배열에서 Stack과 Heap

```java
//stack에 정적 할당
int arr1[] = {2, 3, 4, 5, 6, 7};

//Heap에 동적 할당
int dynamicArr[] = new int[30];
```

Stack의장점 :

```java
//다만 Stack의 경우 속도면에서 빠르다.
//이유 : 정적 할당은 미리 설정이 되있기때문에 동적할당보다 속도가 빠르다.
//Stack의 경우 공간 제약이있었지만
```

Heap의장점 :

```java
//Heap 방식의 경우 필요한만큼 할당할 수 있어서 유연한 접근이 가능하다.
//동적 할당은 모두 프로그램 실행 도중 생성하는 것을 의미
//주의 index는 0~29까지이다.
```

Heap 영역에 할당 예제

```java
Scanner sc = new Scanner(System.in);
System.out.print("학급에 학생이 명 몇 있습니까?");

int StudentNum = sc.nextInt();
int StudentArr[] = new int[StudentNum];

for (int i = 0; i < StudentNum; i++) {
    StudentArr[i] = (int) (Math.random() * 21) + 80;
    System.out.printf("studentArr[%d] = %d\n", i , StudentArr[i]);
}
```

출력 값

```
학급에 학생이 명 몇 있습니까?3
studentArr[0] = 94
studentArr[1] = 91
studentArr[2] = 95
```

Challenge 문제

문제 : 피보나치 수열

```java
// 피보나치수열
// 사용자가 15를 입력하면 15번째 값을, 8을 입력하면 8번째 값을 구하도록 프로그래밍
System.out.println("사용자로부터 n을 입력받아 n 번째 피보나치 수열의 항을 구합니다.");

Scanner scan = new Scanner(System.in);
System.out.print("n 값을 입력하시오: ");

int num = scan.nextInt();
int res = 0;

if (num <= 0) {
    System.out.println("0번째 항 혹은 음수 항은 존재하지 않습니다.");
} else if (num < 3) {
    System.out.println("당신이 찾는 값은 1입니다.");
} else {
    int first = 1, second = 1;
    for (int i = 0; i < num - 2; i++) {
        res = first + second;
        first = second;
        second = res;
    }

} System.out.println("결과는 = " + res);
```

예외 값 출력

예외 값 출력

원하는 값 출력

-2를 한 이유는 first,second의 값을 알고 시작했기때문이다.

출력 값

```
사용자로부터 n을 입력받아 n 번째 피보나치 수열의 항을 구합니다.
n 값을 입력하시오: 10
결과는 = 55
```

배열 문제

문제 : 아래와 같은 형태의 숫자들이 있다.
1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, ...
n을 입력해서 n번째 값을 구하도록 프로그래밍 해보자!

풀이 1

```java
public static void main (String[]args){
    System.out.println("1, 2, 4, 8, ... 1024, ...");

    Scanner scan = new Scanner(System.in);

    System.out.print("몇 번째 항을 구할까요 ? ");
    int num = scan.nextInt();

    if (num <= 0) {
        System.out.println("잘못된 값을 입력하였습니다.");
    } else if (num < 2) {
        System.out.println("당신이 찾는 값은 1입니다.");
    } else {
        int numArr[] = new int[num];

        numArr[0] = 1;

        for (int i = 1; i < numArr.length; i++) {
            numArr[i] = numArr[i - 1] * 2;
        }

        System.out.printf("%d 번째 항은 = %d\n", num, numArr[num - 1]);
    }
```

곱하기를 통한 계산

풀이 2

```java
    for (int i = 1; i < numArr.length; i++) {
        // Math.pow(A, B)는 A^B(A의 B승)을 계산한다.
        // Math.pow는 double을 결과로 내놓기 때문에 강제로 int 타입으로 변형하였음
        numArr[i] = (int)Math.pow(2, i);
    }
```

제곱 함수를 통한계산

풀이3

```java
for (int i = 0; i < Arr.length; i++) {
    Arr[i] = 1<<i;
    System.out.println(Arr[i]);

}
}
```

쉬프트 연산을 통한 계산

Class 생성

클래스 생성

```java
class Person {
    int age;
    String name;

}
// new Person()을 통해 만든 공간은
// 위의 커스텀 데이터타입에 해당하는 정보들을 저장할 수 있는 공간을 생성한 것이다.
```

클래스 사용

새로운 개념 new를 통해 공간을 만드는 방법

1. new 를 적는다.
2. 데이터타입을 적는다.
3. 만약 데이터타입이 배열이라면 대괄호를 열고 몇 개를 만들지 적는다.
   만약 데이터타입이 클래스라면 소괄호를 열고 닫은후 필요하다면 인자를 설정한다.

클래스 사용

```java
public class ClassTest {
    public static void main(String[] args) {
        // 클래스는 사용자가 직접 만들 수 있는 데이터타입(커스텀 가능)
        // 변수를 만드는것과 동일하게 클래스를 사용해서 변수를 만들자
        // 변수 이름 person으로 Person 형태의 텅 빈 공간이 만들어진다(만들어지는 위치는 Heap)
        Person person = new Person();

        person.age = 21;
        // person 변수가 가지고 있는 공간중 name에 "안녕"을 저장한다.
        person.name = "안녕";

        System.out.println("이 사람은 몇살 ? " + person.age);
        System.out.println("얘 이름은  ? " + person.name);

    }
}
```

출력 값

```
이 사람은 몇살 ? 21
얘 이름은  ? 안녕
```

Class Method 생성

```java
class Teacher {
    int age;

    int getAge() {
        return age;
    }
    void setAge(int age) {
        this.age = age;

    }
}
public class ClassMethodTest {
    public static void main(String[] args) {
        Teacher t = new Teacher();

        // 클래스 내부의 값을 Setter로 설정하고
        t.setAge(40);   // 40이 입력

        // 설정된 값을 Getter를 통해 얻는다.
        System.out.printf("나이는 : " + t.getAge());

        // t.name, t.major, t.age 로 출력해도 보입니다.
    }
}
```

매서드를 만드는 방법

1. 먼저 리턴(return) 타입을 작성한다.
2. 매서드의 이름을 작성한다(용도에 맞게 작성한다)
   보통 Getter의 경우 값을 얻고자 할 때(즉 return 용도로 사용)
   Setter의 경우 값을 설정하고자 할 때 사용한다.
3. 소괄호 내부에 인자로 입력 받을 매개변수를 설정한다.
4. 중괄호 내부에 해당 매서드(기능)이 수행할 업무를 작성한다.

주의 : 클래스 작성할 때는 이니셜마다 대문자를 붙였다.
매서드는 시작은 소문자 그 이후부터의 이니셜은 대문자
소괄호 내부는 인자가 배치되는데 텅 비어 있는 것은 인자(입력)이 없다는 뜻이다.

출력 값

```
나이는 : 40
```

결론 : 클래스 내부에 값을 설정 Setter, 설정값을 가져오는 Getter

멍멍이클래스, 고양이클래스 만들기

```java
public class DogClass {
    String name;
    String dogBreed;
    int age;

    String getName() {
        return name;
    }
    void setName(String name) {
        this.name = name;
    }
     String getDogBreed() {
        return dogBreed;
    }
    void setDogBreed(String dogBreed) {
        this.dogBreed = dogBreed;
    }
    int getAge() {
        return age;
    }
    void setAge(int age) {
        this.age = age;
    }
}
```

```java
public class CatClass {
    String name;
    String catBreed;
    int age;

    String getName() {
        return name;
    }
    void setName(String name) {
        this.name = name;
    }
    String getCatBreed() {
        return catBreed;
    }
    void setCatBreed(String catBreed) {
        this.catBreed = catBreed;
    }
    int getAge() {
        return age;
    }
    void setAge(int age) {
        this.age = age;
    }
}
```