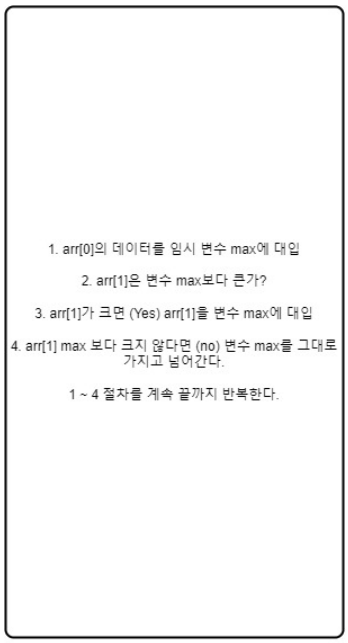
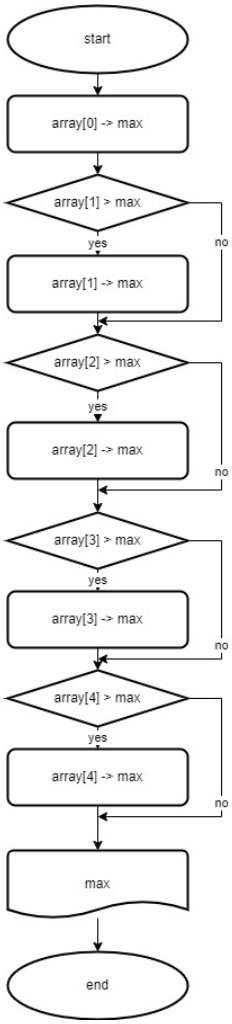
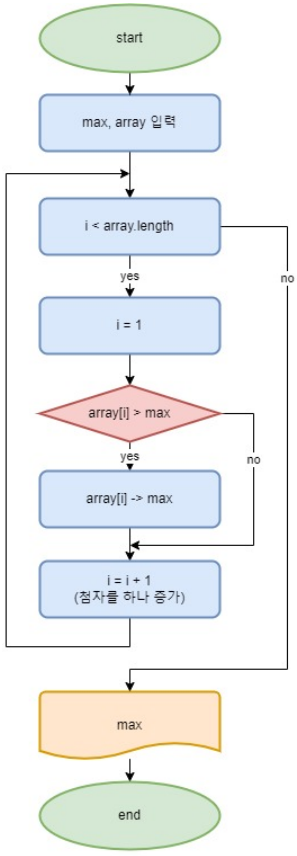
Day16; 20220721

작성일시	@2022년 8월 3일 오전 10:24
강의 번호	
유형	
자료	
복습	☐
속성	







```
package day16_1;

public class Max {

  public static void main(String[] args) {
    // TODO Auto-generated method stub
    int[] arr = {12, 13, 11, 14, 10};
    int max = arr[0];

    for(int i=1; i < arr.length; i++) {
      if(arr[i] > max) {
        max = arr[i];

      }
    }
    System.out.println(max);
  }

}
```

- 탐색 알고리즘
  - 원하는 데이터를 찾아내는 알고리즘
  - 네이버, 구글, 다음 등의 검색 엔진들도 모두 탐색 알고리즘을 매 순간 사용하고 있다.

- 선형 탐색법 Linear Search (선생님이 굳이 성능이 떨어지는데 쓸 필요는 없다고 하심)
  - 선형 탐색법은 맨 앞에서부터 순서대로 찾는 값을 검색하는 탐색 알고리즘이다.
  - 알고리즘 자체가 워낙 쉬워 이해하기 좋다.
  - 하지만 탐색 효율은 많이 떨어진다. 실용적이지는 않다.(너무 오래걸림)

선형탐색법은 왼쪽에서부터 순서대로 하나씩 확인하는 방법이다. 아무런 생각도 아무런 요령도 없는 아주 단순한 탐색법이다.

찾는 값이 앞쪽에 있으면 짧은 시간에 찾을 수 있지만 뒤쪽에 있거나 없을 때는 탐색하는데 많은 시간이 걸린다.

단숨에 이해하기 쉬운 알고리즘이지만 효율은 좋은 편은 아니다.

- 선형 탐색법 알고리즘
  - 배열에 보관 데이터를 맨 앞에서부터 순서대로 탐색하자.
  - 탐색처리는 반복 구조로 기술한다.
  - 반복 구조는 반드시 종료 조건이 중요하다.

  {4, 2, 3, 5, 1} ← 5를 찾는다.

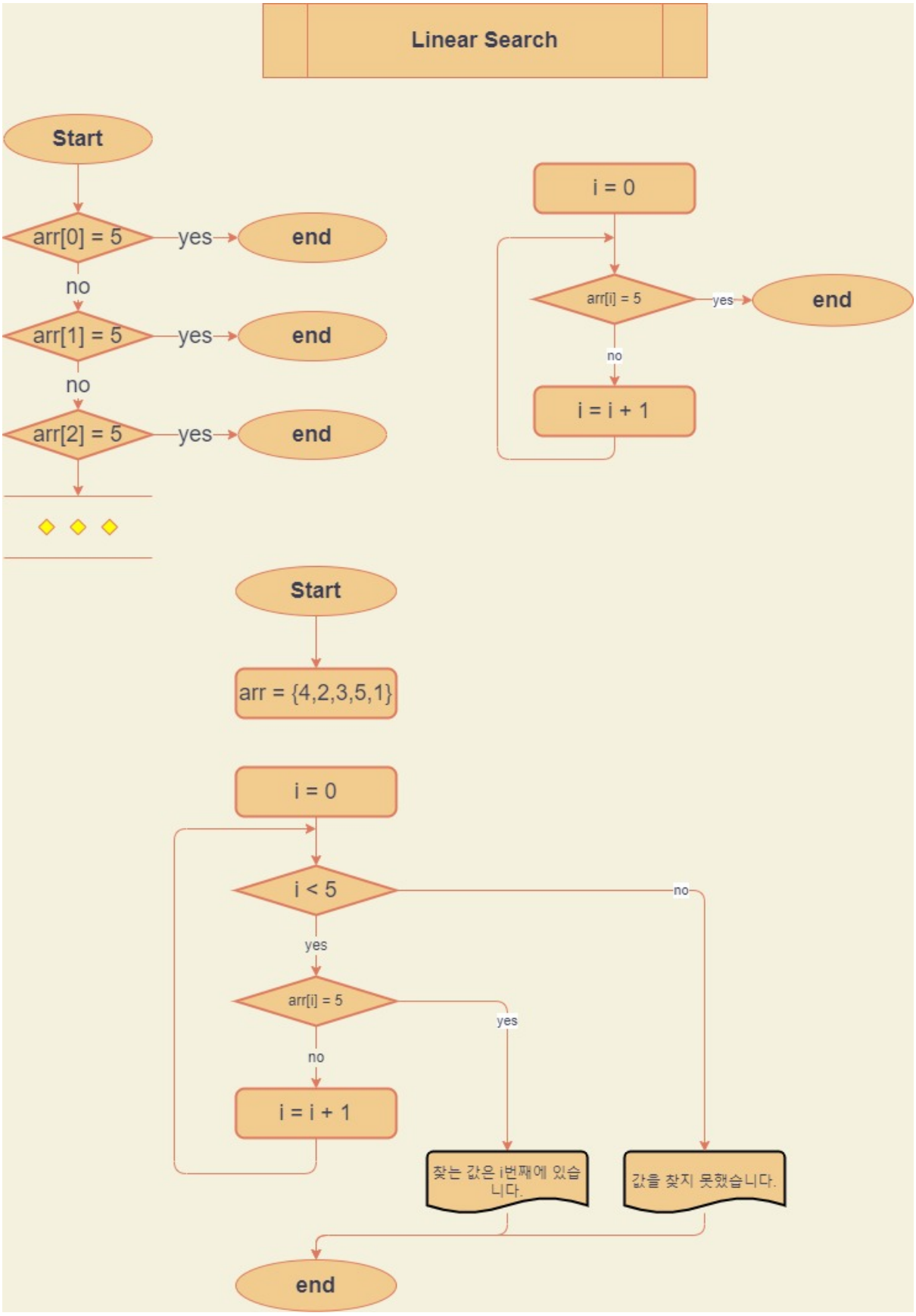
  arr[0] = 4

  arr[1] = 2

  .

  .

  arr[4] = 1


Linear Search
Start
arr[0] = 5
yes
end
no
arr[1] = 5
yes
end
no
arr[2] = 5
yes
end
i = 0
arr[i] = 5
yes
end
no
i = i + 1
Start
arr = {4,2,3,5,1}
i = 0
i < 5
no
yes
arr[i] = 5
yes
no
i = i + 1
찾는 값은 i번째에 있습니다.
값을 찾지 못했습니다.
end

```java
package day16_2;

public class Linear {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int[] arr = {4,2,3,5,1};
        int i;

        for(i = 0; i < arr.length; i++) {
            if(arr[i] == 5) {
                System.out.printf("찾는 값은 " + (i+1) + "번째에 있습니다");
                break;
            }
        }
        if(i == arr.length) {
            System.out.println("찾는 값이 없습니다");
        }

    }
}
```

- Binary Search 이진탐색
  - 원하는 데이터를 찾는 알고리즘
  - 반드시 찾는 데이터 전체가 정렬되어야만 사용할 수 있다. (전제조건)
  - 절반씩 대상 데이터를 줄여가며 탐색한다.

- 2진(수) 탐색 알고리즘
  - 가운데 요소를 찾는 처리
  - 가운데 요소와 원하는 데이터를 비교하는 처리
  - 탐색 범위를 절반으로 줄이는 처리

1. 가운데 요소를 찾는 처리 - 두 숫자의 가운데는 평균으로 구할 수 있다.

   0번방 + 6번방 = 6 / 2 = 3

   0 + 2 = 2 / 2 = 1
2. 가운데 요소와 원하는 데이터를 비교하는 처리

   center 중간값

   head

tail

(head + tail) / 2 = center

요소들의 개수가 짝수 일 때, 예를 들면 요소가 6이라고 생각해보면 center 후보가 2개가 된다. 하지만 2.5라는 첨자는 있을 수 없다. 이럴 경우는 소수점 이하 부분을 제거한 정수 부분을 취하여 인덱스를 사용하면 전혀 문제가 없다.

평균 계산을 통한 가운데 요소의 값과 찾는 값을 비교하여 만약 첫번에 일치하면 프로그램 종료하게 된다.

하지만 no의 경우 즉 원하는 데이터가 아닐 경우에는 두 가지 경우의 수가 발생한다.

찾는 값보다 작은 경우와 찾는 값보다 큰 경우의 둘 중 하나이다.

이 두가지 경우 모두 탐색 범위를 반으로 줄이는 처리로 이동한다

3. 탐색 범위를 절반으로 줄이는 처리
    a. 원하는 데이터(예: 17)가 가운데 데이터(예:9)보다 큰 경우 arr[center] < 17

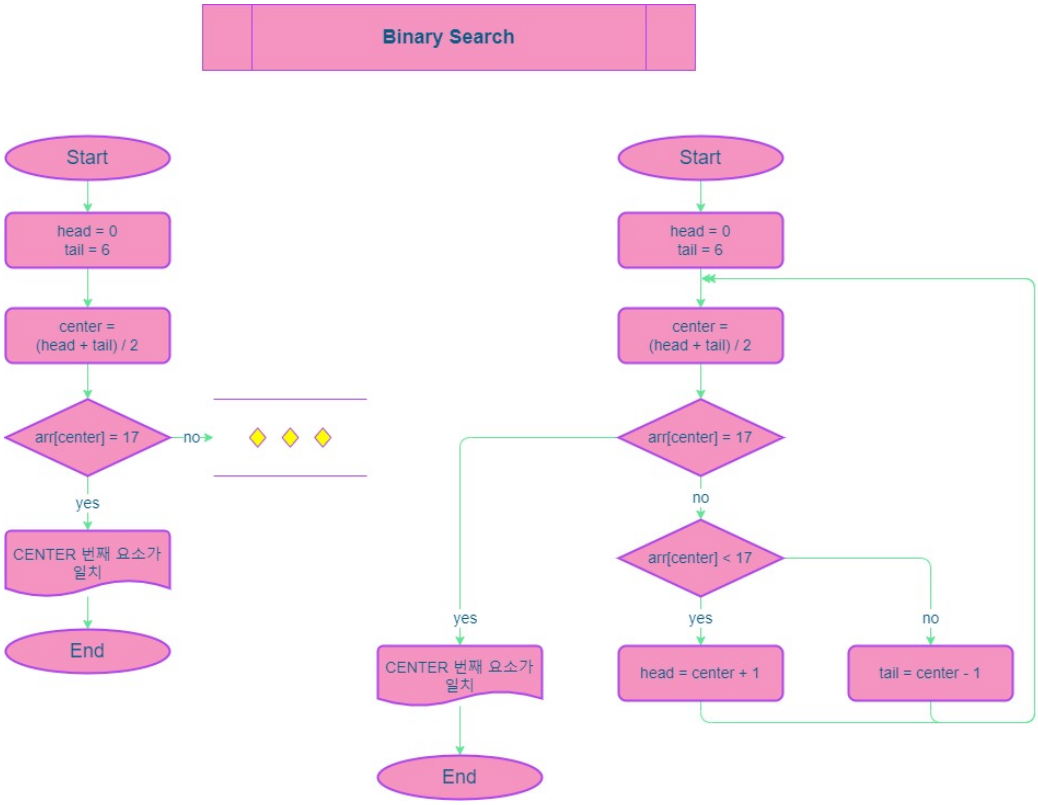
        이 경우 전체 검색 범위의 뒷부분으로 대상을 절반으로 좁힌다. 따라서 탐색 범위의 맨 앞 요소는 arr[center]보다 하나 큰 첨자를 갖는 요소가 된다.

        head = center + 1, tail 을 그대로 사용한다.

    b. 원하는 데이터 17 이 가운데 데이터 21보다 작은 경우 arr[center] > 17

        이 경우 전체 검색 범위의 앞으로 대상을 절반으로 좁힌다. 따라서 탐색 범위의 맨 뒷 요소는 arr[center]보다 하나 작은 첨자를 갖는 요소가 된다.

        tail = center - 1, head는 그대로 사용한다.


Binary Search
Start
head = 0
tail = 6
center =
(head + tail) / 2
arr[center] = 17
no
yes
CENTER 번째 요소가
일치
End
Start
head = 0
tail = 6
center =
(head + tail) / 2
arr[center] = 17
yes
CENTER 번째 요소가
일치
End
no
arr[center] < 17
yes
head = center + 1
no
tail = center - 1