

과제 3 - 1

```
public static void bark()
{
    System.out.println("Bow Wow!");
}

public static void bark(int n)
{
    for (int i = 0; i < n; i++) {
        System.out.println("Bow Wow!");
    }
}

public static void bark(String s)
{
    System.out.println("Grr... " + s + "!!");
}

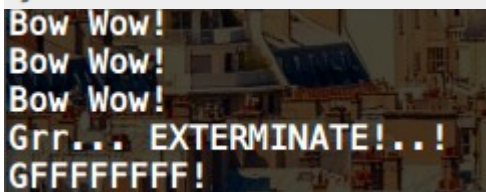
public static void bark(char c)
{
    System.out.print("G");
    for (int i = 0; i < 8; i++) {
        System.out.print(c);
    }
    System.out.println("!");
}
```

Dog클래스의 인스턴스를 생성하지 않고 Dog.bark();로 메소드를 사용 할 수 있도록 모든 메소드를 static으로 선언하였다.

bark()는 BowWow를 출력하고, bark(int) 는 BowWoW를 int 횟수 만큼 출력한다.

bark(String)은 Grr + string + !를, bark(char)는 G후에 8번동안 문자를 반복한 뒤에 느낌표를 붙인다.

```
public static void main (String[] args) {
    Dog.bark(3);
    Dog.bark("EXTERMINATE!");
    Dog.bark('F');
}
```



Bow Wow!
Bow Wow!
Bow Wow!
Grr... EXTERMINATE!!
GFFFFFFFF!

The screenshot shows the output of the Java program. It displays four lines of text: 'Bow Wow!', 'Bow Wow!', 'Bow Wow!', and 'Grr... EXTERMINATE!!'. The final line is 'GFFFFFFFF!', which is the output of the bark(char) method where 'F' is repeated 8 times after a 'G' and followed by an exclamation mark.

과제 3-2

```
private static class MyStack<T>
{
    private ArrayList<T> stack;
    private int count;

    public MyStack(int initSize)
    {
        stack = new ArrayList<>(initSize);
        count = 0;
    }

    public void push(T obj)
    { //push the object to stack, and increase count, size will be automatically managed
        stack.add(count, obj);
        count++;
    }

    public T pop() throws IndexOutOfBoundsException
    { //pops the top object which is managed by count, decrease 1
        T item = null;
        // -1 from count because ArrayList index starts from 0
        count--;
        try {
            item = stack.remove(count);
        } catch (IndexOutOfBoundsException e) {
            System.out.println("Stack is already empty: " + e.toString());
        }
        return item;
    }

    public T peek() throws IndexOutOfBoundsException
    { //doesn't pop
        T item = null;
        try {
            //temporary -1 from count because ArrayList index starts from 0
            item = stack.get(count-1);
        } catch (IndexOutOfBoundsException e) {
            System.out.println("Stack is already empty: " + e.toString());
        }
        return item;
    }
}
```

ArrayList<T>를 이용해 제네릭하게 구현된 MyStack<T>의 코드 중 일부이다. MyStack은 ArrayList<T> stack, int count를 필드로 가지고 있다. count는 현재 스택에 있는 아이템의 수를 의미한다. 스크린샷에는 메소드의 일부인 push, pop, peek이 구현되어있으며 push의 경우에는 ArrayList에 아이템을 추가하여 최대 크기를 넘어설 경우, 자동적으로 추가적인 메모리가 확보가 되므로 크기 관련하여 구현할 것이 없다.

pop과 peek은 둘 다 stack의 맨 위에 있는 아이템을 반환하지만, pop은 그 아이템을 스택에서 제거한다는 차이가 있다. 비어있는 스택에서 pop이나 peek을 하면 안되므로, IndexOutOfBoundsException을 발생하도록 하였다.

```
@Override
public String toString()
{
    String result = "";
    for (int i = 0; i < count; i++) {
        result += stack.get(i);
    }
    return result;
}
```

또한, MyStack의 toString을 구현하여 출력을 용이하도록 하였다.

```
//remove spaces
test = test.replaceAll("\\s+", "");

try (Writer writer = new BufferedWriter(new OutputStreamWriter(
    new FileOutputStream("stack.txt"), "utf-8")))
{
    for (int i = 0; i < test.length(); i++) {
        if (test.charAt(i) == '+') {
            //if +, read the next char to add
            i++;
            writer.write("[push] "+ test.charAt(i)+"\n");
            stk.push(test.charAt(i));
        }
        else if (test.charAt(i) == '-') {
            writer.write("[pop] " + stk.pop()+"\n");
        }
    }
    writer.write(stk.toString());
} catch (IOException ex) {
    System.out.println("Unable to Write to File");
}
```

main함수의 일부분이다. 우선 처리를 편하게 하기 위해서 입력에서 공백을 전부 제거하였다. 그 후, Writer가 stack.txt에 utf-8로 파일쓰기를 하도록 하였다. for문으로 문자열의 끝까지 읽으며, + 가 나오면 i를 하나 증가시켜 +다음에 나오는 문자를 스택에 푸시하도록 하고, -가 나오면 스택에서 하나 pop하도록 한다. 문자열을 다 읽었다면, 비록 과제의 요구사항에는 없었지만 스택이 잘 구현됐는지 확인하기 위해 stk.toString()을 파일에 작성하여준다. Writer.close()는 후에 추가하였지만, 스크린샷에서는 없다.

```
[push] C
[push] o
[push] m
[push] p
[push] u
[push] t
[push] e
[pop] e
[push] r
[pop] r
[push] P
[push] r
[push] o
[push] g
[pop] g
[push] r
[push] a
[push] m
[push] 1
[push] 5
ComputProram15
```

결과 파일 stack.txt

과제 3-3

```
public MyInt(int n)
{
    num = n;
}

@Override
public boolean equals(Object obj)
{
    if (this == obj) {
        return true;
    }
    if (obj == null) {
        return false;
    }
    if (!(obj instanceof MyInt)) {
        return false;
    }
    MyInt other = (MyInt) obj;
    if (num != other.num) {
        return false;
    }
    return true;
}
```

MyInt(int) 생성자로 num field를 n값으로 생성하도록 하였다. 그리고 equals를 오버라이드하였다.

```
public static void main (String[] args)
{
    MyInt[] a1, a2;
    int length = 10;
    a1 = new MyInt[length];
    a2 = new MyInt[length];

    for (int i = 0; i < length; i++) {
        a1[i] = new MyInt(i);
        a2[i] = new MyInt(i);
    }
    System.out.println(Arrays.equals(a1,a2));
}
```

main함수에서, for문으로 배열 a1,a2의 MyInt를 동일하게 생성하는 코드이다. 그리고 equals의 결과값을 출력하도록 하였다.

false

equals를 주석처리 하였을 때 출력값.

true

equals의 주석을 해제하였을 때의 출력값.

과제 3-4


```

public class Ass4 {
    final static String VOWEL = "aeiou";

    public static void main (String[] args)
    {
        char c;
        Random rand = new Random(); //generate random instance

        //+1 to make 'z' inclusive //'a' ~ 'z'
        c = (char) (rand.nextInt('z' - 'a' + 1) + (int)'a');

        System.out.print(c + " is a ");
        //if char c is included in the String VOWEL
        //append "" to make c a String
        System.out.println(VOWEL.contains(c+"") ? "vowel." : "consonant.");
    }
}

```

알파벳 소문자를 임의로 생성하고, 문자열 VOWEL = "aeiou"에 포함되어 있다면 vowel을, 아니면 consonant를 출력하는 코드이다.

contains 메소드는 인자로 문자열을 받기 때문에, c + ""를 통해 문자를 문자열로 바꿔주었다.

문자열로 바꿔줄 필요 없이 VOWEL.indexOf(c)의 값이 -1(문자열에 없다)이라면 consonant, 그 외의 값(0~4)이라면 vowel을 출력해도 되지만, contains를 사용하는 게 이해하기 쉽고, boolean값이므로 boolean ? A : B를 사용할 수 있어서 문자열로 바꿔주는 것을 감수하고 contains함수를 사용하였다.

```

z is a consonant.
a is a vowel.
i is a vowel.
e is a vowel.
o is a vowel.
u is a vowel.

```

```

s is a consonant.

```

계속하려면 엔터 혹은 명령을 입력하십시오
p is a consonant.

과제 3-5

```

public int getNext()
{
    //append to the sequence
    int length = this.getLength();

    if (length <= 1) {
        fibonacci.add(1);
    }
    else {
        fibonacci.add(fibonacci.get(length-1) + fibonacci.get(length-2));
    }
    length++;
    return fibonacci.get(length-1);
}

```

피보나치 수열을 한 숫자씩 추가하는 함수이다. 현재 수열의 길이가 0이거나 1이라면 배열 fibonacci(ArrayList)에 1을 추가하고, 그 이상부터는 뒤에서 두 개의 값을 더한 값을 ArrayList.add()로 추가한다. 그 후, 추가된 값을 반환한다.

```

public FibonacciSeq(int len)
{
    fibonacci = new ArrayList<>(len);
    for (int i = 0; i < len; i++) {
        getNext();
    }
}

```

생성자는 정수를 받아서, 그 길이만큼 getNext()함수를 통하여 피보나치수열을 생성한다. 길이만큼 ArrayList를 생성하지 않더라도 자동적으로 길이가 추가되지만, 미리 길이만큼 공간을 확보하여 그러한 오버헤드를 줄인다.

```

@Override
public String toString()
{
    String fibo = fibonacci.toString();
    //remove start and end brackets([, ])
    return fibo.substring(1, fibo.length()-1);
}

```

수열의 출력을 위해 override한 toString함수이다. ArrayList의 toString은 문자열 [1,1,2,3]을 반환하므로 [] 를 substring()으로 제거한 함수를 반환한다.

과제의 요구사항은 아니었지만, 추가로 구현한 함수들이다.

```

public void setLength(int newLen)
{
    //similar to get but changes array
    int length = this.getLength();

    for (int i = 0; i < newLen - length; i++) {
        this.getNext();
    }
    fibonacci = fibonacci.subList(0, newLen);

    return;
}

```

생성한 피보나치수열의 길이를 나중에 바꾸고 싶을 때 사용하는 함수이다.

새로운 길이가 현재의 길이보다 크다면, getNext()를 통해 수열을 추가한다. 만약 새로운 길이가 현재의 길이보다 짧다면, for문에서는 아무 일도 일어나지 않는다.

그 후, fibonacci의 처음부터 새로운 길이까지의 subList를 fibonacci 배열로 한다.

만약 newLen이 현재의 길이보다 짧다면 fibonacci는 짧은 수열이, 길다면 for문으로 인해 길어진 수열로 바뀔 것이다.

```

public int get(int index) throws IndexOutOfBoundsException
{
    //if index > length calculate to get value(but doesn't change the array)
    int result = 0;
    int a, tmp = 0;
    int length = this.getLength();

    //if index <= length just bring the value instead of calculation.
    if (index <= length) {
        try {
            result = fibonacci.get(index-1);
        } catch (IndexOutOfBoundsException e) {
            System.out.println("ERROR: " + e.toString());
        }
    }
    else {
        //result is last number of the sequence
        a = fibonacci.get(length-2);
        result = fibonacci.get(length-1);

        for (int i = 0; i < index - length; i++) {
            //hold a's value to add to result;
            tmp = a;
            //previous number is updated to the last number
            a = result;
            //next number(result) is last number of sequence + previous number
            result += tmp;
        }
    }
    return result;
}

```

setLength 나 생성자가 배열의 길이를 바꿔 수열의 값을 구한다면, get함수는 수열자체의 값은 바꾸지 않고 단순히 덧셈연산을 통해서 그 위치에 있는 피보나치 수의 값을 구하는 함수이다. 단순히 피보나치 수열의 특정 위치의 값을 한 번 참조해야 한다면 get을 사용하는 것이 생성자나 setLength로 그 위치까지 구하는 것보다 효율적일 것이다.

get의 인자인 index가 현재 수열의 길이보다 짧거나 같다면, 현재 수열의 값을 참조하여 값을 반환한다. index가 현재 수열의 길이보다 길다면, 수열의 마지막 값과 그 전 값을 가져와서 계속 더해가며 해당 index의 피보나치 수열 값을 계산한 뒤에, 반환한다.

```

public static void main (String[] args) {
    int length = Integer.parseInt(args[0]);
    FibonacciSeq fib = new FibonacciSeq(length);
    System.out.println(fib);
    System.out.println(fib.get(15));
    System.out.println(fib);
    fib.setLength(11);
    System.out.println(fib);
    System.out.println(fib.get(10));
}

```

메인 함수이다. 입력받은 개수로 처음 피보나치 수열을 생성한 다음, 출력한다. 그 후에, get(15)를 통해 15번째 수를 출력하지만 println(fib)을 하게 되면 수열을 여전히 처음에 생성한 대로이다. 그 후에 setLength(11)을 하게 되면 fib수열은 11개짜리로 바뀌고, get(10)을 하게 되면 값의 계산이 아닌, fibonacci 배열의 10번째 원소를 참조하여 값을 반환한다.