소프트웨어 프로젝트 과제2

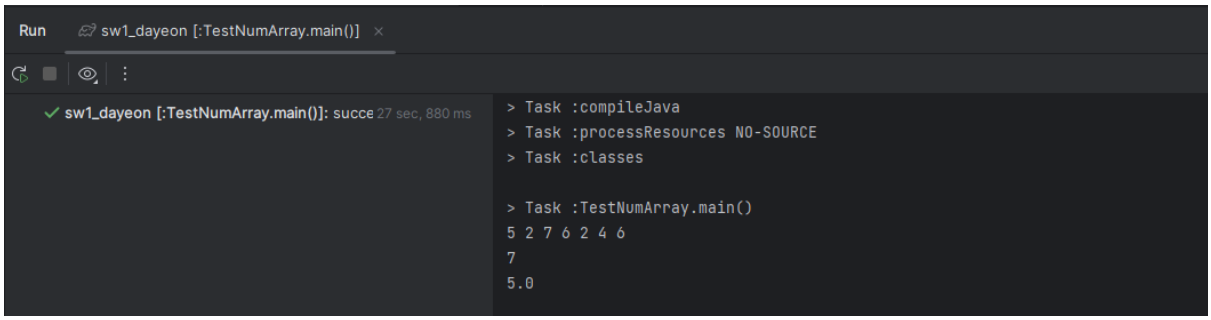
20213509 정다연

테스트 과정 화면

```java

public class TestNumArray{

    public static void main(String[] args) {
        int[] arr1 = {5,2,7,6};
        int[] arr2 = {2,4,6};

        numarray na = new numarray(arr1);

        int[] newArray = na.mergeArrays(arr2);
        numarray na2 = new numarray(newArray);
        na2.printArr();

        System.out.println(na.findMax());
        System.out.println(na.getAvg());

    }

}

```

별도 TestNumArray를 만들어서 지정된 동작을 수행하는지 테스트 진행한다.

- 합성된 배열의 경우 newArray로 return을 받고, 합성된 결과 출력을 위해 이를 numarray 클래스를 만들어 na2.printArr()로 메서드를 접근하여 출력하고자 하였다.
- 이후, 평균과 최대값의 경우 기존 arr1을 대상으로 진행하므로, na의 메서드를 활용하는 방식으로 코드를 작성한다.



C코드 및 java 코드 비교와 설계

findMax 메서드

```
1 usage
int findMax(){
    int max = arr[0];
    for(int i = 1 ; i < arr.length; i++){
        if(arr[i] > max){
            max = arr[i];
        }
    }
    return max;
}
```

C언어의 배열 접근 방식과 다르게 arr.length를 활용하여 길이를 인자로 받지 않고도 array의 속성을 활용하여 for문을 사용할 수 있는 방식으로 코드를 작성하였다.

C언어에서는 배열의 크기를 인자로 받아 for문을 활용하여 최대값을 찾지만, java에서는 arr.length를 활용하여 크기 인자 없이 최대값을 탐색한다. 이러한 방식을 사용하면서 추가적인 인자를 제거하여 오류 확률을 낮추고 배열 크기 변경에도 유연하게 java는 대응이 가능하다.

하지만 이때 java는 포인터연산이 불가능 하므로 메모리 주소를 활용한 성능적인 측면에서의 고고려 했을 때 좋지 않을 수 있다.

특징	**C언어**	**Java**
배열 크기	배열 크기를 인자로 받는다	Arr.length로 배열 크기를 자동으로 참조
배열 접근 방식	반복문에서 배열 크기를 인자로 사용	Arr.length로 배열 크기 확인 후 반복문 실행
유연성	배열 크기 변경 시 불편함	배열 크기 변경 시 유연하게 대응 가능
메모리 관리	포인터 연산을 통한 메모리 접근 가능	포인터 연산 불가, 메모리 주소 사용 불가
오류 가능성	배열 크기 오류가 발생 가능	Arr.length 사용으로 오류 확률 감소

getAvg 메서드

```
1 usage
float getAvg(){
    int sum = 0;
    for(int i : arr){
        sum += i;
    }
    return (float) sum / arr.length;
}
```

for문의 조건문과 증감값을 활용하지 않고도 for-each 구문으로 전체를 확인하는 구문으로 평균값을 구할 수 있다. 해당 방식을 사용하면 배열의 모든 요소를 쉽게 순회하는 것이 가능하다.

해당 방식을 사용하면 index 기반의 접근이 불가능하여 특정 조건에서의 탐색을 할 때에는 적합하지 않을 수 있다.

특징	C 언어	Java
배열 접근 방식	인덱스를 사용한 반복문(for)으로 배열 순회	for-each문을 사용해 배열 요소 순회
조건문과 증감값 사용	for문에서 조건문과 증감값을 명시적으로 설정	for-each문에서 조건문과 증감값 없이 자동 순회
평균 값 구하기	배열의 총합을 구한 후 배열 크기로 나누어 평균을 계산	for-each문을 사용하여 배열의 각 요소를 순회하고 평균 계산
유연성	조건에 맞는 특정 요소를 찾을 때 인덱스 기반 접근 필요	인덱스가 없으므로 특정 조건 탐색 시 불편할 수 있음

mergeArrays 메서드

```
1 usage
int[] mergeArrays(int[] arr1) {
    int[] newArr = new int[arr.length + arr1.length];
    System.arraycopy(arr, srcPos: 0, newArr, destPos: 0, arr.length);
    System.arraycopy(arr1, srcPos: 0, newArr, arr.length, arr1.length);
    return newArr;
}
```

배열을 병합할 때에도 new 키워드를 활용하여 배열을 동적으로 할당하고, System.arraycopy 메서드를 사용하여 배열을 병합하게 된다. 이때 C언어와 달리 Java에서는 메모리 관리가 자동으로 처리되므로 시스템에서 제공하는 메서드를 활용하여 배열을 합치는 것이 가능하다.

또한, C언어에서는 두개의 배열을 받아서 새로운 배열을 return 했지만, java 방식의 경우 추가로 붙일 배열 하나만 받은 이후, 클래스에서 갖고있던 arr 변수에 새로 인자로 받은 배열을 System.arraycopy로 연결하여 int[] 형으로 반환하는 방식으로 구현할 수 ㅣㅇㅆ다.

해당 방식으로 재구성을 하게되면 메모리가 자동으로 이루어지므로 C언어처럼 free를 호출할 필요가 없고 수동으로 for문을 돌리는 것보다 빠르지만 메모리 재할당이 필요한 경우 새로운 배열을 생성해야 하므로 성능이 다소 저하될 수 있다는 단점 또한 존재한다.

특징	C 언어	Java
배열 접근 방식	두 배열을 받아 새 배열을 생성 후 반환	기존 배열에 새로운 배열을 추가, System.arraycopy 사용
배열 메모리 관리	수동 메모리 관리 (malloc, free) 필요	자동 메모리 관리 (가비지 컬렉터)
배열 할당	malloc을 통해 배열 동적 할당, 크기 조정 불편	new 키워드를 사용하여 배열 동적 할당, 크기 조정 가능
배열 재할당	새로운 배열 생성 후 병합, 이전 배열을 free 해야 함	배열 크기 재조정 시 새로운 배열을 생성, 자동 메모리 관리

printArr 메서드

```
1 usage
void printArr(){
    for(int i : arr){
        System.out.print(i + " ");
    }
    System.out.println();
}
```

인자를 받지 않고도 class 내의 변수를 활용하여 출력하는 것이 가능하다. 마찬가지로 for-each 구문을 활용하여 배열을 순회하며 출력하게 된다.

생성자

```
12 usages
int[] arr;

1 usage
numarray(int[] inputArr){
    arr = new int[inputArr.length];
    System.arraycopy(inputArr, srcPos: 0, arr, destPos: 0, inputArr.length);
}
```

Class에 필요한 변수를 선언하고 생성자를 활용해서 array를 만드는 것이 가능하다. 위와 같은 방식으로 입력할 inputArr를 받은 이후, arrayCopy를 활용하여 클래스 내의 arr라는 배열의 내용을 채워넣는 방식으로 생성자를 활용하고자 하였다.

배열을 객체의 속성으로 관리하므로 여러 메서드에서 동일한 배열을 사용할 수 있다는 장점이 존재한다. 또한 C언어의 구조체와는 다르게 java에서는 클래스를 설계하여 위와 같이 skxkof 수 있으며, 단순 배열 조작을 위해 객체를 생성하는 오버헤드가 발생할 수 있다는 단점을 꼽을 수 있다.

특징	C 언어	Java
배열 관리 방식	구조체를 사용하여 배열을	클래스를 사용하여 배열을

	관리 (배열을 구조체 내 속성으로 선언)	객체의 속성으로 선언
배열 접근 방식	구조체를 통해 배열에 접근	객체의 메서드를 통해 배열에 접근
배열 생성	배열 크기 및 메모리 관리 직접 처리 (malloc 사용)	생성자를 사용하여 배열을 동적으로 할당 (new 키워드 사용)
배열 공유	배열을 여러 함수에서 공유하려면 포인터로 전달	클래스 내 배열을 여러 메서드에서 공유 가능
오버헤드	배열 관리가 단순하지만, 배열 크기 변경 불편	클래스 설계로 객체를 생성해야 하므로 오버헤드 발생
장점	구조체를 통한 간단한 배열 관리	객체지향적인 설계로 배열을 여러 메서드에서 공유 가능
단점	배열 크기 변경이 불편하고, 수동 메모리 관리 필요	객체 생성에 따른 오버헤드와 성능 저하 가능

소스 프로그램

Numarray.java

```
package org.example;

public class numarray {

    int[] arr;

    numarray(int[] inputArr){
        arr = new int[inputArr.length];
        System.arraycopy(inputArr, 0, arr, 0, inputArr.length); // inputArr에서 arr로 값을 복사
    }

    int findMax(){
        int max = arr[0];
        for(int i = 1 ; i < arr.length; i++){
```

```
            if(arr[i] > max){
                max = arr[i];
            }
        }
        return max;
    }

    float getAvg(){
        int sum = 0;
        for(int i : arr){
            sum += i;
        }
        return (float) sum / arr.length;
    }

    int[] mergeArrays(int[] arr1) {
        int[] newArr = new int[arr.length + arr1.length];
        System.arraycopy(arr, 0, newArr, 0, arr.length);
        System.arraycopy(arr1, 0, newArr, arr.length, arr1.length);
        return newArr;
    }


    void printArr(){
        for(int i : arr){
            System.out.print(i + " ");
        }
        System.out.println();
    }

}
```

testNumArray.java

```
public class TestNumArray{
```

```
    public static void main(String[] args) {
        int[] arr1 = {5,2,7,6};
        int[] arr2 = {2,4,6};

        numarray na = new numarray(arr1);

        int[] newArray = na.mergeArrays(arr2);
        numarray na2 = new numarray(newArray);
        na2.printArr();

        System.out.println(na.findMax());
        System.out.println(na.getAvg());

    }

}
```

평 가 표

평가 항목	학생 자체 평가 (리포트 해당 부분 표시 및 간단한 의견)	평가 (빈칸)	점수 (빈칸)
Java Style - 동작? Java 스타일로 작성했을 때 동작? - 요구 사항 만족? 즉, 진정한 Java 스타일?	자바 코드 동작O, 정상 결과 출력 O. • class 별도생성 O, 생성자 및 메서드 encapsulation.		
별도의 class에서 main()으로 테스트?	별도 class 테스트 O.		
두 방법 비교 - 평가 항목은? - 한눈에 보이게?	배열크기, 접근방식, 유연성, 메모리관리, 오류가능성, 배열 할당 등에 근거하여 표로 작성.		
기타			
총평/계			

* 학생 자체 평가는 점수에 반영되지 않음.
* 학생 스스로 자신의 보고서를 평가하면서, 체계적으로 프로젝트를 마무리하도록 유도하는 것이 목적임.