CHAPTER 05

변수와 제어문을 이용하는 프로그램 만들기

객체지향 이전에 절차지향으로 문제를 해결해봅시다.
이 장에서는 데이터와 로직 작성을 연습합니다.

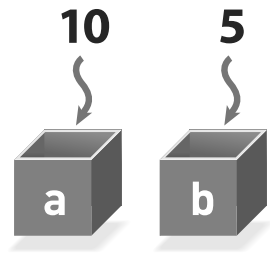
이번 장에서는 앞에서 배운 변수와 제어문을 이용해서 몇 가지의 프로그램을 만드는 방법을 배워봅니다. 이번 장의 목표는 단 하나입니다. 변수와 제어문에 대한 자신감을 늘리고, 나중에 새로운 개념을 배울 때 변수와 제어문이 장애가 되지 않도록 실습해보는 것입니다. 조금 뒤에서부터 나오기 시작하는 객체자료형이라는 개념이 여러분을 흔들기 전에 우선 가장 기본이 되는 변수와 제어문에 대한 기본기를 확실히 하려고 합니다.

1 변수를 언제 써야 하는지 알고 있는가?

변수(Variables)는 데이터를 담아서 보관해 놓기 위한 메모리상의 공간(이하 상자)입니다. 프로그램을 작성하면서 동일한 데이터를 여러 번 사용하려면 변수로 선언해 두고, 나중에 변수의 값만 변경하게 하면 편리해 집니다.

1.1 두 변수의 값을 서로 교환하기

두 변수가 있고 그 값을 서로 맞바꿔야 한다면 어떻게 만드시겠습니까? 즉 'int a = 10;', 'int b = 5;'라는 변수를 선언한 다음 그 값을 서로 변경해야 한다면 어떻게 풀어야 할까요?



두 상자의 내용물을 바꾸려면?

그림 1

생각해보면 우리의 실제 생활과 마찬가지입니다. 데이터를 담은 상자의 내용물을 다른 곳으로 옮기려면 하나의 임시 보관 상자가 필요합니다. 따라서 두 변수의 내용물을 변경하려면 두 개의 데이터가 선언된 변수 외에도 임시로 사용할 변수의 선언이 추가로 필요합니다.

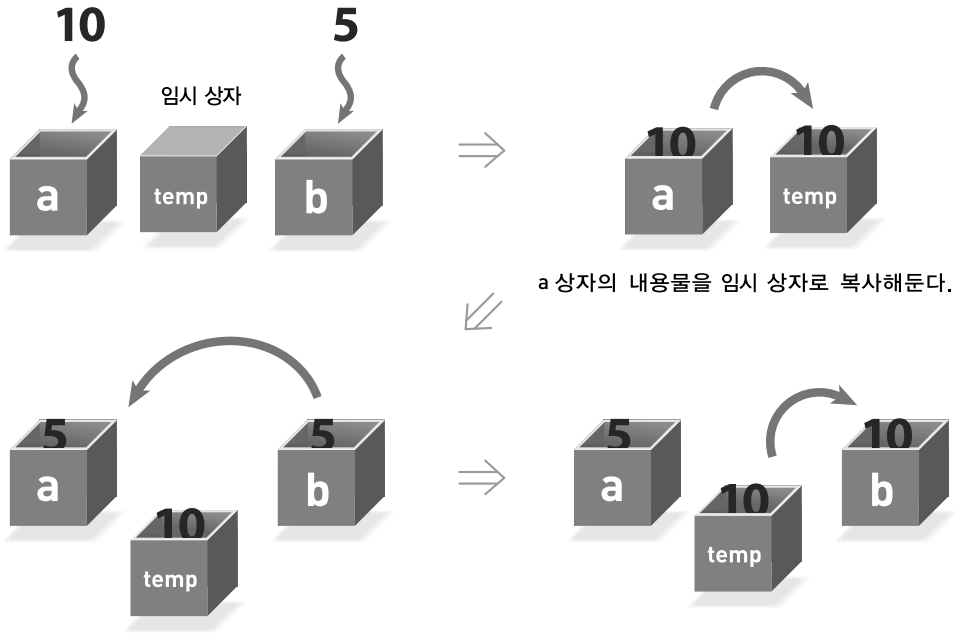


그림 2

예제 | 두 변수의 내용물 바꿔치기하기

```
public class ChangeEx {
	public static void main(String[] args) {

		int a = 10;
		int b = 5;

		//임시로 사용할 변수
		int temp = 0;
		//a의 내용물을 temp로 옮긴다.
		temp  = a;
		//b의 내용물을 a로 옮긴다.
		//a의 내용물은 temp에 들어가 있으니 안전
		a = b;
		//temp의 내용물을 b로 옮긴다.
		b = temp;
		System.out.println("a: "+a);
		System.out.println("b: "+b);

	}
}
```

```
a: 5 ← 원래 b의 값
b: 10 ← 원래 a의 값
```

위의 소스를 보시면 temp라는 임시로 데이터를 저장할 변수를 하나 선언해 주고 데이터를 잠시 보관하고 있습니다. 데이터를 옮기는 것은 마치 물컵의 내용물을 바꿔치기하는 것과 비슷합니다.

1_ a 변수의 내용물을 temp라는 컵으로 옮긴다.

2_ b 변수의 내용물을 a에다가 붓는다.

3_ temp에 있는 a의 내용물을 b로 옮긴다.

1.2 복리 계산기: 변수의 누적

이번에는 돈 계산을 하는 데 있어서 필요한 프로그램을 하나 만들어 보도록 합니다. 변수와 루프를 이용해서 복리를 계산하는 프로그램을 작성해 보도록 합니다(여기서 이자는 매년 계산되는 것을 기준으로 하도록 하겠습니다). 우선 다음해의 전체 금액은 다음과 같이 계산됩니다.

잔액 = 원금 + (원금 × 이자율)

복리는 이것이 여러 번 반복되는 형태라고 할 수 있습니다. 복리는 위의 공식에서 잔액이 다음해의 원금이 되는 방식입니다. 즉 여러 번 반복이 될 것이므로 루프를 이용하면 됩니다. 원금을 money라고 선언하도록 하겠습니다. 금리는 전체 금액의 10%일 경우에는 0.1로 보면 될 듯합니다. 예치 기간(금액을 맡긴 기간) 데이터 역시 필요합니다.

코드

```
//원금
double money  = 10000;

//10%의 금리
double rate = 0.1;

//예치 기간
int year = 10;
```

해마다 복리로 적용되는 금액은 다시 다음해의 원금이 되므로 변수를 이용하면 다음과 같이 작성할 수 있습니다.

```
money = money + (money * rate);
```

이제 이 로직을 이용해서 루프를 사용하면 다음과 같이 처리됩니다.

예제 | 10년간 10% 복리 계산

```
public class CompoundInterest {
	public static void main(String[] args) {
		//원금
		double money  = 10000;

		//10%의 금리
		double rate = 0.1;

		//예치 기간
		int year = 10;
```

```
                for(int i  = 1; i <= year ; i++){
                        money = money  + (money * rate);
                        System.out.println(i +"년 후의 금액: " +  money);
                }
        }
}
```

```
1년 후의 금액: 11000.0
2년 후의 금액: 12100.0
3년 후의 금액: 13310.0
4년 후의 금액: 14641.0
5년 후의 금액: 16105.1
6년 후의 금액: 17715.61
7년 후의 금액: 19487.171000000002
8년 후의 금액: 21435.888100000004
9년 후의 금액: 23579.476910000005
10년 후의 금액: 25937.424601000006
```

이 문제를 프로그램으로 만들려면 변수가 할당 연산자(=)의 왼편인지, 오른편인지에 따라서 다르게 취급된다는 사실을 아는 것이 가장 중요한 요소입니다. 할당 연산자(=)의 왼편에는 결과를 담는 대상 상자가 오며 오른편에는 상자의 내용물을 꺼내는 곳이라는 사실을 이용하셔야만 합니다.

2 순환문의 활용 용도를 잘 알고 있는가?

순환문을 처리하면서 반드시 해봐야 하는 예제는 구구단처럼 반복적인 결과를 만들어 내는 프로그램과 데이터를 누적하는 프로그램입니다. 다음 장에서 배우는 배열과 같은 자료구조와 같이 사용되는 경우가 많으므로 반드시 실습하시기 바랍니다.

2.1 순환문의 기본: 구구단

다음 만들어 보고 싶은 내용은 원하는 구구단을 입력해서 화면에 출력하는 것입니다. 간단히 결과를 먼저 생각해보면 다음과 같습니다.

```
출력하고 싶은 단을 입력해 주세요.
19
19 * 1 = 19
19 * 2 = 38
19 * 3 = 57
19 * 4 = 76
19 * 5 = 95
19 * 6 = 114
19 * 7 = 133
19 * 8 = 152
19 * 9 = 171
```

위의 같은 프로그램을 작성하고 싶다면 어떻게 해야 할까요? 우선 좀 막막하게 느껴지신다면 여러분이 할 수 있는 일부터 하나씩 생각해 보시는 것이 좋습니다. 여러분이 앞에서 배운 개념 중에 적용할 만한 개념이 무엇이 있을지 정리해 보도록 합니다.

- 사용자의 입력은 변수로 받는다.
- 여러 번 반복해야 하는 작업은 루프를 이용한다.
- Scanner라는 것을 이용하면 사용자의 키보드로부터 입력되는 값을 받아들일 수 있다.

위의 내용을 잘 엮는 훈련이 필요합니다. 프로그램을 짤 때 자신이 없다면 우선 추천해드리고 싶은 방법은 자신이 할 수 있는 부분만이라도 정확하게 만들어 보는 것입니다.

■ 막힐 때에는 할 수 있는 것만 생각합니다.

여러분이 이 프로그램을 작성할 때 할 수 있는 것이 무엇일까요? 우선은 클래스를 선언하고 main 메소드라는 것을 만드는 작업은 할 수 있습니다.

예제 | 클래스를 선언하고 메인 메소드 만들기

```
public class GugudanEx {
	public static void main(String[] args) {

	}
}
```

그다음으로 여러분이 할 수 있는 작업이 또 없나요? 만일 잘 모르겠거든 우선은 프로그램에서 준비물과 로직을 적는 부분을 구분해주면 편리합니다.

예제 코드에서 필요한 준비물들과 로직을 작성하는 영역을 구분해줍니다.

```
public class GugudanEx {
	public static void main(String[] args) {
		//준비물
		//로직
	}
}
```

영역을 구분한 다음은 로직을 글로 작성하는 훈련을 해봅니다. 예를 들어 지금의 경우에는 다음과 같이 작성할 수 있을 듯합니다.

1_ 화면에 "출력하고 싶은 단을 입력해 주세요."라는 메시지를 뿌린다.
2_ 키보드에서 사용자가 원하는 단을 입력을 받는다. 이때 숫자로 입력받는다. 문자로 잘못 입력하는 경우는 지금 고려하지 않는다.
3_ 사용자가 입력한 단의 1에서 9까지 루프를 돌린다.

로직이 반드시 완전할 필요는 전혀 없습니다. 로직은 그냥 여러분이 코딩을 만드는 데 있어서 도움이 되는 부분을 글로 먼저 정리하는 의미만을 가진다고 생각하시면 됩니다. 개인적으로 저는 성격이 급한 편이라 늘 이렇게 작성하는 로직이 허술한 경우가 많기 때문에 수정하고 다시 수정하는 작업을 많이 하는 편입니다.

로직을 대강이라도 작성해 두었다면 이제는 실제로 코드를 작성할 차례입니다. 여러분이 할 수 있는 것과 없는 것을 잘 판단해야 합니다. 우선은 1번째는 가능할 듯합니다.

예제

```
public class GugudanEx {
	public static void main(String[] args) {
		//준비물

		//로직

		System.out.println("출력하고 싶은 단을 입력해 주세요.");

	}
}
```

우선은 이 상태에서도 프로그램은 동작합니다. 2번째 로직을 완성하려면 사용자의 입력을 받아들이는 Scanner가 필요할 겁니다. 사용자의 입력을 숫자로 받는 것은 이미 실습한 적이 있기 때문에 Scanner를 로직이 실행하는 데 있어서 필요한 준비물로 선언해 줍니다. 이때 Ctrl + ⇧Shift + O 를 이용해서 import 구문을 정확히 처리하도록 합니다.

예제

```
import java.util.Scanner;

public class GugudanEx {
	public static void main(String[] args) {
		//준비물
		Scanner scanner = new Scanner(System.in);

		//로직

		System.out.println("출력하고 싶은 단을 입력해 주세요.");


	}
}
```

2번째 로직을 완성하려면 사용자의 입력을 받아야 합니다. 사용자의 입력은 변수의 선언 시에도 말씀드렸습니다만 변수를 하나 선언해서 처리하시는 것이 좋습니다.

예제

```
import java.util.Scanner;

public class GugudanEx {
	public static void main(String[] args) {
		//준비물
		Scanner scanner = new Scanner(System.in);

		//로직

		System.out.println("출력하고 싶은 단을 입력해 주세요.");

		int target = scanner.nextInt();

	}
}
```

이제 프로그램을 실행하면 화면에 메시지가 출력된 후에 키보드에서 입력한 숫자가 target이라는 변수의 값으로 들어가게 됩니다.

3번째 로직을 완성하는 데 있어서 하나의 변수가 여러 번 사용된다면 루프를 고려해야 합니다. for 루프와 while 루프 중에서 판단하실 사항은 단 하나, 몇 번 돌아야 하는지 명확하다면 무조건 for 루프를 이용하는 것이 일반적이고, 정확히 알 수 없을 때에는 while 루프가 일반적이라는 겁니다. 구구단의 경우라면 모든 단을 1에서부터 9까지 돌리기 때문에 for 루프가 적합합니다.

예제

```
import java.util.Scanner;

public class GugudanEx {
	public static void main(String[] args) {
		//준비물
		Scanner scanner = new Scanner(System.in);

		//로직

		System.out.println("출력하고 싶은 단을 입력해 주세요.");

		int target = scanner.nextInt();

		for(int i = 0; i <= 9; i++){
```

```
                    System.out.println(target + " : " + i);
                }
            }
}
```

```
출력하고 싶은 단을 입력해 주세요.
19
19 : 0
19 : 1
19 : 2
19 : 3
19 : 4
19 : 5
19 : 6
19 : 7
19 : 8
19 : 9
```

이제 마지막으로, 연산을 해서 결과를 출력하는 작업만 남았습니다. System.out.println( ) 안쪽에 위의 소스를 약간만 수정해주면 될 듯합니다.

예제

```
for(int i = 1; i<= 9; i++){
        System.out.println(target +" * " + i +" = " + (target*i));
}
```

위와 같이 수정하게 되면 화면에는 다음과 같이 출력됩니다.

```
출력하고 싶은 단을 입력해 주세요.
19 Enter
19 * 1 = 19
19 * 2 = 38
19 * 3 = 57
19 * 4 = 76
19 * 5 = 95
19 * 6 = 114
19 * 7 = 133
19 * 8 = 152
19 * 9 = 171
```

어떤가요? 처음에는 조금 난감해 보이지만 자신이 할 수 있는 부분을 정확히 찾아내면 좀 더 쉽게 프로그램을 작성할 수 있습니다. 이렇게 프로그램의 시작은 먼저 자신이 원하는 기능에 어떻게 접근하고 자신이 할 수 있는 것과 없는 것을 파악하는 데에서 시작합니다.

2.2 1에서부터 100까지의 합 구하기

이번에 만들어 볼 예제는 1에서 100까지의 합을 구하는 프로그램을 작성하는 겁니다. 이 프로그램을 제대로 작성하려면 다음과 같은 내용을 알고 있어야 합니다.

- 변수의 값을 사용하는 방법과 변수의 범위를 명확히 알아야 한다.
- 루프를 이용해서 변하는 값을 활용하는 방법을 알아야 한다.

우선 어떤 값을 기존의 변수에 더하는 과정부터 한 번 더 살펴보도록 합니다.

```
int x = 10;
x = x + 1; (혹은 x += 1;)
```

앞의 코드를 보면 x라는 이름의 변수를 하나 선언하고, 변수의 값을 처음에 10으로 할당(Assign)했습니다. 물론 변수가 최초로 선언되기 때문에 int라고 해서 정확히 타입을 명시해주고 있습니다.

두 번째 라인에서는 변수가 =의 왼쪽에 있는가, 오른쪽에 있는가에 따라서 다르게 동작한다는 것을 아실 필요가 있습니다. 변수가 =의 왼편이라면 변수를 담는 상자를 의미하고, 변수가 =의 오른쪽이라면 상자에서 내용물을 사용하는 것이라는 사실을 다시 한번 상기하시면 됩니다. 따라서 'x = x + 1;'이라는 코드는 기존의 x가 가진 값에 1이 더해진 결과를 다시 원래의 변수 x에 담는 것을 의미합니다. 결과는 당연히 x의 값이 10이 아닌 11이 되어 있습니다.

그럼 아래와 같이 작성한 코드는 어떤 결과를 만들어낼지 생각해 보시기 바랍니다.

```
int x = 0;
x = x + 1; ← 변수 x의 값은 1로 변경됩니다.
x = x + 2; ← 변수 x의 값은 2로 변경됩니다.
```

위의 코드는 원래의 값이 계속해서 누적되는 방법을 보여주고 있습니다. 따라서 1에서부터 10까지의 합을 만들어야만 한다면 다음과 같은 코드로 작성해볼 수 있습니다.

```
int x = 0;
x = x + 1;
x = x + 2;
...
x = x + 10;
```

위처럼 코드를 작성하게 되면 x라는 변수에는 계속해서 값이 누적되어서 변경되기 때문에 10까지의 합의 누적된 결과를 생성할 수 있습니다. 이제 마지막으로 남은 작업은 1, 2, 3, 4, ...와 같이 나오는 코드를 루프를 이용해서 자동으로 변경되게 해주는 작업입니다. 이 작업은 결과적으로 코드의 양을 줄이는 작업이라고 할 수 있을 겁니다.

예제 | 1에서 10까지의 합을 구하는 코드

```
public class SumTest {

	public static void main(String[] args) {

		int x = 0;

		for(int i = 0; i <= 10; i++){
			x = x + i;
		}

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

```
55
```

위의 코드를 보면 루프를 이용해서 계속해서 변경되는 i 값이 변수에 누적되는 것을 볼 수 있습니다. 1에서 100까지의 합은 10까지가 아닌 100까지의 합으로만 변경해 주면 됩니다.

2.3 인쇄된 모양의 구구단 만들기

앞의 예제에서는 간단히 한 단의 모양만을 출력해 보았지만 실제로 우리가 눈으로 접하는 구구단은 그 모양이 약간 다릅니다.

```
2 * 1 = 2  3 * 1 = 3  4 * 1 = 4  5 * 1 = 5  6 * 1 = 6  7 * 1 = 7  8 * 1 = 8  9 * 1 = 9
2 * 2 = 4  3 * 2 = 6  4 * 2 = 8  5 * 2 = 10 6 * 2 = 12 7 * 2 = 14 8 * 2 = 16 9 * 2 = 18
2 * 3 = 6  3 * 3 = 9  4 * 3 = 12 5 * 3 = 15 6 * 3 = 18 7 * 3 = 21 8 * 3 = 24 9 * 3 = 27
2 * 4 = 8  3 * 4 = 12 4 * 4 = 16 5 * 4 = 20 6 * 4 = 24 7 * 4 = 28 8 * 4 = 32 9 * 4 = 36
2 * 5 = 10 3 * 5 = 15 4 * 5 = 20 5 * 5 = 25 6 * 5 = 30 7 * 5 = 35 8 * 5 = 40 9 * 5 = 45
2 * 6 = 12 3 * 6 = 18 4 * 6 = 24 5 * 6 = 30 6 * 6 = 36 7 * 6 = 42 8 * 6 = 48 9 * 6 = 54
2 * 7 = 14 3 * 7 = 21 4 * 7 = 28 5 * 7 = 35 6 * 7 = 42 7 * 7 = 49 8 * 7 = 56 9 * 7 = 63
2 * 8 = 16 3 * 8 = 24 4 * 8 = 32 5 * 8 = 40 6 * 8 = 48 7 * 8 = 56 8 * 8 = 64 9 * 8 = 72
2 * 9 = 18 3 * 9 = 27 4 * 9 = 36 5 * 9 = 45 6 * 9 = 54 7 * 9 = 63 8 * 9 = 72 9 * 9 = 81
```

이번에 만들어 볼 구구단은 위와 같은 모습입니다. 어떤 방식으로 만들어야 할지 한번 생각해 보도록 합니다. 우선 여러분이 지금까지 공부한 사항을 생각해 보면 다음과 같습니다.

- 규칙적으로 반복이 필요하다면 루프를 이용한다.
- System.out.println( )을 이용하면 라인이 변경되면서 출력된다.

이 두 가지의 사항에 사소한 지식 한, 두 가지만 추가되면 가능할 듯합니다. 우선 여러분이 알아야 하는 추가적인 기능을 알아보도록 합니다.

- System.out.println( );은 아래와 같이 변경할 수 있다.
  - System.out.print(...);
  - System.out.println( );

System.out.println( )이 내용물을 출력하고 나서 라인은 변경한다면 System.out.print( ) 메소드는 내용물을 출력하고 나서 라인을 변경하지 않습니다. 또한, System.out.println( )은 아무런 데이터가 없을 때에는 단순히 줄 바꾸는 효과만 있습니다. 예제로 보면 다음과 같습니다.

예제 | System.out.println( )은 단순 줄 바꿈 할 때 사용합니다.

```
public class PrintEx {
    public static void main(String[] args) {
        System.out.println("AAA");
        System.out.print("BBB");
        System.out.println();
        System.out.println("CCC");
    }
}
```

```
AAA
BBB
CCC
```

위의 결과를 보면 System.out.println( )과 System.out.print( ) + System.out.println( )을 합친 결과가 동일한 것을 보실 수 있습니다. 따라서 구구단을 화면에 출력하고 싶다면 각 라인을 System.out.print( )로 출력하고, 라인이 변경되는 부분만 System.out.println( )으로 변경해주면 됩니다.

예제 | 구구단의 중간 형태

```
public class GugudanEx2 {
    public static void main(String[] args) {
        for(int i = 1; i <=  9; i++){
            System.out.print("2 * 1 = 2 3 * 1 = 3 4 * 1=....");
            System.out.println();
        }
    }
}
```

```
2 * 1 = 2 3 * 1 = 3 4 * 1=....
2 * 1 = 2 3 * 1 = 3 4 * 1=....
2 * 1 = 2 3 * 1 = 3 4 * 1=....
2 * 1 = 2 3 * 1 = 3 4 * 1=....
2 * 1 = 2 3 * 1 = 3 4 * 1=....
2 * 1 = 2 3 * 1 = 3 4 * 1=....
2 * 1 = 2 3 * 1 = 3 4 * 1=....
2 * 1 = 2 3 * 1 = 3 4 * 1=....
2 * 1 = 2 3 * 1 = 3 4 * 1=....
```

앞의 코드를 보면 각 단의 1, 2, 3, ...을 하나의 라인으로 출력하고 출력한 후에 라인이 바뀌는 것을 볼 수 있습니다. 이제 여기서 아래로 떨어지면서 변경되는 부분을 i라는 변수로 변경해 보면 다음과 같은 결과를 볼 수 있습니다.

예제

```
public class GugudanEx2 {
    public static void main(String[] args) {
        for(int i = 1; i <=  9; i++){
            System.out.print("2 * "+i+" = 2 3 * "+i+" = 3 4 * "+i+"=....");
            System.out.println();
        }
    }
}
```

```
2 * 1 = 2 3 * 1 = 3 4 * 1=....
2 * 2 = 2 3 * 2 = 3 4 * 2=....
2 * 3 = 2 3 * 3 = 3 4 * 3=....
2 * 4 = 2 3 * 4 = 3 4 * 4=....
2 * 5 = 2 3 * 5 = 3 4 * 5=....
2 * 6 = 2 3 * 6 = 3 4 * 6=....
2 * 7 = 2 3 * 7 = 3 4 * 7=....
2 * 8 = 2 3 * 8 = 3 4 * 8=....
2 * 9 = 2 3 * 9 = 3 4 * 9=....
```

i의 값을 루프에서 이용했더니 각 단의 변경되는 모습이 보이기 시작합니다. 이제 결과를 변경하는 작업이 남았습니다. 각 단의 모습을 제대로 출력해 주려면 다음과 같은 형태로 출력되도록 작업해야 합니다.

```
2 * 1 = 2  3 * 1 = 3  4 * 1 = 4  5 * 1 = 5  6 * 1 = 6  7 * 1 = 7  8 * 1 = 8  9 * 1 = 9
```

한 라인을 유심히 보면 이 역시도 루프를 이용해서 옆으로 출력해 주면 될 듯합니다.

```
System.out.print("2 * 1 = 2");
System.out.print("3 * 1 = 3");
System.out.print("4 * 1 = 4");
```

print( )를 이용했기 때문에 위의 결과는 아래쪽으로 출력되지 않고 옆으로 출력되게 됩니다. 따라서 위의 코드를 루프를 이용해 주면 다음과 같은 형태가 됩니다.

코드

```
for(int j = 2; j <=9 ; j++){
        System.out.print(j+" * "+ 1 +" = " + (j* 1)+"  ");
}
```

```
2 * 1 = 2  3 * 1 = 3  4 * 1 = 4  5 * 1 = 5  6 * 1 = 6  7 * 1 = 7  8 * 1 = 8  9 * 1 = 9
```

일부러 i라는 변수 대신에 j라는 변수를 사용했습니다. 마지막 마무리는 루프를 결합해서 하나의 형태로 만들어 주는 겁니다. 즉 완성된 코드는 다음과 같은 형태가 됩니다. 이렇게 루프 안쪽에서 다시 루프가 도는 것을 '이중 루프'라고 합니다.

예제 | 구구단의 완성 형태

```
public class GugudanEx2 {

        public static void main(String[] args) {

                for(int i = 1; i <=  9 ; i++){ // 바깥쪽 루프
                        for(int j = 2; j <=9 ; j++){ //안쪽 루프
                          System.out.print(j+" * "+ i +" = " + (j*i) + "  ");
                        }
                        System.out.println();
                }

        }
}
```

```
2 * 1 = 2  3 * 1 = 3  4 * 1 = 4  5 * 1 = 5  6 * 1 = 6  7 * 1 = 7  8 * 1 = 8  9 * 1 = 9
2 * 2 = 4  3 * 2 = 6  4 * 2 = 8  5 * 2 = 10  6 * 2 = 12  7 * 2 = 14  8 * 2 = 16  9 * 2 = 18
2 * 3 = 6  3 * 3 = 9  4 * 3 = 12  5 * 3 = 15  6 * 3 = 18  7 * 3 = 21  8 * 3 = 24  9 * 3 = 27
2 * 4 = 8  3 * 4 = 12  4 * 4 = 16  5 * 4 = 20  6 * 4 = 24  7 * 4 = 28  8 * 4 = 32  9 * 4 = 36
2 * 5 = 10  3 * 5 = 15  4 * 5 = 20  5 * 5 = 25  6 * 5 = 30  7 * 5 = 35  8 * 5 = 40  9 * 5 = 45
2 * 6 = 12  3 * 6 = 18  4 * 6 = 24  5 * 6 = 30  6 * 6 = 36  7 * 6 = 42  8 * 6 = 48  9 * 6 = 54
2 * 7 = 14  3 * 7 = 21  4 * 7 = 28  5 * 7 = 35  6 * 7 = 42  7 * 7 = 49  8 * 7 = 56  9 * 7 = 63
2 * 8 = 16  3 * 8 = 24  4 * 8 = 32  5 * 8 = 40  6 * 8 = 48  7 * 8 = 56  8 * 8 = 64  9 * 8 = 72
2 * 9 = 18  3 * 9 = 27  4 * 9 = 36  5 * 9 = 45  6 * 9 = 54  7 * 9 = 63  8 * 9 = 72  9 * 9 = 81
```

라인의 간격이 좀 안 맞기는 하지만 거의 모습을 갖춘 것을 볼 수 있습니다. 마지막으로 공백 간격을 '\t(탭)'를 이용해서 변경해 보도록 합니다.

예제

```
for(int i = 1; i <=  9 ; i++){
        for(int j = 2; j <=9 ; j++){
                System.out.print(j+" * "+ i +" = " + (j* i)+"\t");
        }
        System.out.println();
}
```

```
2 * 1 = 2   3 * 1 = 3   4 * 1 = 4   5 * 1 = 5   6 * 1 = 6   7 * 1 = 7   8 * 1 = 8   9 * 1 = 9
2 * 2 = 4   3 * 2 = 6   4 * 2 = 8   5 * 2 = 10  6 * 2 = 12  7 * 2 = 14  8 * 2 = 16  9 * 2 = 18
2 * 3 = 6   3 * 3 = 9   4 * 3 = 12  5 * 3 = 15  6 * 3 = 18  7 * 3 = 21  8 * 3 = 24  9 * 3 = 27
2 * 4 = 8   3 * 4 = 12  4 * 4 = 16  5 * 4 = 20  6 * 4 = 24  7 * 4 = 28  8 * 4 = 32  9 * 4 = 36
2 * 5 = 10  3 * 5 = 15  4 * 5 = 20  5 * 5 = 25  6 * 5 = 30  7 * 5 = 35  8 * 5 = 40  9 * 5 = 45
2 * 6 = 12  3 * 6 = 18  4 * 6 = 24  5 * 6 = 30  6 * 6 = 36  7 * 6 = 42  8 * 6 = 48  9 * 6 = 54
2 * 7 = 14  3 * 7 = 21  4 * 7 = 28  5 * 7 = 35  6 * 7 = 42  7 * 7 = 49  8 * 7 = 56  9 * 7 = 63
2 * 8 = 16  3 * 8 = 24  4 * 8 = 32  5 * 8 = 40  6 * 8 = 48  7 * 8 = 56  8 * 8 = 64  9 * 8 = 72
2 * 9 = 18  3 * 9 = 27  4 * 9 = 36  5 * 9 = 45  6 * 9 = 54  7 * 9 = 63  8 * 9 = 72  9 * 9 = 81
```

루프가 한 번 순환할 때 내부에 있는 루프가 다시 한 번 순환하는 구조라고 생각하시면 됩니다. 구구단은 이중 루프에 대한 좋은 예제가 될 수 있기에 추가해 보았습니다.

2.4 원하는 만큼 숫자를 계속해서 입력받아서 더하기

프로그램을 작성하다 보면 때로는 루프를 얼마나 돌아야 하는지 판단하기 어려운 경우가 많습니다. 주로 이런 경우는 처리해야 하는 데이터의 양이 가변적인 경우입니다. 예제의 목표는 사용자가 입력한 숫자를 가늠할 수 없는 상황이기 때문에 while 루프를 이용해서 작업하는 것이 일반적입니다.

```
원하는 만큼 숫자를 입력, 종료는 -1
10
20
30
-1
sum : 60
```

2.4.1 프로그램을 만들려면 로직과 준비물로 나누어 봅니다.

우선은 프로그램이 어떻게 작성되어야 할지 잘 모르겠거든 main 메소드 안에 C 언어처럼 로직을 정리해보는 것이 도움이 됩니다.

예제 | 준비물과 로직만을 작성해본 InputEx

```
import java.util.Scanner;

public class InputEx {

	public static void main(String[] args) {

		//준비물

		//로직
		//사용자에게 문자를 입력하게 메시지를 보여준다.
		//루프를 돌리기 시작한다.
		//사용자가 숫자를 입력한다.
		//입력한 숫자를 계속해서 누적해서 쌓는다.

	}
}
```

아직 로직이 완성되지는 않았습니다만, 대강은 위와 같은 흐름으로 적용될 겁니다. 루프를 멈추게 하고 싶다면 사용자가 -1을 입력했을 때 처리하는 방법을 추가해주면 됩니다. 로직은 아마도 다음과 같이 정리할 수 있습니다.

예제 |

```
//로직
//사용자에게 문자를 입력하게 메시지를 보여준다.
//루프를 돌리기 시작한다.
//사용자가 숫자를 입력한다.
//사용자가 입력한 수가 -1이라면 루프를 벗어난다.
//입력한 숫자를 계속해서 누적해서 쌓는다.
```

이제 로직에 대한 흐름이 정리되었으면 이것을 가지고 프로그램을 작성해 주어야 합니다. 로직을 구현하다가 무언가 필요한 것이 있다면 준비물로 다시 가면 됩니다.

예제 | 중간 형태의 InputEx

```
import java.util.Scanner;

public class InputEx {

	public static void main(String[] args) {

		//준비물
		Scanner scanner = new Scanner(System.in);

		//로직
		//사용자에게 문자를 입력하게 메시지를 보여준다.
		//루프를 돌리기 시작한다.
		//사용자가 숫자를 입력한다.
		//사용자가 입력한 수가 -1이라면 루프를 벗어난다.
		//입력한 숫자를 계속해서 누적해서 쌓는다.

		System.out.println("원하는 만큼 숫자를 입력, 종료는 -1");

		while(true){
			int userInput = scanner.nextInt();
		}
	}
}
```

우선은 간단히 사용자가 키보드를 통해서 계속해서 데이터를 넣어줄 수 있도록 작성해봅니다. 키보드에서 데이터를 입력받으려면 Scanner가 필요하기 때문에 준비물로 설정했습니다. 위의 코드를 보면 입력을 받기는 하지만 누적해서 쌓지 못하고 있고 –1일 경우에 루프를 벗어나는 로직이 없는 상태입니다. 따라서 계속해서 누적해서 결과를 쌓을 변수 sum과 –1인 경우에 루프를 벗어나는 로직을 적용하면 다음과 같은 형태가 됩니다.

예제 | 완성된 형태의 InputEx

```
import java.util.Scanner;

public class InputEx {

	public static void main(String[] args) {

		//준비물
```

```
        Scanner scanner = new Scanner(System.in);

        //로직
        //사용자에게 문자를 입력하게 메시지를 보여준다.
        //루프를 돌리기 시작한다.
        //사용자가 숫자를 입력한다.
        //사용자가 입력한 수가 -1이라면 루프를 벗어난다.
        //입력한 숫자를 계속해서 누적해서 쌓는다.

        System.out.println("원하는 만큼 숫자를 입력, 종료는 -1");

        int sum = 0;

        while(true){
                int userInput = scanner.nextInt();

                if(userInput == -1){
                        break;
                }

                sum = sum + userInput;
        }
        System.out.println("sum : " + sum);
    }
}
```

프로그램을 만들다 보면 어떻게 접근해야 하는지 갈피를 잡기 어려운 문제들을 만나게 됩니다. 거기에서 어떻게 자신만의 방식으로 문제에 접근할 수 있는가가 프로그래밍을 배우는 과정이라고 생각합니다. 그 방법을 때로는 금방 찾을 수도 있고 그렇지 못할 때도 있습니다. 시간의 차이가 있기는 해도 과정은 마찬가지라고 생각합니다. 위와 같은 문제 역시 처음에는 좀 낯설어 보이고, 복잡해 보이지만 결국 글로 정리하면 할수록 문제가 단순해지는 것을 알 수 있습니다.

2.4.2 윤년 계산기를 만들어 봅시다.

뒤에서 배울 날짜와 시간을 Java에서 제공하는 기능들을 공부한다면 윤년 계산을 직접 할 필요는 없게 됩니다만, 모바일 환경이나 다른 프로그래밍으로 달력 프로그램을 만들려고 한다면 때로는 로직을 이용해서 윤년을 계산하는 방식이 필요할지도 모릅니다. 우선 윤년을 계산하는 방식은 다음과 같습니다.

1_ 연도를 4로 나눈 값이 0이라면 윤년일 수 있다.

2_ 그러나 해당 연수가 100으로 나누어지면 평년이다.

3_ 2에서처럼 평년이라고 해도 다시 400으로 나누어지는 연도는 윤년이다.

좀 알아보고 나니 복잡하군요. 좀 더 생각을 정리해볼 필요가 있습니다. 우선 기준이 되는 가장 중요한 데이터는 4, 100, 400이라는 숫자입니다. 우선 4와 100의 관계를 보도록 합니다.

4_ 4로 나누어지면서 100으로도 나누어지면 평년이라고 생각할 수 있다.

4번째 로직으로 프로그램을 만들면 다음과 같습니다.

예제 | 4로 나누어지면서 100으로 나눌 수 있으면 평년

```
public class LeapYearEx {

	public static void main(String[] args) {

		int year = 1900;

		if(year % 4 == 0 && year % 100 == 0){
			System.out.println(year +  "평년입니다.");
		}
	}
}
```

2.4.3 무조건 코딩을 만들지 말고 다른 방법도 생각해보세요.

4와 100의 관계를 생각해보면 우선 100으로 나눈 나머지가 0일 경우에는 4로 나눈 나머지 값도 마찬가지로 0인 것을 알 수 있습니다. 따라서 위의 코드를 조금 수정할 수 있습니다.

코드 |

```
		if(year % 100 == 0){//4로 나누어도 0이므로
			System.out.println(year + "평년입니다.");
		}else if(year % 4 == 0){
			System.out.println(year + "윤년입니다.");
		}
```

if ~ else를 적용할 때에는 위에서 좁은 그물, 아래에서는 넓은 그물을 이용해 주어야 합니다. 따라서 100과 4의 처리는 100으로 나누는 경우가 걸리는 데이터가 더욱 작기 때문에 위쪽에 있어야만 합니다. 그럼 100과 400의 관계를 보도록 합니다.

5_ 100으로 나누어지더라도 400으로 나눈 값이 0이면 무조건 윤년이다.

이 역시 그냥 코드를 만들려고 하면 if(year % 100 == 0 && year % 400 == 0)과 같이 생각할 수 있겠지만 데이터를 고려해 보면 400으로 나눈 값이 0일 경우에는 반드시 100으로 나눈 나머지도 0이기 때문에 조건을 단순화시킬 수 있습니다.

코드

```
if(year % 400 == 0){
        System.out.println(year + "윤년입니다.");
}else if(year % 100 == 0){
        System.out.println(year + "평년입니다.");
}
```

지금 위의 두 가지 상황을 고려해 보니 다음과 같은 코드가 되었습니다.

코드

```
//400 과 100
if(year % 400 == 0){
        System.out.println(year + "윤년입니다.");
}else if(year % 100 == 0){
        System.out.println(year + "평년입니다.");
}
//100과 4
if(year % 100 == 0){
        System.out.println(year + "평년입니다.");
}else if(year % 4 == 0){
        System.out.println(year + "윤년입니다.");
}
```

그런데 좁은 범위에서 넓은 범위로 내려오다 보니 중복된 경계선이 발생하는 것이 보입니다. 이제 이것을 통합해 보면 다음과 같습니다.

예제

```
public class LeapYearEx {

	public static void main(String[] args) {

		int year = 1900;

		if(year % 400 == 0){
			System.out.println(year + "윤년입니다.");
		}else if(year % 100 == 0){
			System.out.println(year + "평년입니다.");
		}else if(year % 4 == 0){
			System.out.println(year + "윤년입니다.");
		}
	}
}
```

우선 2000년의 경우에는 윤년이었고, 1900년은 평년이었습니다. 2000년은 맨 위의 로직에서 걸러지고, 1900년은 아래의 로직에서 걸러 집니다.

프로그램을 작성하다가 보면 비슷비슷한 데이터를 가지고 로직을 만들어야 하는 경우가 빈번합니다. 이때 그냥 무작정 프로그램을 작성하다 보면 운이 좋으면 그냥 잘 해결되겠지만, 한 번에 해결되지 않으면 점점 더 복잡한 코드를 작성하게 됩니다. 나중에는 돌아는 가지만 거의 알아보기도 어려운 코드를 작성하게 되는데 이것을 흔히 '스파게티 소스'라고 합니다. 어디가 시작인지 알 수 없는 코드라고 생각하시면 됩니다. 가끔은 여러분이 작성하는 코드가 점점 더 복잡해지는 경우에는 주로 코드를 잘 못 시작했다고 판단하는 것이 거의 정확합니다. 소스를 작성하는 중에 점점 더 복잡해지면 무언가 다시 시작해 보는 것이 좋겠다고 판단하는 것이 나중을 위해서 더 나은 선택이 됩니다.