> 01. 클래스 메소드(method)

함수란? 반복되거나 복잡한 코드를 미리 선언해 놓고 필요할때 호출해서 사용하는 코드 블록이다. 복잡한 수식을 기호화 한 것이다.

메소드와 함수의 차이는 기능적으로 같은데 객체 안에 존재하는 함수를 메소드라 부르고, 객체와 관계없이 독립적으로 사용되는 메소드를 함수라한다.

수학에서 함수 f(a,b)=2a+b=y 라고 할때 a를 1로 b를 2로 바꾼 f(1,2)=2*1+2의 결과 값이 4이여서 f(1,2)=4이 된다. f(1,1)=3, f(2,1)=5 이 될 것이다. f(2,1)=5 라 할때 f(2,1)+3=8이다. f(2,1)의 실행결과가 5이므로 5+3은 8이 된 것이다. f함수 내부의 내용을 몰라도 입력 값과 결과 값만 있다면 함수를 사용 할 수 있다.

1. f()=5+3이라 할때 f()+f()+1의 결과 값은 무엇인가?
2. 함수 f(a,b)=2a+b=y 라고 할때 f(0,3),f(1,4),f(2,2)의 결과 값이 무엇인지 생각해 보자.
3. f(a,b)=3a+b+1라는 함수가 있을때 f(0,3),f(1,4),f(2,2)의 결과 값이 무엇인지 생각해 보자.
4. f(a)=3a+1라는 함수가 있을때 f(0),f(1),f(2)의 결과 값이 무엇인지 생각해 보자.
5. f1(a)=2a+1,f2(a)=a+1 라는 함수가 있을 때 f1(3),f2(1),f1(f2(1))의 결과 값을 구하여라.
6. f1(a,b)=3a+b+1,f2(a)=3a+1,f3(a)=a+1 이라 할때 f1(f2(1),f3(2))의 결과 값을 구하여라.
7. f(a)=3a+1이라 할때 f(f(1))의 결과 값을 구하여라.

System.out.println()도 메소드이다. 내부에 어떻게 동작하는지 몰라도 실행 결과를 화면에 찍히게 하는 메소드이다. 대부분의 프로그램들은 함수에 입력값 매개변수를을 주면 컴퓨터가 메소드를 실행해서 메소드 내부가 어떻게 구성되어 있는지 몰라도 결과 값을 얻을 수 있도록 구현되어 있다.

프로그램에서 메소드의 용도는 반복되는 코드를 쉽게 재사용할 수 있고, 특정 값을 넣어서 원하는 결과를 얻어 낼 수 있다.

다음 문제를 해결할 수 있는 함수를 만들어 보자.

문제1. 1달러는 1,161원이다. 6달러가 우리나라 돈으로 얼마인지 구하는 함수를 만들어보자.

문제2. 구입할 컵의 개수와 가격을 통해 비용을 계산하는 함수를 만들어 보자.

문제3. 온도를 표시하는데에는 섭씨와 화씨 2가지가 있다. 섭씨로 화시를 구하는 함수를 만들어 보자.

섭씨(Celsius)는 물의 어는점과 끓는점 사이의 온도를 0℃와 100℃로 정의한 온도

체계입니다.

화씨(Fahrenheit)는 32℉를 물의 어는 점으로, 212℉를 물의 끓는 점으로 정의한 온도 체계 입니다.

섭씨를 화씨로 구하는 방법은 다음과 같다. °F = (°C * 1.8) + 32

답안. f(x)=x*1161 f(x,y)=x*y cupSum(cupCount,cupPrice)=cupCount*cupPrice

f(x)=(x*1.8)+32=y f(섭씨온도)=(섭씨온도*1.8)+32=y=화씨온도

메소드는 선언부와 호출 부로 구분 되어 있다. 선언부는 함수를 어떻게 동작 시킬 것인지 결정하는 부분이고, 호출부는 실제로 그 함수를 호출하는 부분이다. 수식에서 f(a,b)=2*a+b 가 선언부 이고 f(2,1)이 호출 부가 된다. 프로그램에서 함수를 사용 하려면 선언부을 먼저 구현하고 호출 부를 이용하여 함수를 실행 한다. 여기서 호출부는 함수를 사용하고 싶을 때 언제든지 여러 번 호출하여 사용 할 수 있다.

System.out.println()메소드는 전문 프로그래머가 미리 만들어 놓은 함수여서 선언하지 않고 바로 호출해서 사용 해도 되지만 본인이 함수를 만들 때에는 반드시 선언부를 만들고 사용하고 자 할때 호출해야 한다.

프로그램에서 호출부가 실행되고 나면 함수를 실행한 결과만 남는다. 즉, f(a,b)=2*a+b 에서 int a = f(2,1)는 f(2,1)이 호출되어 실행되면 f(2,1)는 없어지고 실행 결과인 5가 남는다. f(2,1)를 5로 바꿔 쓴것과 같은 결과가 된다는 이야기이다. int a = f(2,1); 와 int a=5; 는 f(2,1)이 실행된 결과가 5가 되기 때문에 둘다 a에 5가 들어간다. 함수를 실행하고 남은 결과 값을 함수의 리턴값이라고 한다.

객체지향 언어에서 객체안에 함수를 메소드라고 부른다. 자바 코드에서 메소드 종류는 크게 인스턴스 메소드와 클래스 메소드 2가지가 있다.

static이 붙은 메소드를 클래스 메소드라고 하고 static이 없는 메소드를 인스턴스 메소드라고 한다. 클래스 메소드는 앞에 static이 붙어 있어서 static 메소드, 전역에서 접근할 수있어서 전역 메소드라 하는데 정상적인 명칭은 클래스 메소드이다.

다음은 클래스 메소드 관련 설명이다.

클래스 메소드는 선언부와 호출부로 구분 된다. 선언부를 이용해서 선언한 다음 호출부를 이용해서 호출해여 동작한다.

다음은 클래스 메소드 선언부 구현하는 방법과 실제 구현한 코드 이다.

```
public static 리턴될자료형 메소드이름(자료형 매개변수1, 자료형 매개변수2 ...) {
    //메소드 내용
    return 리턴될데이터;
}
```

```
public static int sum(int a,int b) {
    int sum=a+b;
    return sum;
}
```

메소드 이름은 여러 개의 메소드들 중 특정 메소드를 구분할 때 사용한다. f(a,b)=a+b=y와 왼쪽의 코드를 비교한다면, 함수 이름 f는 sum에 해당하며 여러 메소드를 구분하는데 사용 한다. 간단한 이름 f 대신에 좀 더 의미있는 이름 sum을 사용 하였다. 함수 이름도 변수명 처럼 마음대로 이름을 붙일 수 있는데 최대한 함수가 어떤 일을 하는지 알 수 있도록 이름을 만드는 것이 좋다.

매개변수는 메소드를 실행 시킬때 필요한 데이터를 기술 하는데 사용한다.

f(a,b)와 sum(int a,int b)에서 a,b가 인자라 하는데 보통 매개변수라 부른다. 자바에서는 선언시 매개 변수 앞에 자료형을 기술해야 하고, 필요한 데이터 개수 만큼 매개변수를 선언하여 사용하면 된다.

리턴값은 해당 메소드가 실행되고 나면 생성된 결과 데이터를 의미한다. f(5,2)의 실행결과가 4가 된다면 리턴 값은 4이다. 실행결과가 10이라면 리턴값은 10이다.

리턴될자료형은 f(a,b)=a+b=y에서 y와 같이 함수를 실행하고 남은 데이터의 자료형이다.

public은 접근 제한자이다. 모든 지역에서 접근 가능하다는 이야기이다. static은 클래스 메소드에 붙여줘야 한다. 접근제한자와 static은 일단 그냥 붙여주면 된다고 생각하자.

다음 예제들을 확인해 보자.

0. 실행 결과를 확인해 보자.

```
public class MethodExample {
    public static void myPrint() {// 메소드 선언
        System.out.println("a");
        System.out.println("b");
        System.out.println("c");
        return;  // 리턴값이 없다. 생략 가능하다.
    }
    public static void main(String[] args) {
        MethodExample.myPrint();// 메서드 호출
        MethodExample.myPrint();
        MethodExample.myPrint();
        MethodExample.myPrint();
    }
}
```

1.

```
public class MethodExample {
```

```
    public static int add(int a, int b) {// 메소드 선언
        return a + b;  //리턴값이 a+b한 값이다
    }
    public static void main(String[] args) {
        int result = MethodExample.add(5, 3); // 메서드 호출
        System.out.println("Result: " + result);
        // 리턴값이 없으면 return를 생략 가능하다.
    }
}
2.
//OverloadingExample.java
public class OverloadingExample {
    public static int add(int a, int b) {
        return a + b; // 메소드 이름이 같아도 매개변수가 다르면 만들 수 있다.
    }
    public static double add(double a, double b) {
        return a + b;
    }
}
//OverloadingMain.java
public class OverloadingMain {
    public static void main(String[] args) {
        int result1 = OverloadingExample.add(5, 3);
        double result2 = OverloadingExample.add(2.5, 1.5);
        System.out.println("Result 1: " + result1);
        System.out.println("Result 2: " + result2);
    }
}
3.
class SquareExample {
    public static int square(int num) {
        return num * num;
    }
}
public class SquareMain {
    public static void main(String[] args) {
        int result = SquareExample.square(4);
        System.out.println("Square: " + result);
    }
}
4.
class StringLengthExample {
    public static int getStringLength(String text) {
        return text.length();
    }
}
public class StringLengthMain {
```

```
    public static void main(String[] args) {
        String message = "Hello, Java!";
        int length = StringLengthExample.getStringLength(message);
        System.out.println("Length: " + length);
    }
}
5.
class ArraySumExample {
    public static int getArraySum(int[] arr) {
        int sum = 0;
        for (int num : arr) {
            sum += num;
        }
        return sum;
    }
}
public class ArraySumMain {
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3, 4, 5};
        int sum = ArraySumExample.getArraySum(numbers);
        System.out.println("Sum: " + sum);
    }
}
6.
class MaxValueExample {
    public static int getMaxValue(int[] arr) {
        int max = arr[0];
        for (int num : arr) {
            if (num > max) {
                max = num;
            }
        }
        return max;
    }
}
public class MaxValueMain {
    public static void main(String[] args) {
        int[] numbers = {14, 6, 23, 8, 31};
        int max = MaxValueExample.getMaxValue(numbers);
        System.out.println("Max value: " + max);
    }
}
7.
class EvenOddExample {
    public static boolean isEven(int num) {
        return num % 2 == 0;
    }
```

```
}
public class EvenOddMain {
    public static void main(String[] args) {
        int number = 7;
        if (EvenOddExample.isEven(number)) {
            System.out.println(number + " is even.");
        } else {
            System.out.println(number + " is odd.");
        }
    }
}
8.
class PrimeNumberExample {
    public static boolean isPrime(int num) {
        if (num <= 1) {
            return false;
        }
        for (int i = 2; i <= Math.sqrt(num); i++) {
            if (num % i == 0) {
                return false;
            }
        }
        return true;
    }
}
public class PrimeNumberMain {
    public static void main(String[] args) {
        int number = 17;
        if (PrimeNumberExample.isPrime(number)) {
            System.out.println(number + " is a prime number.");
        } else {
            System.out.println(number + " is not a prime number.");
        }
    }
}
```

```
2  public class JavaStart {
3      public static int sum(int a,int b) {
4          int sum=0;
5          sum=a+b;
6          return sum;
7          //sum=sum+1;
8      }
9      public static void main(String[] args)
10         int a=sum(1,2);
11         int b=a+sum(2,2);
12         System.out.println(a+b);
13         //min mul div   함수를 만들어서 사용해보자.
14     }
15 }
```

메소드명 앞에 있는 리턴될 자료형은 메소드가 실행된 후 생성된(리턴된) 데이터의 자료형을 기술해 주어야 한다. 옆의 예제에서는 리턴값은 sum메소드가 종료되면 리턴되는 변수명 sum의 자료형이다.

리턴될 자료형이 없으면 리턴될 자료형은 void 이고 리턴 값은 return;처럼 기술 한다. return값이 있을 때는 생략이 불가능 하지만 없으면 return;를 생략 할 수 있다. int sum=a+b;가 함수가 호출 되었을때 실행되는 코드이다.

상위 코드를 실행해 보고 이후 나오는 설명을 읽어 보자.

3~8라인은 선언부에 해당한다. 10, 11라인의 메소드 호출부에 해당 한다. 호출부는 메소드 이름에 매개변수를 넣어 소괄호로 묶어주면 해당 함수를 호출해 선언부의 매개변수에 넣어 준다.

public static : 이 부분은 어디서나 접근할 수 있는 전역 메소드를 의미한다.

public은 접근제한자로 어디서나 접근할 수 있음을 나타내고 static은 클래스 메소드임을 나타낸다.

메소드 이름 : 여러 메소드 중에 본인이 원하는 메소드를 식별 하기 위해서 사용한다. 함수 보다 메소드라고 부르는 것이 객체에서는 올바른 표현이다.객체 안에 기술된 함수를 메소드라 한다.

기본적으로 메소드 이름은 모두 달라야 한다. 하지만, 메소드 이름이 같아도 매개 변수 개수나 자료형이 다르면 메소드를 구분 할 수 있기 때문에 같은 메소드 이름을 허용 한다. 이를 전문 용어로 오버로드(overload)라 한다. 오버로드는 매개변수의 개수가 다르거나 종류가 달라 식별이 가능할 때 메소드 이름 중복을 허용 한다는 의미이다. 함수의 이름을 짓는 방법은 변수명을 짓는 것과 동일하다. 여기서는 함수 이름을 sum 이라 지었다.

매개변수 : 선언할 때는 매개변수라 하고 호출할때는 인자라 하는데 보통 둘다 매개 변수라 한다. 함수가 호출될 때 필요한 데이터가 함께 넘어와야 하는데 이때 매개 변수를 사용한다. 매개 변수가 없어도 되고, 하나여도 되고, 두 개 여도 되고, 여러 개 여도 된다.

사용방법은 선언시 메소드명 옆 소괄호 안에 자료형과 식별자로 필요한 만큼 ,표로 연결하여 선언하면 된다. 상위 예제에서 sum(int a,int b)와 같이 기술 하였다.

호출시에는 메소드명 옆 소괄호에 선언할 대의 매개변수 자료형과 개수에 맞춰서 호출 하면 된다. sum(10,20);

메소드내용 : 실제 함수가 호출 되었을때 실행될 코드 부분이다. 상의 코드에서는 중괄호{} 안에 있는 부분이 메소드의 내용이다.

리턴될 자료형 : 리턴이란 함수가 실행되고 생성되는 데이터을 의미한다. 리턴될 자료형 위치에는 이때 생성된 데이터의 자료형을 기술한다.

메소드 이름 앞에 리턴값의 자료형을 기술해야 한다. 메소드가 실행되고 생성된 숫자가 3이면 3이 정수 이므로 int를 써 주면 된다. 메소드가 실행하고 남은 데이터가 문자열이면 String을 써 주면 된다. 메소드를 실행하고 남은 데이터가 실수 3.0이면 double을 써 주면 된다. 여기에서는 정수이므로 int를 사용 하였다. 혹시 리턴 될 값이 없다면 함수 앞에 리턴된 값에는 void라고 입력하면 된다.

메소드 종료전에 실제 리턴할 값을 다음과 같이 기술해야 한다. return 리턴할값; 리턴할 값으로 상수나 계산 결과 변수 등이 올수 있다. return 값이 없다면 return;이라 기술 하거나 아에 return; 구문을 생략해도 된다.

리턴될 자료형에는 int, double, String 등과 같은 예전에 배운 자바에서 존재하는 모든 자료형이 올 수 있다. 기본 자료형 뿐만 아니라 참조 자료형도 올 수 있다.

여기에서는 변수 sum에 들어 있는 정수가 리턴된다.

메소드 안에서 리턴을 만나면(return) 리턴값을 가지고 메소드가 종료되므로 리턴 다음에 프로그램 코드가 있으면 접근 할 수 없는 코드가 되므로 작성하면 안된다.

메소드 사용 방법을 다시 한번 정리해 보자면 첫 번째로 메소드를 정의 하고 두 번째로 필요할때 마다 메소드를 호출 해야 한다.

정의한 함수를 실행 시키고 싶으면 sum(1,2); 처럼 함수 이름에 매개변수 값을 1:1 연결(매핑)시켜 소괄호 안에 넣어 호출부를 기술하면 된다.

다음 문제를 메소드로 만들어 보자.

1. 1달러는 1,161원이다. 6달러가 우리나라 돈으로 얼마인지 구하는 함수를 만들어보자.

2. 구입할 컵의 개수와 가격을 통해 비용을 계산하는 함수를 만들어 보자.

3. 온도를 표시하는데에는 섭씨와 화씨 2가지가 있다.

```
package com.human.ex2;
public class MyFunction {
	//달러랑 환율을 입력받아 원을 리턴해주는 함수
	public static double exchange(double dollar,double exchangeRate) {
		double won=dollar*exchangeRate;
		return won;//리턴값 함수가 종료되면 남는 값
	}
	//나머지 메소드 제작 위치
```

```
	public static int expense(int cupCount, int cupPrice) {// 함수 선언
		int expense = cupCount * cupPrice;
		return expense;
	}
	public static double fahrenheit(double celsius) {
		double fahrenheit = (celsius) * 4 / 9 + 32;
		return fahrenheit;// 화씨
	}
	public static void main(String[] args) {
		//달러를 원으로 출력 1161
		double dollar=6.2;
		double won=0;
		double exchangeRate=1161;
		won=dollar*exchangeRate;
		System.out.println(dollar+" 달러를 "+exchangeRate
				+" 환율로 변환하면 "+won+"원이 된다.");
		//exchange 메소드를 사용한방법
		dollar=6.2;
		won=0;
		exchangeRate=1161;
		won=MyFunction.exchange(dollar, exchangeRate);//f(6.2,1161)
		System.out.println(dollar+" 달러를 "+exchangeRate
	+" 환율로 변환하면 "+won+"원이 된다.");

		// 구입할 컵의 개수와 가격을 통해 비용을 계산하는 함수를 만들어 보자.
		int cupCount = 10;
		int cupPrice = 2500;
		int expense = 0;
		expense = cupCount * cupPrice;
		System.out.println("발생하는 비용은 총 " + expense + "원 입니다.");
		cupCount = 10;
		cupPrice = 2500;
		expense = 0;
		expense = MyFunction.expense(cupCount, cupPrice);//함수
		System.out.println("발생하는 비용은 총 " + expense + "원 입니다.");

		System.out.print("섭씨 온도: ");
		double celsius =30;
		double fahrenheit = MyFunction.fahrenheit(celsius);
		System.out.print("섭씨 " + celsius + "도는 화씨" +
 fahrenheit(celsius) + "°F 입니다.");
	}
 }
```

4. 4칙 연산 가능한 메소드를 min mul div 메소드를 사용하여 프로그램으로 만들어 보자. 새로운 메소드는 상위 코드의 8-9라인 사이에 메소드를 추가 해야 한다.

5. System.out.println(sum(sum(1,1),sum(2,1));의 결과 값은 무엇인가? 메소드는

실행된 리턴값만 남는다고 생각 하면 된다. 결과를 생각한 후 쳐서 확인해 보자.

6. 매개변수와 리턴 값이 없는 함수를 이용해서 “안녕하세요” 라는 내용을 출력하는 함수 hello를 만들고 프로그램 시작을 의미 하는 메인 함수에서 헬로 함수를 여러 번 호출하여 “안녕하세요”를 출력하는 프로그램을 만들어 사용해 보자.

```java
package com.human.ex;
public class JavaStart3 {
    public static int sum(int a,int b) {
        int sum=0;
        sum=a+b;
        return sum;
        //sum=sum+1; 실행될수 없는 부분이다.
    }
    public static int div(int a,int b) {
        return a/b;
    }
    public static int min(int a,int b) {
        return a-b;
    }
    public static int mul(int a,int b) {
        return a*b;
    }
    public static void main(String[] args) {
        int a=sum(1,2);
        int b=a+sum(2,2);
        System.out.println(a+b);
        System.out.println(mul(a,b));
    }
}
```

```java
public static void hello() {
    System.out.println("안녕하세요");
}
public static void main(String[] args) {
    hello();
    hello();
    hello();
    for(int i=0;i<3;i++) {
        hello();
    }
}
```

7. 입력받은 숫자가 7의 배수인지 아닌지 true, false값을 리턴하는 메소드를 구현해 보자.

8. 문자열과 숫자를 입력 받아 해당문자열을 숫자만큼 반복 출력하는 메소드를 만들어 보자.

```java
public static int sumAll(int a,int b) {
    int sum=0;
    if(a>b) {
        int temp; temp=a; a=b; b=temp;//교환작업
    }//b가 항상 큰수가됨
    for(int i=a;i<=b;i++) {
        sum=sum+i;
    }
    return sum;
}
public static void main(String[] args) {
    int a=10,b=12;
    System.out.println(sumAll(a,b));
    System.out.println(sumAll(5,12));
}
```

9. 두 수의 사이 수의 합을 구하는 함수를 만들어 사용해 보자.
ex)다음과 같은 입력에도 5,2 2,5 결과가 14가 나오게 만들어 보자.

다음 처럼 메소드 안에서 다른 메소드를 호출할 수 있다.

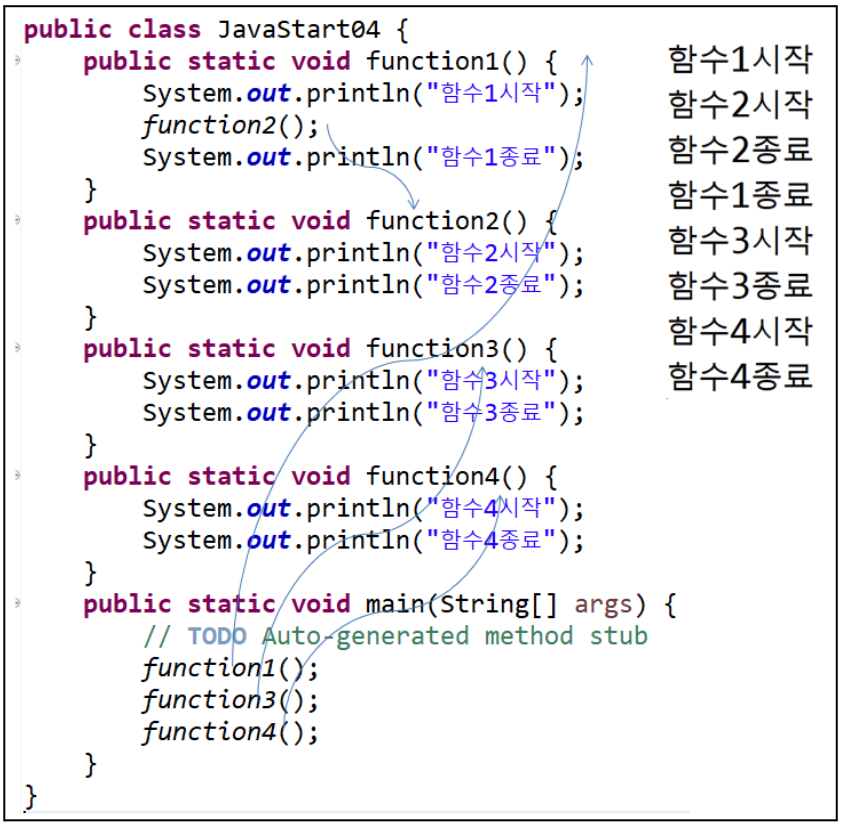
메소드가 호출되면 메소드 선언부로 이동해 위에서 아래로 순서대로 실행을 이어 나가다 메소드가 종료되면 이전에 호출한 코드 부분으로 복귀 한다.

```
public class MethodExampleOutput {
    public static void main(String[] args) {
        System.out.println(“main  start");
        methodA();
        System.out.println("main end");
    }
    public static void methodA() {
        System.out.println("Method A start");
        methodB();
        System.out.println("Method A end");
    }
    public static void methodB() {
        System.out.println("Method B");
    }
}
```

여기서 순서는 다음과 같다
main 메소드가 호출되면, methodA가 실행됩니다.
methodA 내에서 "Method A start"를 출력한 후 methodB를 호출합니다.
methodB 내에서 "Method B"를 출력합니다.
다시 methodA로 돌아와 "Method A end"를 출력합니다.
main 메소드로 돌아가면서 프로그램이 종료됩니다.

다음은 출력 결과이다.

```
Method A start
Method B
Method A end
```



왼쪽 이미지의 코드를 확인하여 메인에서 순서대로 따라 가면서 출력 결과를 확인해 보자.

메소드에서 본인의 메소드를 호출할 수 있다.

```
public void f1(){

	f1();

}
```

재귀 호출이라고 한다. 재귀 호출은 무한 루프를 돌수 있다. 무한 루프란 프로그램이 종료되지 않고 계속 반복되는 것을 의미한다. 나중에 실력이 쌓이면 웹에서 검색해서 자세히 공부해 보자. 지금은 간단히 다음 코드를 확인해 보고 넘어가자.

```
public class selfFunction {
	public static void f1(int x) {
		System.out.println(x);
		if(x<5) {//종료조건
			f1(x+1);
		}
	}
	public static void main(String[] args) {
			f1(1);
	}
}
```

메소드, if문, for문등의 {}중괄호 안에 기술된 변수는 대부분 {} 내부에서만 사용 할 수

있어서 지역변수라 부른다.

메소드의 매개변수도 해당 함수의 지역변수이다. 지역변수는 해당 중괄호 지역에서만 사용할 수 있고 중괄호가 닫히면 메모리에서 삭제 된다.

메모리는 스택과 힙이 있는데 지역변수는 메소드로 구분되어 호출 될 때마다 메모리 스택에 잡혔다가 메소드가 종료되면 스택 메모리에서 사라진다. } 중괄호가 닫히면 스택 메모리에서 완전히 사라진다는 이야기이다.

연습문제

1.메소드란? 메소드를 크게 두부분으로 나누면 어떻게 구분되는가?

2.메소드를 사용하려면 두부분을 어떻게 사용해야 하는가?

3.메소드 종류 2가지는 무엇이고 어떻게 구별하는가?

4.전역과 지역의 의미는?

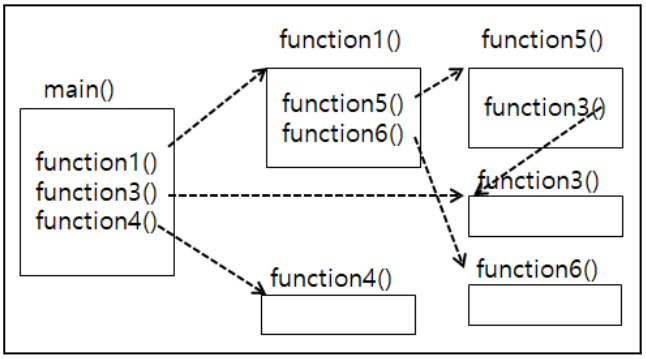
5.메소드이름의 용도는? 매개변수의 용도는? return의 용도는?

6.함수 선언시 리턴될자료형의 용도는?

7.리턴될자료형의 자리에 올수 있는 키워드들 기술하고 어느때 오는지 이야기 해보자.

8. 리턴될 자료형이 없으면 어떻게 하여야 하는가?

9.오버로드란? 메소드 만드는 방법을 설명해 보자.



11.함수와 메소드의 차이는?재귀 호출이란? 무한 루프란?

12.지역변수의 사용 범위에 대하여 설명하시오.

13. 왼쪽 이미지 처럼 동작하는 함수들을 만들어 보자.

클래스 필드는 모든 지역에서 접근가능하기 때문에 메소드 안에서도 접근 할 수 있다.

```
public class StaticExample {
    // static 변수
    static int staticVariable = 10;
    // static 메소드
    public static void main(String[] args) {
        System.out.println("M-StaticVariable:"+StaticExample.staticVariable);
        // static 메소드에서 static 변수 사용
        StaticExample.staticMethod();
        System.out.println("M-StaticVariable:"+StaticExample.staticVariable);
    }
    // 다른 static 메소드
    public static void staticMethod() {
        System.out.println("S-StaticVariable:"+StaticExample.staticVariable);
        // static 메소드에서 static 변수 변경
        StaticExample.staticVariable += 5;
        System.out.println("S-StaticVariable:"+StaticExample.staticVariable);
    }
}
```

출력결과는 다음과 같다.

```
M-StaticVariable:10

S-StaticVariable:10

S-StaticVariable:15

M-StaticVariable:15
```

```
1  public class TestClass4 {
2      public static int total = 0; // 클래스변수
3      public static int num1 = 0; // 클래스변수
4      public static int num2 = 0; // 클래스변수
5
6      public static void add() {
7          //이부분에 코드를 작성하시오
8      }
9      public static void main(String[] args) {
10         num1 =10;
11         TestClass4.num2 =30;
12         add();
13         System.out.println(total);
14     }
15 }
```

왼쪽 예제는 전역변수 num1과 num2의 값을 더한 결과를 total에 저장해서 화면에 출력하는 프로그램을 7번 라인에 완성해 보자.

4칙연산이 가능하게 다른 메소드들을 추가해 보자.

풀이

total=num1+num2; 를 추가하면 된다.

실습문제

다음 문제를 풀때 main메소드를 포함하지 않는 MyFunction클래스의 클래스 메소드로 만들어 풀어보자.

1. triangle1(), triangle2(), triangle3() 메소드는 각각의 삼각형을 그릴 수 있는 메소드이다. 사용자 입력을 받아 사용자가 원하는 삼각형 별찍기 프로그램을 만들어 보자.

2. 명함을 만드는 메소드를 만들고 메소드를 이용해 이름, 이메일, 전화번호를 입력 받아 명함을 출력하는 프로그램을 만들어 보자.

3. 배열을 매개변수로 받아 배열 내용의 합을 리턴해 주는 메소드를 만들어 보자.

4. 매개 변수가 하나인 메소드 6개(intFunc, doubleFunc, stringFunc, catFunc, intArrFunc, catArrFunc)를 각각 만들어 보자.

메소드 별 각각의 매개 변수와 리턴값은 다음과 같다.

인트(int), 더블(double), 스트링 (String), 이름 나이를 저장하는 고양이 객체(Cat), int형 배열(int[]), 고양이 객체 배열(Cat[])

해당 메소드들의 용도는 매개변수로 받은 데이터의 내용을 변경해서 배열로 리턴해 준다.

이후 메인 메소드에서 리턴 된 값을 화면에 출력해 보자.

5. 메뉴에 어떤 도형의 넓이를 구할 것인지 사용자에게 물어본 다음 원의 넓이 구하는 메소드 사각형의 넓이를 구하는 메소드 삼각형의 넓이를 구하는 메소드를 이용해서 구현해 보자.

6. 매개변수에 숫자 하나를 받아 들여서 구구단중 해당 숫자의 단을 모두 출력하는 메소드를 작성해서 사용자 입력을 받아 사용자가 입력한 단을 모두 출력하는 프로그램을 완성해 보자.

int, double, String, Cat 객체, int[] 배열, Cat[] 배열을 매개변수로 받아 처리하는 Java 프로그램입니다. 이 프로그램은 각 매개변수를 출력하거나 처리하는 방식으로 동작합니다.

고양이 객체 (Cat 클래스) 정의:

먼저, Cat 클래스를 정의합니다. 이 클래스는 고양이의 이름과 나이를 저장합니다.

```
package com.the.ex;

public class Cat {

    private String name;

    private int age;
```

```
    // 생성자
    public Cat(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Getter 메서드
    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    // 고양이 정보를 문자열로 반환
    @Override
    public String toString() {
        return "Cat{name='" + name + "', age=" + age + "}";
    }
```

메인 프로그램 구현:

이제 `int`, `double`, `String`, `Cat`, `int[]`, `Cat[]`를 매개변수로 받는 메서드를 구현합니다.

```
package com.the.ex;
```

```
public class MultiParameterExample {
    public static void main(String[] args) {
        // 테스트 데이터 준비
        int intValue = 10;
        double doubleValue = 3.14;
        String stringValue = "Hello, Java!";
        Cat cat = new Cat("나비", 3);
        int[] intArray = {1, 2, 3, 4, 5};
        Cat[] catArray = {
            new Cat("치즈", 2),
            new Cat("모모", 5),
            new Cat("루이", 1)
        };

        // 메서드 호출
        processParameters(intValue, doubleValue, stringValue, cat, intArray, catArray);
    }

    // 매개변수를 처리하는 메서드
    public static void processParameters(
        int intValue,
        double doubleValue,
        String stringValue,
        Cat cat,
        int[] intArray,
        Cat[] catArray
```

```
    ) {
        // int 출력
        System.out.println("int 값: " + intValue);

        // double 출력
        System.out.println("double 값: " + doubleValue);

        // String 출력
        System.out.println("String 값: " + stringValue);

        // Cat 객체 출력
        System.out.println("Cat 객체: " + cat);

        // int 배열 출력
        System.out.println("int 배열:");
        for (int i : intArray) {
            System.out.print(i + " ");
        }
        System.out.println();
        // Cat 배열 출력
        System.out.println("Cat 배열:");
        for (Cat c : catArray) {
            System.out.println(c);
        }
    }
}
```

프로그램 설명:

1. **`Cat` 클래스**:
    - 고양이의 이름(name)과 나이(age)를 저장합니다.
    - `toString()` 메서드를 오버라이드하여 고양이 정보를 문자열로 반환합니다.
2. **`processParameters` 메서드**:
    - `int`, `double`, `String`, `Cat`, `int[]`, `Cat[]`를 매개변수로 받습니다.
    - 각 매개변수의 값을 출력하거나 처리합니다.
3. **`main` 메서드**:
    - 테스트 데이터를 준비하고 `processParameters` 메서드를 호출합니다.

실행 결과:

int 값: 10

double 값: 3.14

String 값: Hello, Java!

Cat 객체: Cat{name='나비', age=3}

int 배열:

1 2 3 4 5

Cat 배열:

Cat{name='치즈', age=2}

Cat{name='모모', age=5}

Cat{name='루이', age=1}

추가 설명:

- **`int[]` 배열**: 정수 배열을 순회하며 각 값을 출력합니다.
- **`Cat[]` 배열**: `Cat` 객체 배열을 순회하며 각 고양이의 정보를 출력합니다.
- **확장성**: 필요에 따라 `processParameters` 메서드 내에서 각 매개변수를 추가로 처리할 수 있습니다. 예를 들어, `int[]` 배열의 합계를 계산하거나 `Cat[]` 배열에서 특정 조건을 만족하는 고양이를 찾는 등의 작업을 추가할 수 있습니다.

> 07. 이중 for문 사용하기

반복문안에 반복문을 넣을 수 있다. for문 안에 for문을 중복해서 사용 할 수있다.

0123456789012345678901234567890123456789012345678901234567890123456789…

0123456789를 출력하는 이작업을 10번 반복하여 출력하고 싶으면 다음과 같이 기술 하면 될 것이다. for문 을 이용해서 0123456789를 10번 출력하는 코드로 다음과 같이 기술하면 될 것이다.

```
for(int i=0;i<10;i++) {
    System.out.print("0123456789");
}
```

잘 생각해 보면 for문 사이에 있는 프린트 문을 for문을 이용해서 0부터 9까지 출력해서 화면에 0123456789가 출력 되도록 할 수 있다. print문으로 0123456789를 출력하는 대신에 다음과 같은 for문을 이용하여 같은 결과를 만들 수 있을 것이다.

```
for(int j=0;j<10;j++) {
    System.out.print(j);  // 출력결과 0123456789
}
```

이 2가지 결과를 합친 결과는 다음과 같다.

```
for(int i=0;i<10;i++) {
    for(int j=0;j<10;j++){
        System.out.print(j);
    }
}
```

이중for 문을 사용할 때 조심해야 할 점은 초기식으로 선언되는 i,j 같은 변수들의 이름이 겹치거나 잘못 사용하면 문제가 발생 한다는 점이다.

바깥쪽 for문의 반복 코드 부분이 새로 시작 하게 되면 내부에 선언한 변수와 for문의 초기문도 새로 실행되어 변수의 값들이 초기화 되어 이전에 가지고 있던 값이 사라진다. 코드에서 j 값은 i가 0부터 변환식에 도착해서 하나씩 증가 할때 마다 0으로 초기화 되어 다시 하나씩 증가한다.

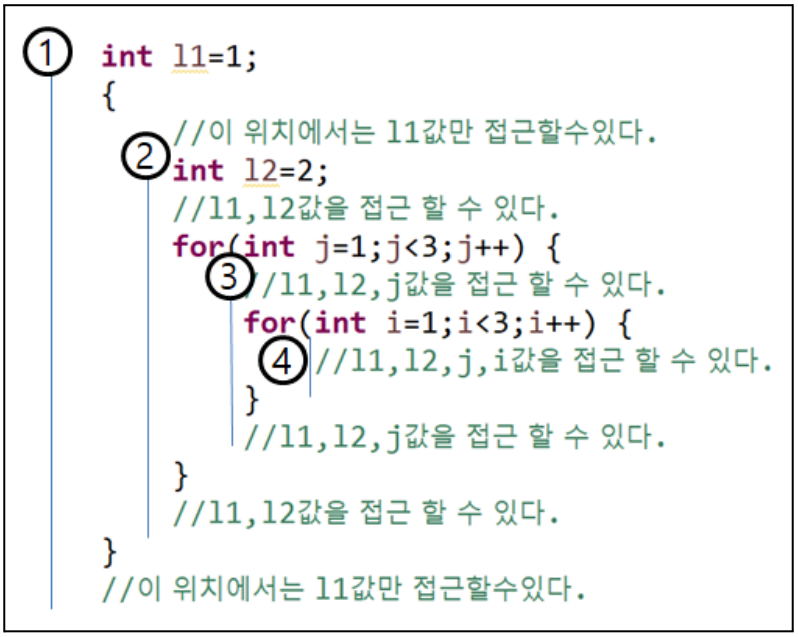
변수는 지역 변수와 전역변수로 구성되어 있는데 지역변수는 특정 지역에서만 사용할 수 있는 변수이고 전역변수는 모든 지역에서 사용할 수 있는 변수이다. 지금 까지 사용한 변수는 모두 지역 변수이고 전역 변수 앞에는 static이 붙는다. 나중에 좀더 자세히 살펴볼 예정이고 일단 지역변수의 특성만 좀 기억해 보자.

지역변수는 기본적으로 중괄호 블럭안에 선언한 변수로 선언되면 해당 중괄호 블럭이나 해당 중괄호 블럭 안쪽의 중괄호 블럭들에서만 사용할 수 있다.

지역변수가 선언된 범위에서 다시 같은 이름의 지역 변수를 선언하면 동일한 지역 변수를 식별 할 수 없어서 문제가 발생 하니 주의하자.

메소드의 매개변수는 해당 메소드 중괄호 블록 안에서 지역변수로 사용된다.

for문의 초기화 식은 해당 for문 중괄호 블록 안에서 지역변수로 사용된다.



왼쪽 이미지의 1번 부분은 메인 메소드 선언부에 중괄호안 이여서 l1은 main 메소드에 선언된 지역변수 이다.

이 중괄호 부분 에서 선언된 l1는 중괄호 내부인 왼쪽 이미지에서 표시된 1, 2, 3, 4 중괄호 안쪽 부분에서 사용할 수 있다.

2번 부분에 선언된 l2의 경우 1번 부분은 2번 중괄호 부분의 바깥쪽 중괄호 이므로 1번을 제외한 2,3,4 중괄호 부분에서 사용할 수 있다.

3번에 선언된 변수 j같은 경우 3번 중괄호 블록에서 사용하려고 선언한 변수이므로 3,4에서만 사용할 수 있다.

4번 블록에서 선언된 i값은 4번 블록에서만 사용할 수있다.

상위 모든 주석 부분에 l1,l2,i,j 변수을 찍을수 있는지 코드를 통해서 직접 확인해보자.

다음과 같이 아래로 출력하는 방법에 대해서 생각해 보자.

```
0123456789      //012345678901234567890123456… 옆으로 출력되는 것과
0123456789      //아래로 출력되는 것의 차이점을 생각해보자.
0123456789      //옆으로 출력되는 것은 엔터가 없는것이고
0123456789      //아래로 출력되는 것은 엔터가 있는것이여서
0123456789      // '0123456789'가 반복되는 것이아니라.
0123456789      // '0123456789엔터'가 반복되는 것이다.
```

```
for(int i=0;i<10;i++) {
    //System.out.print(i+"\n");
    System.out.print(i);

}
System.out.print("\n");
```

하나의 for문으로 "0123456789엔터"를 출력하는 프로그램을 구현하려면 왼쪽과 같이 하면 된다. 간혹 주석 부분처럼 잘못 구현하는 경우가 있는데 잘 생각해 보면 주석 한 부분은 다음과 같이 잘못 출력 되는 걸 알 수 있다.

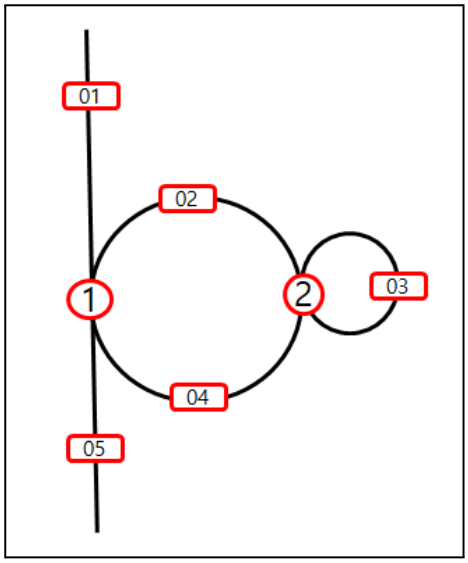
“0엔터1엔터2엔터3엔터4엔터5엔터6엔터7엔터8엔터9엔터엔터”

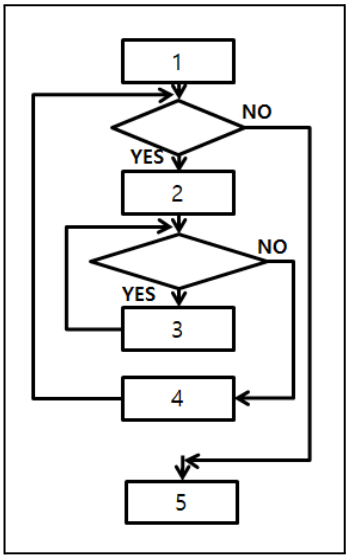
올바르게 구현하려면 상위 코드처럼를 이용해서 “0123456789엔터”가 여섯번 반복되도록 코드를 구현하면 우리가 원하는 출력 결과를 얻을 수 있다.

```
for(int j=0;j<10;j++) {
    for(int i=0;i<10;i++) {
        //System.out.print(i+"\n");
        System.out.print(i);

    }
    System.out.print("\n");
}
```

“0123456789엔터”이 아래로 열개 출력되도록 구현해 보자. 다음과 같이 구현 가능할 것이다.






```
1

while( ){
        2
        while( ){
                3
        }
        4
}

5
```

상위 맨왼쪽 이미지를 보고 순서도와 의사 코드를 작성 하시오.

상위 순서도에서 출력 되는 모든 경우를 기술해 보자. 반복되는 부분을 소괄호로 묶었다.

1번 부부의 반복문이 거짓인 경우 출력 결과는 01, 05가 된다.

1번 부분이 반복문이 참이고 2 번 부분이 반복무늬 거짓이면 출력 결과는 01, 04, 02, 05가된다.

1번 부분이 반복문이 참이고 2 번 부분이 반복무늬 거짓인 상태에서 1번 부분이 여러번 반복되면 출력 결과는 01, (04, 02), 04, 02, 04, 02,..., 05가 된다.

1번 부분이 반복문이 참이고 2 번 부분이 참 일때 출력 결과는 01, 04, 03, 02, 05가 된다.

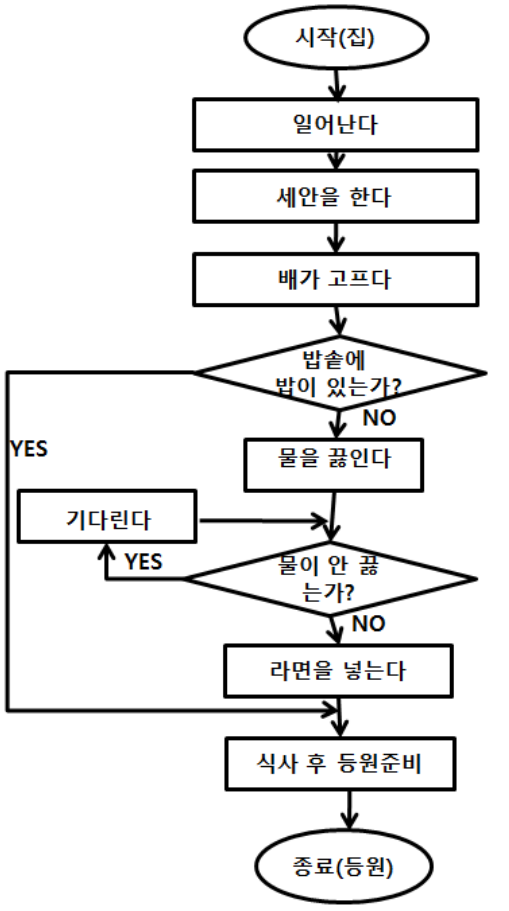
1번 부분이 반복문이 참이고 2 번 부분이 참 일때 반복문 1번은 1번 반복문 2번은 여러번 반복된다면 출력 결과는 01, 04, (03), 03, 03, ..., 02, 05가 된다.

1번 부분이 반복문이 참이고 2 번 부분이 참인 일때 반복문 1번은 여러번 반복문 2번은 한번 반복된다면 출력 결과는 01, (04, 03, 02),04, 03, 02,04, 03, 02, ..., 05,가

된다.

1번 부분이 반복문이 참이고 2 번 부분이 참인 일때 반복문 1번은 여러번 반복문 2번도 여러번 반복된다면 출력 결과는 01, (04, (03), 03, 03,... 02),04, 03, 02,04, 03, 02, ..., 05가 된다.

문제1) 아래의 순서도 의사코드에서 뭐가 잘못되었는지 확인해서 올바르게 고쳐 보자.



의사코드

시작(집)

일어난다

세안을 한다

배가 고프다

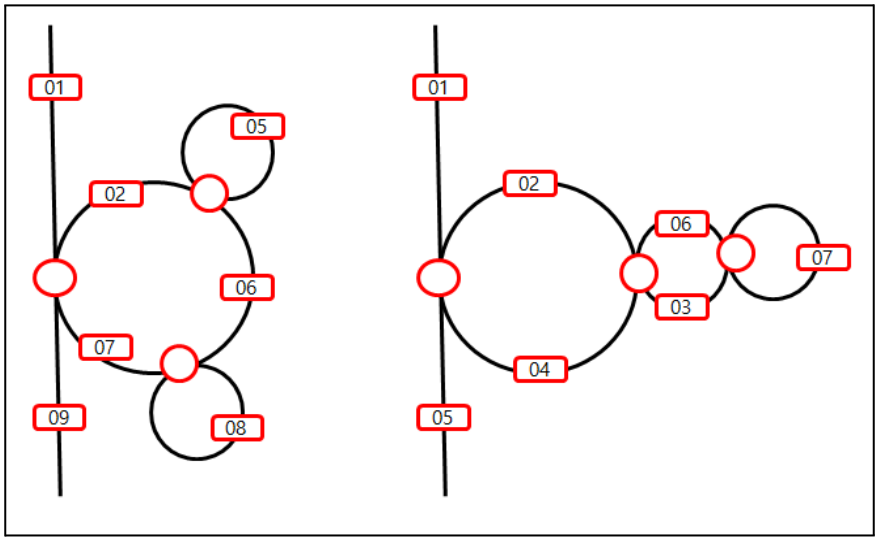
if(밥솥에 밥이 있는가?) {
식사 후 등원준비
}else{
물을 끓인다
}

While(물이 안 끓는가?){
기다린다
}

라면을 넣는다

식사 후 등원준비

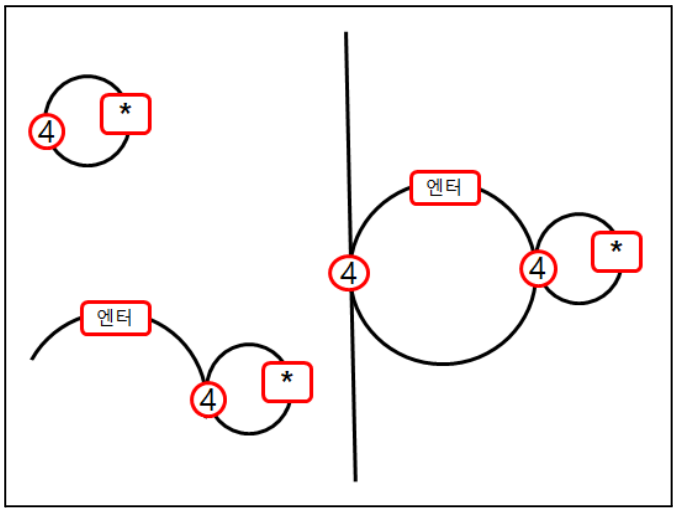
종료(등원)



문제 2)왼쪽 기찻길 이미지 두개를 순서도와 의사 코드로 구현하여 보자.

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

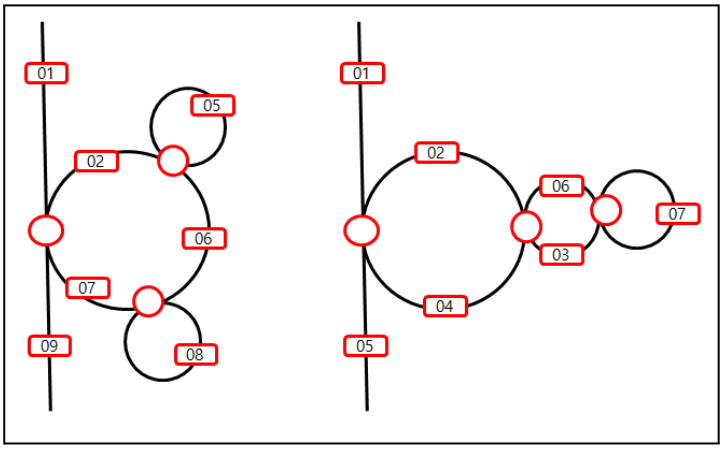
다음을 생각해 보자. 별을 4번찍다가 엔터를 한번치고 별을 4번찍다가 엔터를 한번치고 이 작업을 4번 반복하면 ' ****엔터 ****엔터 ****엔터 ****엔터 '와 같이 출력이 될 것이고 실제로 엔터가 줄을 바꾼다면 왼쪽 처럼 직사각형 모양이 될 것이다. 이것을 기찻길로 만든다고 생각해보자. ****은 별한개 출력하는 작업을 4번 반복하는것과 같다. 이는 아래 왼쪽 상단의 기찻길 모양과 같이 될것이다. 그 다음에 엔터를 쳐야 할 것이다.



이것은 왼쪽 아래 이미지처럼 별 4 번 찍은 것에 엔터 한번 친 것이 될 것이다. 별 4번 찍고 엔터 1 번 치는 것을 총네 번 반복해야 된다. 빨간색 동그라미 안에 숫자가 반복 횟수 라면 최종 결과가 왼쪽 이미지 처럼 될 것이고 이것은 사각형 모양을 화면에 출력하는 기찻길이 될 것이다. 왼쪽 이미지 기찻길을 순서도와 의사 코드로 만들어 보자.

문제 1) 아래처럼 출력될 수 있도록 기찻길을 만들고 순서도를 만든 다음 의사 코드를 작성해보자.

1. 1**********1
2. *****1*****1*****1*****1
3. 2*****2*****2*****2*****
4. 21**********1
5. 1****21****21****21****2
6. 1111****21111****21111****21111****2
7. 1111****22221111****22221111****2222
8. 11112222333344441111222233334444111122223333444

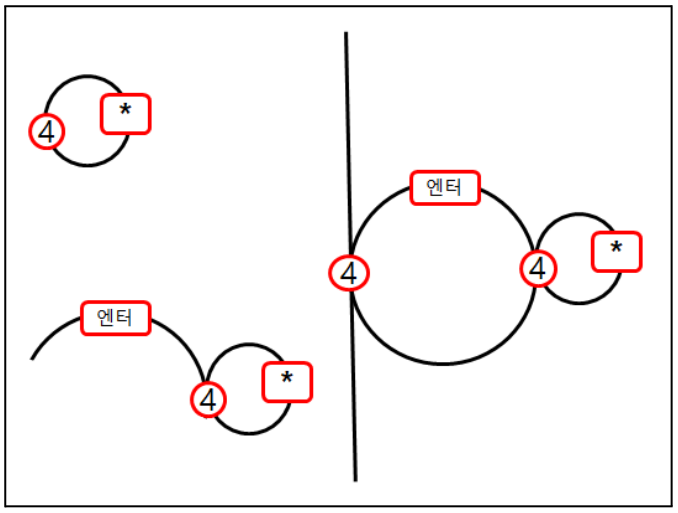


문제 1)왼쪽 기찻길 이미지 두개를 순서도와 의사 코드로 구현하여 보자.

이중 반복문을 이용해서 아래 이미지 처럼 정사각형 별을 찍는 프로그램을 작성하는 방법을 생각해보자.

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

다음을 생각해보자. 별을 4번찍다가 엔터를 한번치고 별을 4번찍다가 엔터를 한번치고 이 작업을 4번 반복하면 ' ****엔터 ****엔터 ****엔터 ****엔터 '와 같이 출력이 될 것이고 실제로 엔터가 줄을 바꾼다면 왼쪽 처럼 직사각형 모양이 될 것이다. 이것을 기찻길로 만든다고 생각해보자. ****은 별한개 출력하는 작업을 4번 반복하는것과 같다. 이는 아래 왼쪽 상단의 기찻길 모양과 같이 될것이다. 그 다음에 엔터를 쳐야 할 것이다.



이것은 왼쪽 아래 이미지처럼 별 4 번 찍은 것에 엔터 한번 친 것이 될 것이다. 별 4번 찍고 엔터 1 번 치는 것을 총네 번 반복해야 된다. 빨간색 동그라미 안에 숫자가 반복 횟수 라면 최종 결과가 왼쪽 이미지 처럼 될 것이고 이것은 사각형 모양을 화면에 출력하는 기찻길이 될 것이다. 코드로 구현해 보자.

별을 한개 찍는다. `System.out.print("*");`

상위 왼쪽상단 이미지 처럼 별을 4번 찍는다.

```
for(int i=0;i<4;i++) {
    System.out.print("*");
}
```

상위 왼쪽하단 이미지 처럼 별을 4번 찍고 엔터를 한번 친다.

```
for(int i=0;i<4;i++) {
    System.out.print("*");
}
System.out.println();
```

상위 오른쪽 이미지 처럼 별을 4번 찍고 엔터를 한번 치는 작업을 4번 반복한다.

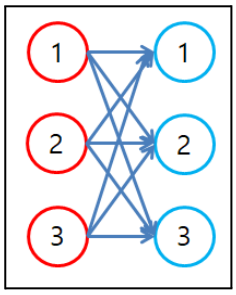
```
for(int j=0;j<4;j++) {
    for(int i=0;i<4;i++) {
        System.out.print("*");
    }
    System.out.println();
}
```

문제 1) 아래처럼 출력될 수 있도록 기찻길을 만들고 순서도를 만든 다음 코드를 작성해보자. 구현이 어려우면 기찻길을 그려보고 하자. 그래도 어려우면 순서도를 그려보고

하자.

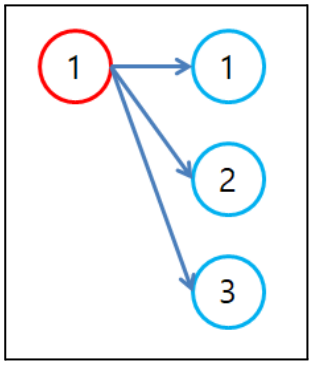
1. 1**********1
2. *****1*****1*****1*****1
3. 2*****2*****2*****2*****
4. 21**********21
5. 1****21****21****21****2
6. 1111****21111****21111****21111****2
7. 1111****22221111****22221111****2222
8. 11112222333344441111222233334444111122223333444

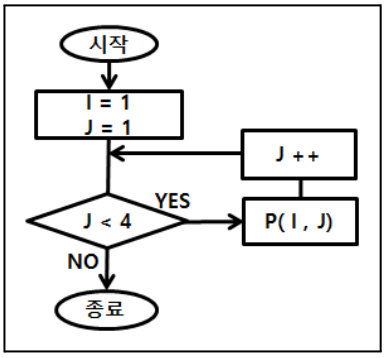
빨간공과 파란공을 모두 짝지어 찍는 프로그램을 구현해 보자.



빨간공 1,2,3 3개와 파란공 1,2,3 3개를 모두 짝지어 보자. 모든 결과를 출력해 보면 다음과 같은 결과를 가진다. (1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3)

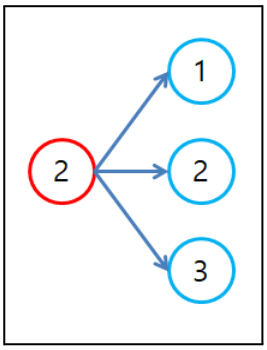
다음 3가지를 확인해 보자. 빨간색 1번 공을 가지고 파란공들과 짝을 지어 보면 (1,1), (1,2), (1,3) 과 같다. 빨간색 2번 공을 가지고 파란공들과 짝을 지어 보면 (2,1), (2,2), (2,3) 과 같다. 빨간색 3번 공을 가지고 파란공들과 짝을 지어 보면 (3,1), (3,2), (3,3)과 같다. 이 3가지를 하나씩 나눠서 순서도로 만들어 보면 다음과 같고 결국 3가지 결과를 합치면 빨간색 공과 파란색 공의 모든 짝이 된다.

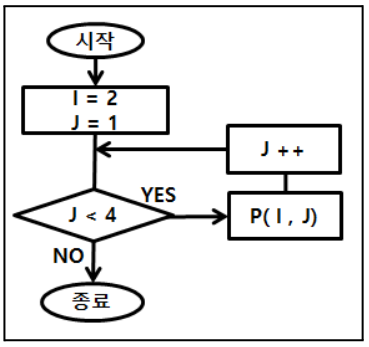




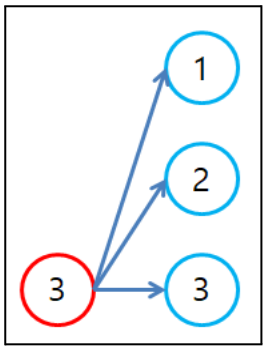
맨 왼쪽 이미지 처럼 빨간색 공 1개는 파란색공 3개와 짝을 지어보면 다음과 같이 될 것이다. 앞이 빨간공 뒤가 파란공이라면 (1,1), (1,2), (1,3) 와 같은 짝이 만들어 질 것이다. 자세히 보면 빨간 공은 1인 상태로 고정하고 파란 공을 1부터 3까지 하나씩 증가 하면서 짝을 지었다. 빨간색 공을 i라 하고 파란색 공을 j라 한다면 반복문을 이용해서 j값을 하나씩 증가시 키는 방법은 이미 공부 하여서 충분히 생각해 볼 수 있을 것이다. 고정된 값 i는 반복문 밖에 선언하고 반복문 안에서는 j 값을 1부터 하나씩 증가 시키며 i와 j 값을 출력하면 원하는 결과를 얻을 수 있을 것이다. 상위 순서도를 확인하고 기찻길과 의사코드를 만들어 보자. 순서도를 잘 따라 가며

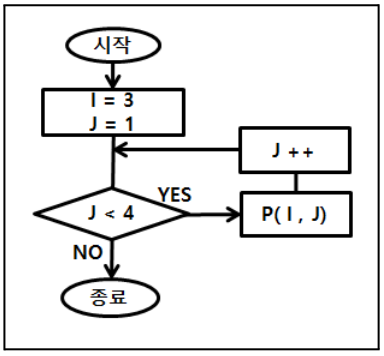
출력값을 확인해 보자.



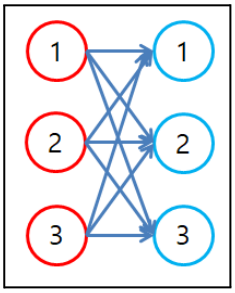


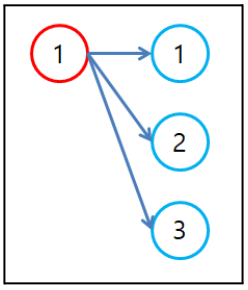
왼쪽 첫 번째 이미지 처럼 빨간 공이 2라면 (2, 1),(2, 2),(2, 3)와 같은 짝이 만들어 질 것이다. 이전 상황과 잘 비교해 보면 i 값이 2로 바뀐 것 말고는 변한게 없다는 것을 알 수 있다.

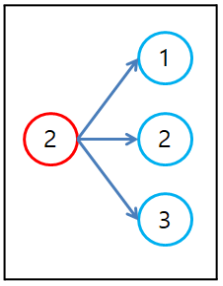


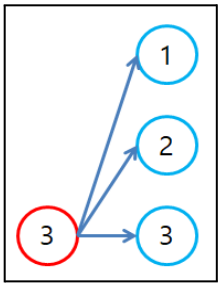


왼쪽 첫 번째 이미지 처럼 빨간 공이 3이라면 (3, 1),(3, 2),(3, 3)와 같은 짝이 만들어 질 것이다. 이전 상황과 잘 비교해 보면 i 값이 3로 바뀐 것 말고는 변한게 없다는 것을 알 수 있다.



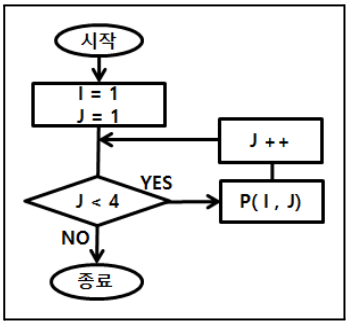


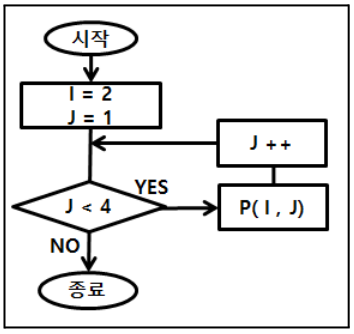


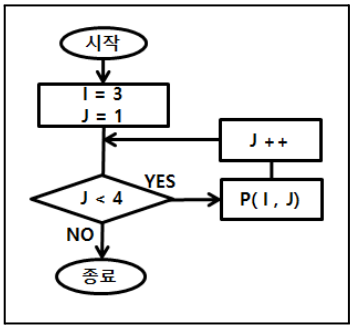


왼쪽 3개의 이미지를 합치면 맨 왼쪽 이미지 처럼 된다. 짝을 짓는다면 다음과 같은 결과가 나올 것이다. (1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), (3,3), 상위와 같이 출력 되려면 어떻게 순서도와 코드를 구현 할 것인지 생각해 보자.

잘 살펴보면 맨 왼쪽 이미지는 나머지 세 개 이미지를 합친 것과 같다. 따라서 오른쪽 세 개 이미지를 순서도로 그리고 순서대로 실행 한다면 같은 결과가 나올 것이다.







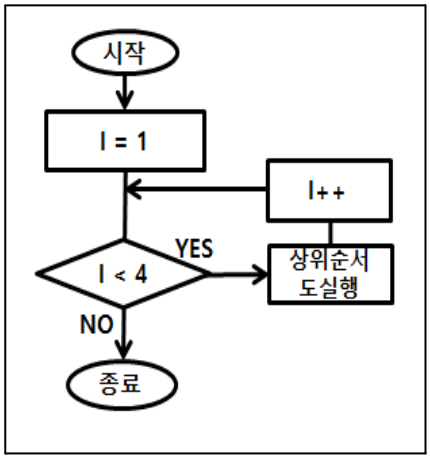
출력 결과를 잘 살펴보면 상위 첫 번째 순서도 출력결과는 (1, 1),(1, 2),(1, 3) 이다. 두 번째 순서도 출력결과는 (2, 1),(2, 2),(2, 3)이다. 세 번째 순서도 출력결과는 (3, 1),(3, 2),(3, 3) 이다. 세 가지 순서도를 왼쪽부터 순서대로 실행한다면 우리가 원하는 결과를 얻을 수 있고, 결국 (i, 1),(i, 2),(i, 3) 과 같음을 알 수 있다.

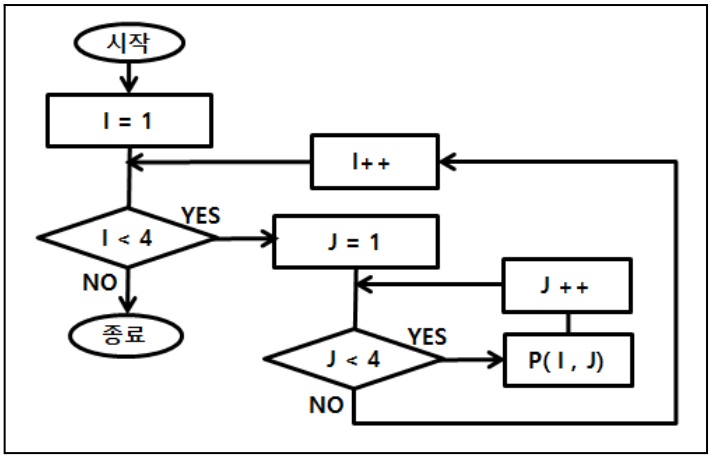
```
int i=1;
for(int j=1;j<4;j++) {
    System.out.println(i+":"+j);
}
```

다음을 자바코드로 변경하면 다음과 같다. i가 무엇이냐에 따라서 빨간색 공의 번호가 결정되는 것과 같다.

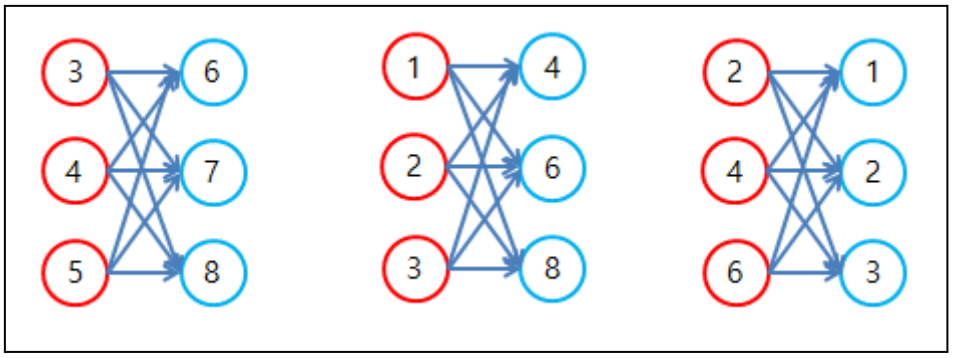
잘 생각해 보면 세 개의 순서도가 상당히 유사하다는 것을 알 수 있고 유일한 차이점을 찾는다면 첫 번째 순서도는 i는 1이고 두 번째 순서도는 i가 2이고 세 번째 순서도는 i가 3이라는 것이다. i 값이 하나씩 1, 2, 3 증가 하고 있다는 것만 빼고는 달라진 것이 하나도 없다. 결국에 i 값이 하나씩 증가 하면서 반복 되고 있다는 것이다. 숫자 하나씩 증가되면서 반복되는 것은 반복문으로 바꿀수 있다는 사실은 무수히 경험해 왔을 것이다. 반복문으로 만들어 보면 i값이 증가하는 형태의 반복문안에 상위 순서도 1개가 들어가 있으면 될 것이다.

상위 왼쪽 순서도를 확인해 보면 i가 1부터 3까지 3번 반복하면서 상위 순서도가 3번 실행된다. i가 1일때 i가 1인 상태에서 상위순서도가 실행이 되고 i가 2일때 i가 2인 상태에서 상위순서도가 실행이 되고 i가 3일때 i가 3인 상태에서 상위순서도가 실행이 되면 우리가 원하던 (1, 1),(1, 2),(1, 3) (2, 1),(2, 2),(2, 3) (3, 1),(3, 2),(3, 3) 결과를 얻을 수 있다. 다음 이미지는 상위 3개의 이미지를 합쳐서 하나의 순서도로 만들었다. i가 1씩 증가 하면서 상위 순서도를 하나씩 반복되는 형태이다. 천천히 확인해 보자.





오른쪽 순서도는 왼쪽 순서도에서 상위 순서도 실행 부분을 실제로 붙여서 만든 순서도로 하나하나 따라 가며 출력값을 확인해 보면 원하는 결과가 출력 된다는 것을 알 수 있다. 다음에 코드가 있는데 보지말고 for문과 while문으로 만들어 보자. 반복문으로 바꾸어 보자.



문제 1) 상위 3개의 이미지를 보고 어떤 결과가 출력되는지 기술 해 보고, 기찻길을 그려보고,순서도를 그려보고, 의사 코드를 만들어 보자.

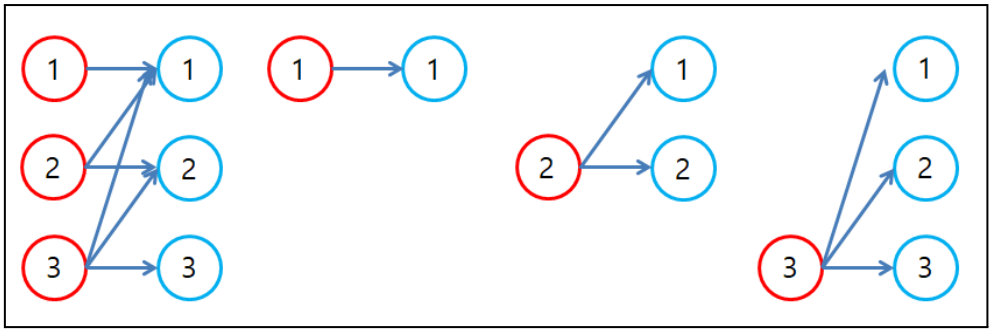
1*1=1	1*2=2	1*3=3	1*4=4	1*5=5	1*6=6	1*7=7	1*8=8	1*9=9
2*1=2	2*2=4	2*3=6	2*4=8	2*5=10	2*6=12	2*7=14	2*8=16	2*9=18
3*1=3	3*2=6	3*3=9	3*4=12	3*5=15	3*6=18	3*7=21	3*8=24	3*9=27
4*1=4	4*2=8	4*3=12	4*4=16	4*5=20	4*6=24	4*7=28	4*8=32	4*9=36
5*1=5	5*2=10	5*3=15	5*4=20	5*5=25	5*6=30	5*7=35	5*8=40	5*9=45
6*1=6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

1단 1*1=1 1*2=2 1*3=3 1*4=4 1*5=5 1*6=6 1*7=7 1*8=8 1*9=9 2단 2*1=2 2*2=4 2*3=6 2*4=8 2*5=10 2*6=12 2*7=14 2*8=16 2*9=18 3단 3*1=3 3*2=6 3*3=9 3*4=12 3*5=15 3*6=18 3*7=21 3*8=24

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

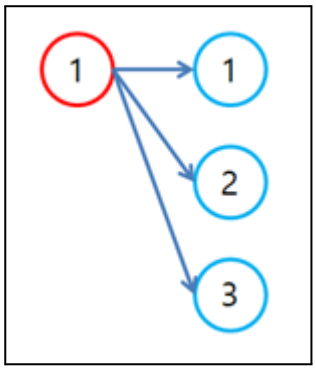
문제 2) 이미지와 같은 형태의 구구단을 출력할 수 있는 코드를 만들어 보자.

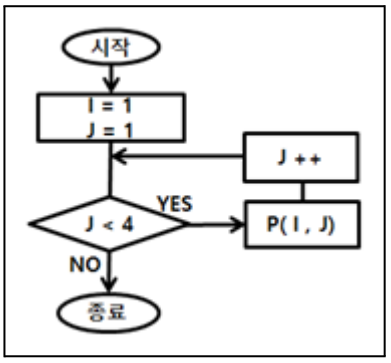
다음은 빨간색 i값이 파란색 j값에 영향을 주는 경우이다.

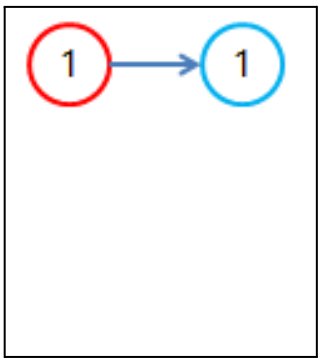


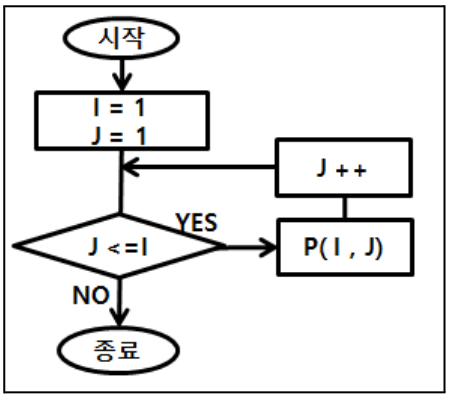
상위를 실행시키면 다음과 같은 결과가 나올 것이다. (1, 1) (2, 1),(2, 2) (3, 1), (3, 2),(3, 3) 빨간색 공을 i라 하고 파란색 공을 j라 한다면 i값은 j값을 찍는데 영향을 준다. i값이 1일때 j값은 1 한개 i값이 2일때는 j값 1,2 두 개 i값이 3일 때는 j값은 1, 2, 3 세개가 출력 된다. 잘 생각해 보면 파란공 j가 빨간공 i보다 같거나 작을때만 실행이

되는 것을 확인할 수 있다. i값에 따라 출력되는 결과가 규칙성 있게 달라지고 있고, 이렇게 i값이 j값 반복에 영향을 주려면 어떻게 해야 하는지 생각해 보자. i값이 j값 반복에 영향을 주려면 j값 반복문 안의 조건식으로 i값을 사용해야 한다.

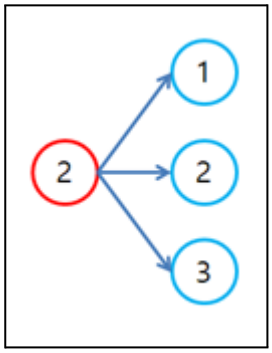


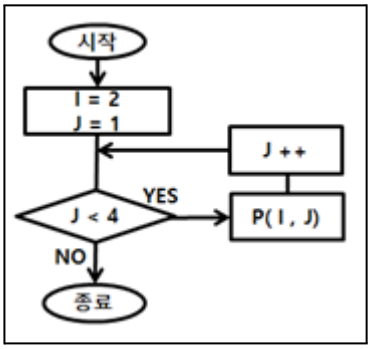


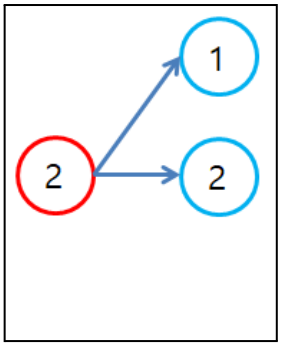


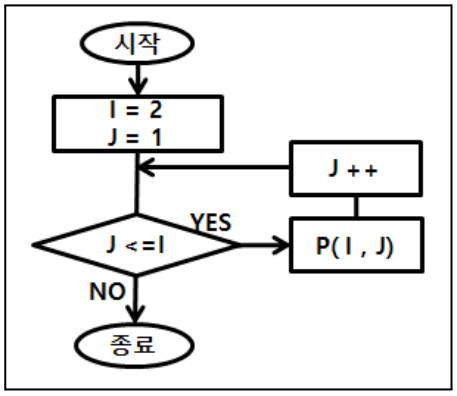


상위 첫번째 이미지와 두번째 이미지에서 반복문 밖의 i값이 반복문의 안쪽에서 출력 할때 영향을 주었다. 두번째 순서도 구조를 유지하면서 세번째 이미지 처럼 (1,1) 만 출력되도록 하려면 i값을 가지고 반복문의 조건식에 영향을 주어야 한다. 무조건 3번 반복되는 상태에서 i값이 j값보다 같거나 작을때만 반복 되도록 해야 한다. 그래서 조건식을 왼쪽 처럼 J<=i 과 같이 바꾸거나 J<i+1로 바꾸어 주면 된다. 이렇게 하면 조건식을 첫번째 돌때 j값이 1이고 i값은 1이므로 1<=1 비교가 true여서 반복 부분를 반복하고 두번째 돌때 j값이 2이고 i값은 여전이 1이여서 2<=1 비교가 false여서 반복문을 빠져나와 반복문이 종료 된다. 출력 결과는 (1, 1) 이다.



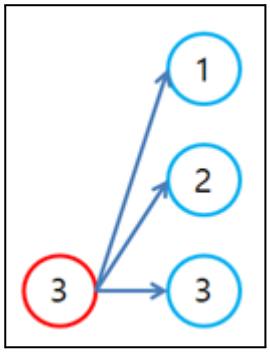


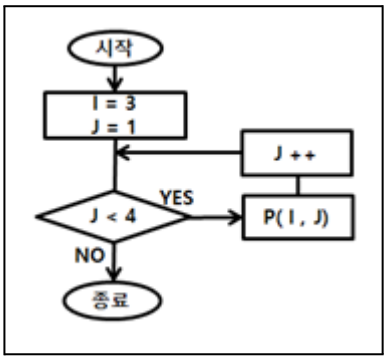


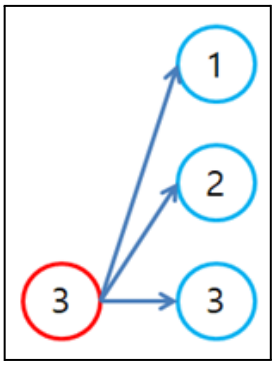


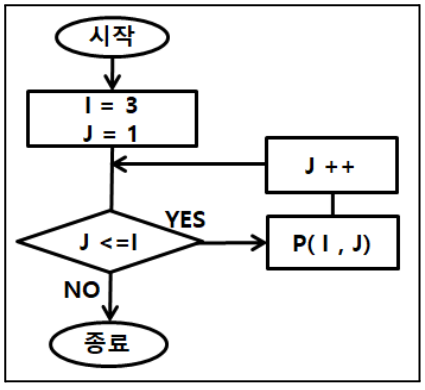
상위 첫번째 이미지와 두번째 이미지에서 반복문 밖의 i값이 반복문의 안쪽에서 출력 할때 영향을 주었다. 두번째 이미지 순서도 구조를 유지하면서 세번째 형태로 만들려면 i값을 가지고 반복문의 조건식에 영향을 주어야 한다. 무조건 3번 반복한 것을 i값에 따라서 반복 횟수를 줄여야 한다. 그래서 조건식을 왼쪽 처럼 J<i+1 나 J<=i로 바꾸어 주면 이전 빨간공이 1일때와 동일한 구조를 가지며 원하는 결과를 출력 할 수 있게 된다. 이렇게 하면 조건식을 첫번째 돌때 j값이 1이고 i값은 2이므로 1<=2 비교가 true여서 반복 부분을 반복하고 두번째 돌때 j값이 2이고 i값은 여전히 2이여서

2<=2 비교가 true여서 반복 부분을 반복하고 세번째 돌대 j값이 3이고 i값은 여전히 2여서 3<=2 비교가 false이므로 반복문을 빠져나와 반복문이 종료 된다.



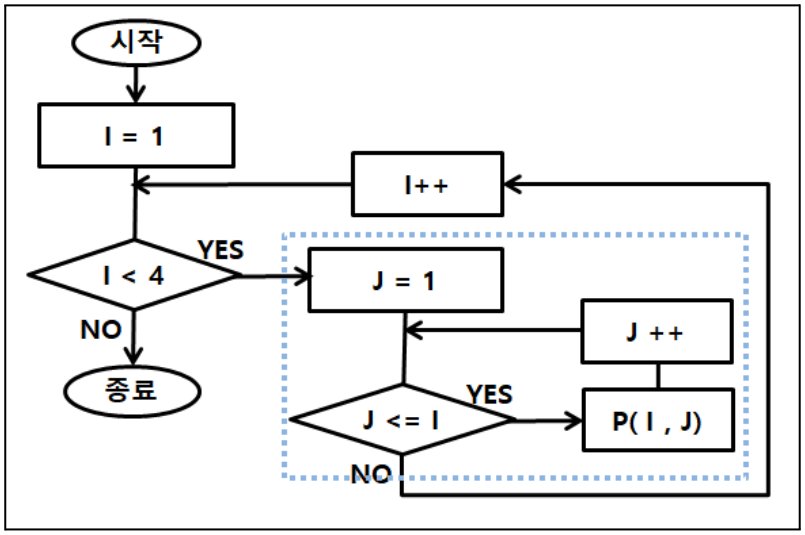






상위 첫번째 이미지와 세번째 이미지를 확인해 보면 동일하지만 네번째 이미지 순서도를 이용 해서 같은 결과를 얻을 수 있다.. 이렇게 하면 조건식을 첫번째 돌때 j값이 1이고 i값은 3이므로 1<=3 비교가 true여서 반복 부분을 반복하고 두번째 돌때 j값이 2이고 i값은 여전히 3이여서 2<=3 비교가 true여서 반복 부분을 반복하고 세번째 돌때 j값이 3이고 i값은 여전히 3이여서 비교식 3<=3의 결과가 true여서 반복 부분을 반복하고 네번째 돌대 j값이 4이고 i값은 여전히 3이여서 4<=3 비교 결과가 false이므로 반복문을 빠져나와 반복문이 종료 된다.

잘 생각해보면 상위 3개의 순서도는 i값이 1에서 하나씩 증가하여 3이 될 때까지 반복하는 것과 동일하여 다음과 같이 순서도를 변경할 수 있다.



왼쪽은 i값을 1, 2, 3씩 변경하면서 점선 부분을 반복하는 순서도이다.

1) 다음 문자열을 출력 할 수 있는 기찻길 순서도 의사 코드를 만들어 보자.

1. *엔터**엔터***엔터****엔터*****엔터******엔터*******엔터

2. *******엔터******엔터*****엔터****엔터***엔터**엔터*엔터

3. 공간공간공간공간공간공간*엔터공간공간공간공간공간**엔터공간공간공간공간**
*엔터공간공간공간****엔터공간공간*****엔터공간******엔터*******엔터

공간을 진짜 공간 키보드의 스페이스 ‘ ‘로 찍고 *은 *로 찍고 엔터는 실제 엔터

줄바꿈으로 표시하면 아래와 같은 삼각형 모양이 나올 것이다.

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

2)웹에서 자바 별찍기로 검색을 해서 다양함 별을 찍어 보자.

```
11111 12345 ******  *   *****  *****    *
22222 23456         **  ****   ****    ***
33333 34567 *       *** ***    ***    *****
44444 45678 *       **** **    **    *******
55555 56789 *       *****  *   *    *********
            *                  **    *******
12345 56789         *****  *   ***    *****
12345 45678 *****   ****   **  ****    ***
12345 34567 *****   ***    *** *****    *
12345 23456 *****   **     ****
12345 12345 *****   *      *****
            *****
```

다음 6,5,1,8,7,4,2,3을 값으로 가지는 배열이 아래와 같이 있을 때 배열의 내용을 1,2,3,4,5,6,7,8순으로 오름차 순으로 배열안의 내용을 바꾸어 정렬해 보자.

다음 버블정렬을 이해해 보자.

1. 배열의 인덱스 0과 1에 들어 있는 수를 비교하여 큰수를 배열의 인덱스 1쪽으로 교환하여 이동 시킨다.
2. 0이 작으면 교환되지 않는다.

```
int arr[]={6,5,1,8,7,4,2,3};
if(arr[0]>arr[1]){
      int temp;
      temp=arr[0];
      arr[0]=arr[1];
      arr[1]=temp;
}
```

3. 배열의 인덱스 1과 2에 들어 있는 수를 비교하여 큰수를 배열의 인덱스 2쪽으로 교환하여 이동 시킨다.

```
//{5,6,1,8,7,4,2,3};
if(arr[1]>arr[2]){
      int temp;
      temp=arr[1];
      arr[1]=arr[2];
      arr[1]=temp;
}
//{5,1,6,8,7,4,2,3};
```

4. 이작업을 배열이 끝날때 까지 반복한다.
5. 이 작업을 배열이 끝날때 까지 반복하면 배열안에 가장 큰 수가 맨 마지막에 정렬된다.

```
for(int i=0;i<arr.lenth-1;i++){
      if(arr[i]>arr[i+1]){
          int temp;
          temp=arr[i];
          arr[i]=arr[i+1];
          arr[i]=temp;
      }
}
```

6. 이 작업을 2번 하면 1번째에 가장큰 수가 배열의 맨 마지막으로 이동하고 2번째에 그다음 큰수가 배열의 뒤에서 2번째 위치로 이동 한다.

```
for(int i=0;i<arr.lenth-1;i++){
      if(arr[i]>arr[i+1]){
            int temp;
            temp=arr[i];
            arr[i]=arr[i+1];
            arr[i]=temp;
      }
```

```
}
for(int i=0;i<arr.lenth-1;i++){
	if(arr[i]>arr[i+1]){
		int temp;
		temp=arr[i];
		arr[i]=arr[i+1];
		arr[i]=temp;
	}
}
```

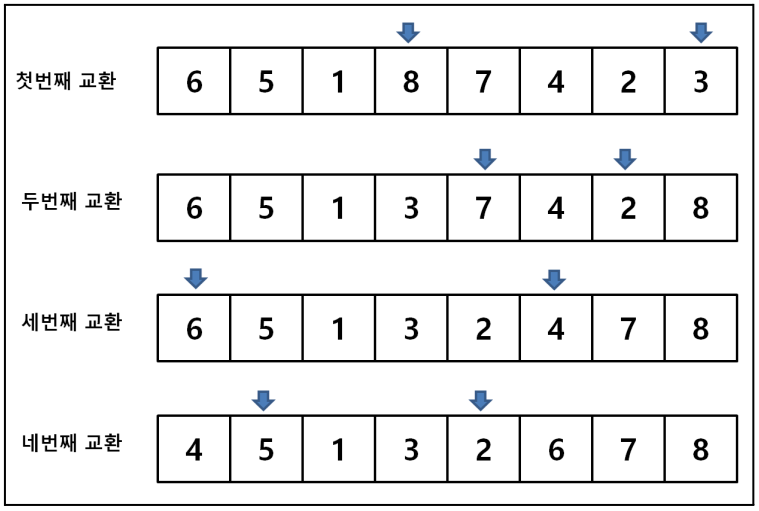
7. 반복문을 1번 돌때마다 1개의 데이터가 배열의 뒤쪽부터 정렬되므로 배열의 개수만큼 반복하면 배열의 모든 데이터가 정렬 된다.

```
for(int j=0;j<arr.lenth-1;j++){
	for(int i=0;i<arr.lenth-1;i++){
		if(arr[i]>arr[i+1]){
			int temp;
			temp=arr[i];
			arr[i]=arr[i+1];
			arr[i]=temp;
		}
	}
}
```

8. 내림차수로 정렬하고 싶다면 작은수를 뒷쪽으로 보내면 된다.

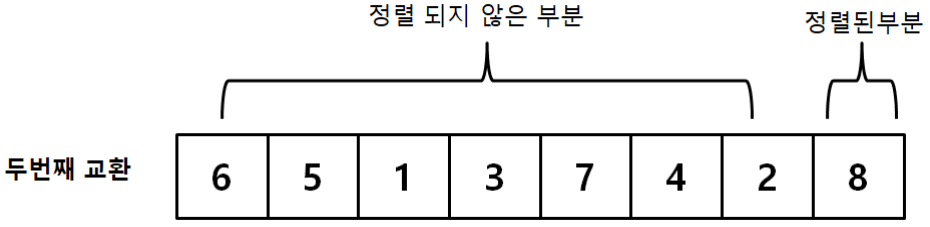
```
int [] arr= {7,5,9,0,3,1,6,2,4,8};
for(int i=0;i<arr.length;i++) {
	System.out.print(arr[i]+" ");
}
System.out.println();
for(int j=0;j<arr.length-1;j++) {
	for(int i=0;i<arr.length-2;i++) {
		if(arr[i] <arr[i+1]) {
			int temp=arr[i];
			arr[i]=arr[i+1];
			arr[i+1]=temp;
		}
	}
}
```

다음 선택정렬을 이해해 보자.



1. 배열안에서 가장 큰 수(첫번째 교환 부분의 첫 번째 화살표)를 찾고 배열의 마지막 위치(첫번째 교환 부분의 두 번째 화살표)와 두 값을 교환한다. 다음을 진행하고 나면 배열 안에 어떤 수가 들어 있더라 하더라도 가장 큰 수가 배열의 맨 마지막에 들어 간다.

2. 첫번째 교환이 이루어 지고 난 다음에는 배열의 맨마지막에 가장 큰 수가 들어가고 배열 안의 내용은 두번째 교환 부분과 같은 모양이 된다. 여기서 오름차순으로 배열 내용을 만든다고 생각해 보자. 맨 마지막 부분은 이미 정렬이 끝난 상태이므로더 이상 손을 댈 필요가 없다. 다음에 어떤 작업을 해할지 고민해보자.



3. 두번째 교환 부분에서 정렬 된 부분과 정렬되지 않은 부분을 구분한 다음 정렬된 부분은 더 이상 손대지 않으면 되고, 정렬되지 않은 부분중에서 가장 큰수 (두 번째 교환 부분의 첫 번째 화살표)를 찾아 정렬되지 않은 부분의 마지막 배열 부분(두 번째 교환 부분의 두 번째 화살표)의 값과 교환하면 세 번째 교환 부분 배열처럼 정렬이 된다.

4. 세번째 교환 부분에서 정렬 된 부분과 정렬되지 않은 부분을 구분한 다음 정렬된 부분은 더 이상 손대지 않으면 되고, 정렬되지 않은 부분중에서 가장 큰수 (세 번째 교환 부분의 첫 번째 화살표)를 찾아 정렬되지 않은 부분의 마지막 배열 부분(세 번째 교환 부분의 두 번째

화살표)의 값과 교환하면 네 번째 교환 부분 배열처럼 정렬이 된다. 이런 방법으로 계속 반복하다 보면 배열안의 모든 내용이 정렬될 것이다.

총 몇번 반복해야 정렬이 되는가?

3개라면 2개만 정렬되면 나머지 하나는 저절로 정렬이 된다. 따라서, 배열크기-1 한 만큼 반복하면 배열은 정렬이 된다. 이후 부터 배열의 크기를 배열이름.length로 표기 한다.

상위 내용을 정리해 보면 정렬하기 위해서 다음과 같은 과정을 거치면 된다.

0~7 까지 8개 중에 가장 큰수를 찾아 배열 인덱스 7에들어 있는 수와 교환해야 한다.

0~6 까지 7개 중에 가장 큰수를 찾아 배열 인덱스 6에들어 있는 수와 교환해야 한다.

0~5 까지 6개 중에 가장 큰수를 찾아 배열 인덱스 5에들어 있는 수와 교환해야 한다.

0~4 까지 5개 중에 가장 큰수를 찾아 배열 인덱스 4에들어 있는 수와 교환해야 한다.

0~3 까지 4개 중에 가장 큰수를 찾아 배열 인덱스 3에들어 있는 수와 교환해야 한다.

0~2 까지 3개 중에 가장 큰수를 찾아 배열 인덱스 2에들어 있는 수와 교환해야 한다.

0~1 까지 2개 중에 가장 큰수를 찾아 배열 인덱스 1에들어 있는 수와 교환해야 한다.

반복되는 숫자를 확인해 보면 반복문으로 바꿀수 있을 것이다.

> 08. 다양한 예제

문제 1) 1~50까지의 짝수를 출력하는 코드를 만들어 보자.

문제 2) 1~100사이의 10의 배수을 출력하는 코드를 만들어 보자. 어떤 숫자가 배수 인지 아닌지 알고 싶으면 7로 나눈 나머지가 0이면 7의 배수이고 아니면 7의 배수가 아니다.

문제 3) 30~300까지의 6의 배수의 합을 출력하는 코드를 만들어 보자.

문제 4) 숫자를 하나 입력 받아 1부터 입력한 수까지 순서대로 화면에 출력 되도록 코드를 만들어 보자.

문제 5) 사용자에게 두 수를 입력받아 두 수의 사이에 있는 모든 수를 오름차 순으로 출력하는 순서도와 프로그램을 만들어 보자. 예)5 9 입력시 6 7 8 더한 결과를 얻음

문제 6) 두수를 입력 받아 사이에 있는 짝수를 화면에 오름차 순으로 출력 되도록 순서도와 프로그램을 만들어 보자.

문제 7) 1-2+3-4+5-6……+99-100의 결과를 구하는 프로그램을 작성해 보자.

문제 8) 1/2+2/3+3/4+4/5+ 5/6+6/7+....+99/100의 결과를 구하는 프로그램을 작성해 보자.

문제 9) 피보나치 수열을 10개를 순서대로 출력하는 프로그램을 작성해 보자. 피보나치 수열이 무엇인지는 웹사이트를 검색해서 스스로 알아보자.

문제 10) 원하는 색의 전구와 밝기를 입력 받아 처리하는 프로그램을 만들어 보자. 사용자가 원하는 밝기와 색상을 입력받아 원하는 결과를 출력 받자. 아래 설명한 내용대로 운영 되도록 기술 하자.

초기 변수값 : color=”빨강” brightness =50

값의 범위 : color=빨강,노랑,파랑 brightness는 0~100

제안사항 : brightness의 숫자 변경은 1씩 가능하다. brightness값이 10이라면 다음 brightness값은 11 이나 9 만 가능하다. 10을 50으로 변경하려면 반복문을 사용해야 한다.

사용자입력변수 : colorInput, brightnessInput

최종결과값출력 : p(“현재 색상은”+color+”밝기는”+brightness+”이다”);

문제 11) 1-2+3-4+5-6……+99-100의 결과를 구하는 순서도와 프로그램을 작성해 보자.

문제 12) 1/2+2/3+3/4+4/5+ 5/6+6/7+....+99/100의 결과를 구하는 순서도와 프로그램을 작성해 보자.

문제 13) 다음과 같이 출력 되도록 프로그램을 완성해 보자.

1 2 3 4 5

10 9 8 7 6

11 12 13 14 15

20 19 18 17 16

21 22 23 24 25

문제 14) 해당 달의 시작 요일과 일수를 입력 받아 달력을 출력해 보자.\t 탭을 이용해서 만들어 보자.

문제 15) 배열 1,2,3,4,5,6,7,8,9 에서 이동방향, 이동칸수, 채울수자를 입력 받아 배열의 내용을 변경후 출력해보자.

ex)입력 왼쪽 3 2 결과 4,5,6,7,8,9,2,2,2

ex)입력 오른쪽 3 4 결과 4,4,4,1,2,3,4,5,6

문제 16)배열 1,2,3,4,5,6,7,8,9 에서 회전방향과 회수를 입력받아 배열 내용을 회전시키고 출력해보자.

ex)입력 왼쪽 2 결과 3,4,5,6,7,8,9,1,2

ex)입력 오른쪽 3 결과 7,8,9,1,2,3,4,5,6

문제 17)배열을 100개 선언하여 0~99까지 넣은 다음 i=2 부터 50까지 i를 제외한 i의 배수와 같은 인덱스에 0를 넣은 다음 배열에 0이 아닌 수를 출력해 보자. 출력 결과가 모두 소수인데 이유를 생각해 보자.

ex)i가 2이면 2를 제외한 2의 배수는 4,6,8,10,12,14,16… 등이 있고 해당 인덱스에 0를 넣으면된다.

ex)i가 3이면 3를 제외한 3의 배수는 6,9,12,15,18… 등이 있고 해당 인덱스에 0를 넣으면된다.

문제 18) 17번에서 배열의 값이 0이 아닌 수를 출력해보면 나오는 결과물이 소수이다. 소수란 1과 자기 자신만으로 나눠지는 수를 의미한다.

```java
double sum=0;
for(int i=1;i<100;i++) {
	sum+=(double)i/(i+1);
}
System.out.println(sum);

String color="빨강";
int brightness=50;

String colorInput="노랑";
int brightnessInput=70;

color=colorInput;
while(brightness!=brightnessInput) {
	if(brightness>brightnessInput) {
		brightness--;
	}else {
		brightness++;
	}
}
System.out.println("현재 색상은"+color+"밝기는"+brightness+"이다");


//1   1   2   3   5   8
int preValue=1;
int curValue=1;
System.out.println(preValue);
for(int i=0;i<9;i++) {
	System.out.println(curValue);
	int temp=curValue;
	curValue=preValue+curValue;
	preValue=temp;
}
```

> 09. 2중 메뉴

메뉴 아래 하위메뉴가 존재할때 처리방법

프로그램 설명:

1. **메인 메뉴**:
    - 사용자는 커피 메뉴(1) 또는 디저트 메뉴(2)를 선택할 수 있습니다.
    - `0`을 입력하면 프로그램이 종료됩니다.
2. **커피 메뉴**:
    - 아메리카노, 카페라떼, 카푸치노, 에스프레소 중 선택할 수 있습니다.
    - `0`을 입력하면 메인 메뉴로 돌아갑니다.
3. **디저트 메뉴**:
    - 케이크, 마카롱, 쿠키 중 선택할 수 있습니다.
    - 각 디저트는 하위 메뉴를 가지고 있습니다.
    - `0`을 입력하면 메인 메뉴로 돌아갑니다.
4. **하위 메뉴**:
    - 케이크, 마카롱, 쿠키의 하위 메뉴에서 추가 옵션을 선택할 수 있습니다.
    - `0`을 입력하면 디저트 메뉴로 돌아갑니다.

실행화면

```
=== 메인 메뉴 ===

1. 커피 메뉴

2. 디저트 메뉴

0. 종료

메뉴를 선택하세요: 2


=== 디저트 메뉴 ===

1. 케이크
```

2. 마카롱

3. 쿠키

0. 메인 메뉴로 돌아가기

디저트 메뉴를 선택하세요: 1

케이크를 선택하셨습니다. 가격은 5000원입니다.

=== 케이크 하위 메뉴 ===

1. 초코케이크

2. 치즈케이크

3. 당근케이크

0. 디저트 메뉴로 돌아가기

케이크 종류를 선택하세요: 1

초코케이크를 선택하셨습니다. 추가 가격은 1000원입니다.

```
package com.the.ex;

import java.util.Scanner;

public class CoffeeAndDessertMenu {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String inputString = "";

        while (!inputString.equals("0")) {
            System.out.println("=== 메인 메뉴 ===");
            System.out.println("1. 커피 메뉴");
            System.out.println("2. 디저트 메뉴");
            System.out.println("0. 종료");
            System.out.print("메뉴를 선택하세요: ");

            inputString = scanner.nextLine();

            switch (inputString) {
                case "1":
                    coffeeMenu(scanner);
                    break;
                case "2":
                    dessertMenu(scanner);
                    break;
                case "0":
```

```
                System.out.println("프로그램을 종료합니다.");
                break;
            default:
                System.out.println("잘못된 입력입니다. 다시 선택해주세요.");
        }
        System.out.println(); // 줄바꿈
    }

    scanner.close();
    System.out.println("프로그램이 종료되었습니다.");
}

// 커피 메뉴 처리
private static void coffeeMenu(Scanner scanner) {
    String inputString = "";

    while (!inputString.equals("0")) {
        System.out.println("=== 커피 메뉴 ===");
        System.out.println("1. 아메리카노");
        System.out.println("2. 카페라떼");
        System.out.println("3. 카푸치노");
        System.out.println("4. 에스프레소");
        System.out.println("0. 메인 메뉴로 돌아가기");
        System.out.print("커피 메뉴를 선택하세요: ");

        inputString = scanner.nextLine();
```

```
            switch (inputString) {
                case "1":
                    System.out.println("아메리카노를 선택하셨습니다. 가격은
3000원입니다.");
                    break;
                case "2":
                    System.out.println("카페라떼를 선택하셨습니다. 가격은
3500원입니다.");
                    break;
                case "3":
                    System.out.println("카푸치노를 선택하셨습니다. 가격은
4000원입니다.");
                    break;
                case "4":
                    System.out.println("에스프레소를 선택하셨습니다. 가격은
2500원입니다.");
                    break;
                case "0":
                    System.out.println("메인 메뉴로 돌아갑니다.");
                    break;
                default:
                    System.out.println("잘못된 입력입니다. 다시 선택해주세요.");
            }
            System.out.println(); // 줄바꿈
        }
    }
```

```
    // 디저트 메뉴 처리
    private static void dessertMenu(Scanner scanner) {
        String inputString = "";

        while (!inputString.equals("0")) {
            System.out.println("=== 디저트 메뉴 ===");
            System.out.println("1. 케이크");
            System.out.println("2. 마카롱");
            System.out.println("3. 쿠키");
            System.out.println("0. 메인 메뉴로 돌아가기");
            System.out.print("디저트 메뉴를 선택하세요: ");

            inputString = scanner.nextLine();

            switch (inputString) {
                case "1":
                    System.out.println("케이크를 선택하셨습니다. 가격은
5000원입니다.");
                    cakeSubMenu(scanner);
                    break;
                case "2":
                    System.out.println("마카롱을 선택하셨습니다. 가격은
3000원입니다.");
                    macaronSubMenu(scanner);
                    break;
                case "3":
                    System.out.println("쿠키를 선택하셨습니다. 가격은
2000원입니다.");
```

```
                cookieSubMenu(scanner);
                break;
            case "0":
                System.out.println("메인 메뉴로 돌아갑니다.");
                break;
            default:
                System.out.println("잘못된 입력입니다. 다시 선택해주세요.");
        }
        System.out.println(); // 줄바꿈
    }
}

// 케이크 하위 메뉴
private static void cakeSubMenu(Scanner scanner) {
    String inputString = "";

    while (!inputString.equals("0")) {
        System.out.println("=== 케이크 하위 메뉴 ===");
        System.out.println("1. 초코케이크");
        System.out.println("2. 치즈케이크");
        System.out.println("3. 당근케이크");
        System.out.println("0. 디저트 메뉴로 돌아가기");
        System.out.print("케이크 종류를 선택하세요: ");

        inputString = scanner.nextLine();
```

```
            switch (inputString) {
                case "1":
                    System.out.println("초코케이크를 선택하셨습니다. 추가 가격은
1000원입니다.");
                    break;
                case "2":
                    System.out.println("치즈케이크를 선택하셨습니다. 추가 가격은
1500원입니다.");
                    break;
                case "3":
                    System.out.println("당근케이크를 선택하셨습니다. 추가 가격은
1200원입니다.");
                    break;
                case "0":
                    System.out.println("디저트 메뉴로 돌아갑니다.");
                    break;
                default:
                    System.out.println("잘못된 입력입니다. 다시 선택해주세요.");
            }
            System.out.println(); // 줄바꿈
        }
    }

    // 마카롱 하위 메뉴
    private static void macaronSubMenu(Scanner scanner) {
        String inputString = "";

        while (!inputString.equals("0")) {
```

```
            System.out.println("=== 마카롱 하위 메뉴 ===");
            System.out.println("1. 딸기 마카롱");
            System.out.println("2. 초코 마카롱");
            System.out.println("3. 바닐라 마카롱");
            System.out.println("0. 디저트 메뉴로 돌아가기");
            System.out.print("마카롱 종류를 선택하세요: ");

            inputString = scanner.nextLine();

            switch (inputString) {
                case "1":
                    System.out.println("딸기 마카롱을 선택하셨습니다. 추가 가격은
500원입니다.");
                    break;
                case "2":
                    System.out.println("초코 마카롱을 선택하셨습니다. 추가 가격은
500원입니다.");
                    break;
                case "3":
                    System.out.println("바닐라 마카롱을 선택하셨습니다. 추가 가격은
500원입니다.");
                    break;
                case "0":
                    System.out.println("디저트 메뉴로 돌아갑니다.");
                    break;
                default:
                    System.out.println("잘못된 입력입니다. 다시 선택해주세요.");
            }
```

```
            System.out.println(); // 줄바꿈
        }
    }

    // 쿠키 하위 메뉴
    private static void cookieSubMenu(Scanner scanner) {
        String inputString = "";

        while (!inputString.equals("0")) {
            System.out.println("=== 쿠키 하위 메뉴 ===");
            System.out.println("1. 초코칩 쿠키");
            System.out.println("2. 오트밀 쿠키");
            System.out.println("3. 버터 쿠키");
            System.out.println("0. 디저트 메뉴로 돌아가기");
            System.out.print("쿠키 종류를 선택하세요: ");

            inputString = scanner.nextLine();

            switch (inputString) {
                case "1":
                    System.out.println("초코칩 쿠키를 선택하셨습니다. 추가 가격은
300원입니다.");
                    break;
                case "2":
                    System.out.println("오트밀 쿠키를 선택하셨습니다. 추가 가격은
400원입니다.");
                    break;
```

```
                case "3":
                    System.out.println("버터 쿠키를 선택하셨습니다. 추가 가격은
350원입니다.");
                    break;
                case "0":
                    System.out.println("디저트 메뉴로 돌아갑니다.");
                    break;
                default:
                    System.out.println("잘못된 입력입니다. 다시 선택해주세요.");
            }
            System.out.println(); // 줄바꿈
        }
    }
}
```

> 09. 생성자 메소드

1. 현재 코드 (필드 직접 접근)

```
class Human {
    public int age;        // 필드를 public으로 선언
    public String name;    // 필드를 public으로 선언
}
public class Main {
    public static void main(String[] args) {
        Human h = new Human();          // Human 객체 생성
        h.age = 25;                     // 필드에 직접 값 설정
        h.name = "Alice";
        // 필드 값 출력
        System.out.println(h.name + "의 나이는 " + h.age + "살입니다.");
        h.age = 30;        // 필드 값 변경
        h.name = "Bob";
        // 변경된 값 출력
        System.out.println(h.name + "의 나이는 " + h.age + "살입니다.");
    }
}
```

2. 생성자를 사용해 값을 설정

생성자를 사용하면 객체를 생성하는 동시에 필드 값을 초기화할 수 있습니다.

```
class Human {
    public int age;
    public String name;

    // 생성자 정의
    public Human(int age, String name) {
        this.age = age;
```

```
        this.name = name;
    }
}

public class Main {
    public static void main(String[] args) {
        // Human 객체 생성과 동시에 값 설정
        Human h = new Human(25, "Alice");

        // 필드 값 출력
        System.out.println(h.name + "의 나이는 " + h.age + "살입니다.");

        // 새로운 객체 생성
        Human h2 = new Human(30, "Bob");

        // 새로운 객체의 필드 값 출력
        System.out.println(h2.name + "의 나이는 " + h2.age + "살입니다.");
    }
}
```

자바에서 **생성자**(Constructor)는 클래스가 객체로 초기화될 때 호출되는 특수한 메서드입니다. 생성자는 객체를 생성하고 초기화하는 데 사용됩니다. 생성자의 주요 특징과 사용법은 다음과 같습니다:

---

1. 생성자의 특징

- **클래스 이름과 동일**: 생성자의 이름은 클래스의 이름과 동일해야 합니다.
- **반환 타입 없음**: 생성자는 반환 타입을 명시하지 않으며, void도 쓰지 않습니다.
- **자동 호출**: 객체가 생성될 때 자동으로 호출됩니다.
- **오버로딩 가능**: 생성자는 매개변수의 개수나 타입에 따라 여러 개를 정의할 수 있습니다.

---

2. 기본 생성자

클래스에 생성자를 명시적으로 정의하지 않으면, 컴파일러가 자동으로 매개변수가 없는 기본 생성자를 추가합니다. 본인이 직접 기술해서 기본 생성자를 만들수 있다.

```
class Human {
    int age;
    String name;

    // 기본 생성자
    public Human() {
        System.out.println("기본 생성자가 호출되었습니다.");
    }
}

public class Main {
    public static void main(String[] args) {
        Human h = new Human(); // 기본 생성자 호출
    }
}
```

---

3. 매개변수가 있는 생성자

생성자를 통해 객체 생성 시 원하는 값으로 초기화할 수 있습니다.

```
class Human {
    int age;
    String name;

    // 매개변수가 있는 생성자
    public Human(int age, String name) {
        this.age = age;
        this.name = name;
```

```
    }
}

public class Main {
    public static void main(String[] args) {
        Human h = new Human(25, "Alice");
        System.out.println(h.name + "의 나이는 " + h.age + "살입니다.");
    }
}
```

4. 매개변수가 있는 생성자를 만들면 디폴트 생성자도 만들어 주어야 한다.

클래스에 매개변수 생성자를 추가하면, 컴파일러는 디폴트 생성자를 더 이상 추가하지 않습니다.

```
class Human {
    public int age;
    public String name;

    // 매개변수 생성자
    public Human(int age, String name) {
        this.age = age;
        this.name = name;
    }
}

public class Main {
    public static void main(String[] args) {
        Human h = new Human(); // 오류 발생
    }
}
```

오류 내용:

- **문제 원인**: 컴파일러는 디폴트 생성자를 자동으로 추가하지 않았기 때문에 `new Human()` 호출 시 적절한 생성자를 찾지 못함.
- 클래스에는 **매개변수 생성자**만 존재하고 **디폴트 생성자**는 없음.

문제 해결: 디폴트 생성자 명시적으로 추가하기

디폴트 생성자를 직접 정의하면 문제가 해결됩니다:

```
class Human {
    public int age;
    public String name;

    // 디폴트 생성자
    public Human() {
        // 원하는 초기화 로직 추가 가능
        System.out.println("디폴트 생성자 호출!");
    }

    // 매개변수 생성자
    public Human(int age, String name) {
        this.age = age;
        this.name = name;
    }
}

public class Main {
    public static void main(String[] args) {
        Human h1 = new Human();               // 디폴트 생성자 호출
        Human h2 = new Human(25, "Alice");    // 매개변수 생성자 호출
    }
}
```

> 09. 용어정리

아래는 클래스의 구성 요소(필드, 생성자, 메서드)를 하나의 코드로 작성한 예제와 함께 설명입니다.

```
public class Example {

    // 1. 클래스 필드 (Static Field)

    static String classField = "이건 클래스 필드입니다.";


    // 2. 인스턴스 필드 (Instance Field)

    String instanceField;


    // 3. 생성자 (Constructor)

    public Example(String instanceField) {

        // 생성자를 통해 인스턴스 필드를 초기화

        this.instanceField = instanceField;

    }


    // 4. 클래스 메서드 (Static Method)

    public static void printClassField() {

        System.out.println("클래스 필드 값: " + classField);

    }


    // 5. 인스턴스 메서드 (Instance Method)

    public void printInstanceField() {

        System.out.println("인스턴스 필드 값: " + instanceField);

    }


    // 6. 정적 블록 (Static Block)

    static {
```

```
        System.out.println("정적 블록 실행: 클래스 필드 초기화 가능");
        classField = "초기화된 클래스 필드 값";
    }

    public static void main(String[] args) {
        // 클래스 필드 사용
        Example.printClassField(); // "클래스 필드 값: 초기화된 클래스 필드 값"

        // 객체 생성 (인스턴스화)
        Example example1 = new Example("첫 번째 인스턴스 필드 값");
        Example example2 = new Example("두 번째 인스턴스 필드 값");

        // 인스턴스 필드 출력
        example1.printInstanceField(); // "인스턴스 필드 값: 첫 번째 인스턴스 필드 값"
        example2.printInstanceField(); // "인스턴스 필드 값: 두 번째 인스턴스 필드 값"

        // 클래스 필드는 모든 인스턴스에서 공유됨
        Example.classField = "공유된 클래스 필드 값";
        Example.printClassField(); // "클래스 필드 값: 공유된 클래스 필드 값"
    }
}
```

설명

1. **클래스 필드 (Static Field)**
    - `static String classField`: 객체 생성 없이 `Example.classField`로 접근 가능.
    - 정적 블록에서 초기화하거나 클래스 메서드에서 접근 가능.
2. **인스턴스 필드 (Instance Field)**
    - `String instanceField`: 객체마다 개별 값을 가짐.
    - 생성자를 통해 초기화하며, `this.instanceField`를 사용.

3. **생성자 (Constructor)**
    - 클래스와 이름이 같으며 반환 타입이 없음.
    - 객체 생성 시 인스턴스 필드를 초기화.
4. **클래스 메서드 (Static Method)**
    - `printClassField`: 객체 없이 `Example.printClassField()`로 호출 가능.
5. **인스턴스 메서드 (Instance Method)**
    - `printInstanceField`: 특정 객체의 필드를 사용하며, 객체를 통해 호출.
6. **정적 블록 (Static Block)**
    - 클래스가 메모리에 로드될 때 실행.
    - 주로 클래스 필드 초기화에 사용.

출력 결과

```
정적 블록 실행: 클래스 필드 초기화 가능
클래스 필드 값: 초기화된 클래스 필드 값
인스턴스 필드 값: 첫 번째 인스턴스 필드 값
인스턴스 필드 값: 두 번째 인스턴스 필드 값
클래스 필드 값: 공유된 클래스 필드 값
```

이 코드로 클래스 구성 요소들의 역할과 사용 방법을 확인할 수 있습니다.