59     if(usr[1] == com[1]) strike++;
60     if(usr[2] == com[2]) strike++;
```

10. 숫자 야구 게임 만들기

```
61
62         if(usr[0] == com[1]) ball++;
63         if(usr[0] == com[2]) ball++;
64         if(usr[1] == com[0]) ball++;
65         if(usr[1] == com[2]) ball++;
66         if(usr[2] == com[0]) ball++;
67         if(usr[2] == com[1]) ball++;
68
69         System.out.println("Strike: " + strike + " Ball: " + ball); // 볼카운트
출력
70
71         }while((strike < 3) && (count < 11)); // 답을 맞혔거나 10번이상 시
도해서 못맞출 때까지 반복
72
73         return count; // 문제를 맞히려고 시도한 횟수를 반환
74     }
75
```

10. 숫자 야구 게임 만들기

```
76     public static void main(String[] args) throws IOException{
77         int result;
78
79         if(args.length == 3) {           // 인수가 있는 경우
80             int x = Integer.valueOf(args[0]).intValue();
81             // 인수는 String형이므로 int형으로 형변환
82             int y = Integer.valueOf(args[1]).intValue();
83             int z = Integer.valueOf(args[2]).intValue();
84
85             result = playGame(x, y, z); // 인수를 playGame() 메서드에 전달
86         }else {
87             result = playGame();        // 인수없는 playGame() 메서드 호출
88         }
89
90         System.out.println();
```

10. 숫자 야구 게임 만들기

```
91         if(result <= 2) {           // 문제를 푼 횟수에 따라 칭찬 메시지 출력
92             System.out.println("참 잘했어요!");
93         }else if(result <= 5) {
94             System.out.println("잘했어요!");
95         }else if(result <= 9) {
96             System.out.println("보통이네요!");
97         }else {
98             System.out.println("분발하세요!");
99         }
100     }
101 }
```




Thank You
