6월 7일 복습

이태양


```java
public class PerformanceUtil {
    public static long startTime;
    public static long estimatedTime;

    public static void performanceCheckStart() {
        // 현재 시간을 밀리초 단위로 구해서 startTime에 저장
        startTime = System.currentTimeMillis();
    }

    public static void performanceCheckEnd() {
        // 현재 시간을 밀리초 단위로 구하고 startTime 을 뺌
        estimatedTime = System.currentTimeMillis() - startTime;
    }

    // 값을 가져오는 단위가 ms(1 / 1000 초)
    public static void printPerformance() {
        System.out.println(
                "걸린 시간: " + estimatedTime / 1000.0 + " s");
    }
}
```

현재 시간을 저장하고

또 현재 시간을 저장해서

걸린 시간을 구해서 알아보는 예제


```java
public class SequencePerformanceTest {
    final static int ZERO = 0;
    final static int END5 = 1000000000;
    final static int START5 = 800000001;
    final static int END4 = 800000000;
    final static int START4 = 600000001;
    final static int END3 = 600000000;
    final static int START3 = 400000001;
    final static int END2 = 400000000;
    final static int START2 = 200000001;
    final static int END = 200000000;
    final static int START = 1;
    final static double COEFFICIENT = Math.pow(10, -15);
    final static double DEG2RAD = 180.0;

    public static void main(String[] args) {
        double sum = ZERO;
        double sum2 = ZERO;
        double sum3 = ZERO;
        double sum4 = ZERO;
        double sum5 = ZERO;

        PerformanceUtil.performanceCheckStart();

        for(int i = START; i <= END; i++) {
            // Math.sin() 자체가 연산량이 많다.
            // 그래서 스레드를 활용했을때와 활용하지 않았을때의 차이를 보기에 좋다.
            // 삼각함수 연산을 도입해서 스레드 없이 순차 처리하였을때
            // 얼마나 오래걸리는지를 체크해보고
            // 실제 스레드를 도입했을때 성능이 대폭 상승하는 것을 보면 됩니다.
            sum += (i * (COEFFICIENT * i)) * Math.sin(i * Math.PI / DEG2RAD);
        }
        for(int i = START2; i <= END2; i++) {
            sum2 += (i * (COEFFICIENT * i)) * Math.sin(i * Math.PI / DEG2RAD);
        }
        for(int i = START3; i <= END3; i++) {
            sum3 += (i * (COEFFICIENT * i)) * Math.sin(i * Math.PI / DEG2RAD);
        }
        for(int i = START4; i <= END4; i++) {
            sum4 += (i * (COEFFICIENT * i)) * Math.sin(i * Math.PI / DEG2RAD);
        }
        for(int i = START5; i <= END5; i++) {
            sum5 += (i * (COEFFICIENT * i)) * Math.sin(i * Math.PI / DEG2RAD);
        }
        System.out.println("sum = " + sum);
        System.out.println("sum2 = " + sum2);
        System.out.println("sum3 = " + sum3);
        System.out.println("sum4 = " + sum4);
        System.out.println("sum5 = " + sum5);

        PerformanceUtil.performanceCheckEnd();

        //System.out.println("sum = " + sum);

        PerformanceUtil.printPerformance();
```

순차처리할 시
얼마나 걸리는지
확인해보기 위한 예제


```java
import java.net.MalformedURLException;
import java.net.URL;

public class NetWorkUrlTest {
    // Malform 이라는것이 악성 코드에 해당해서
    // 이상한 URL로 링크를 태워서 공격을 할 수 있기 때문에 그것에 대한 방어 조치라 보면 됨
    // www.daum.net, http://www.daum.net
    // URL을 반드시 후자로 줘야 합니다.
    // 이유는 www.daum.net 으로 하면 위와 같이 악성코드 공격이 가능함
    public static void main(String[] args) throws MalformedURLException {
        URL myURL = new URL( spec: "http://www.loanconsultant.or.kr/source/index.jsp?t=20191216");

        // Protocol: HTTP(웹 애플리케이션 전용 프로토콜입니다)
        System.out.println("Protocol = " + myURL.getProtocol());
        System.out.println("authority = " + myURL.getAuthority());
        System.out.println("host = " + myURL.getHost());
        System.out.println("port = " + myURL.getPort());
        System.out.println("path = " + myURL.getPath());
        System.out.println("query = " + myURL.getQuery());
        System.out.println("filename = " + myURL.getFile());
        System.out.println("ref = " + myURL.getRef());
    }
}

// int[] test = { 2400, 5000, 123, 123245, 23542345648, 234923, 23492334, 2349 ..... }
// 풀이 방식: 1. HashSet, 2. 라이브러리 없이 통으로 만들기, 3. 다른 Collection 사용해서
```

네트워크 첫 코딩예제

설명을 듣지못해서 눈으로만 한번 읽어보았습니다


```java
import java.util.*;

public class SortingTest {
    public static void main(String[] args) {
        String[] sample = {"I", "walk", "the", "line", "Apple", "hit", "me", "Ground", "attack", "you"};

        List<String> list = Arrays.asList(sample);

        // 정렬 법칙(대문자 우선, 그 다음 소문자)
        Collections.sort(list);
        System.out.println(list);
        Integer[] numbers = {1, 2, 3, 100, 77, 2342, 2342354, 345, 12323, 12, 4};
        List<Integer> numList = Arrays.asList(numbers);
        Collections.sort(numList);
        System.out.println(numList);
        Set fruits = new HashSet();
        fruits.add("strawberry");
        fruits.add("watermelon");
        fruits.add("grape");
        fruits.add("orange");
        fruits.add("apple");
        fruits.add("banana");

        List fruitsList = new ArrayList(fruits);
        fruitsList.add("ofcourse");

        Collections.sort(fruitsList);

        System.out.println(fruitsList);
    }
}
```

Sort를 사용하여 정렬하는것을 보여주는 예제

대문자 우선 그다음 소문자


```java
class FrequencyChecker{
    Set<Integer> frequencySet;
    Map<Integer, Integer> frequencyMap;
    int[] backUp;

    public FrequencyChecker (int[] arr){
        frequencySet = new HashSet<Integer>();
        frequencyMap = new HashMap<Integer,Integer>();
        backUp = arr;


        for(Integer elem : arr){
            frequencySet.add(elem);
            frequencyMap.put(elem, 0);
        }
        System.out.println("frequencySet : "+frequencySet);
        System.out.println("frequencyMap : " + frequencyMap);
    }

    public Map<Integer, Integer> getFrequencyMap() {
        return frequencyMap;
    }

    public void allocRandomFrequency(int num){
        for(int i = 0; i < num; i++){
            int tmp = (int)(Math.random()*10);
            int key = backUp[tmp];
            if(frequencySet.contains(key)){
                int cnt = frequencyMap.get(key);
                frequencyMap.put(key, ++cnt);
            }
```

인티저형 해시셋과 해시맵을 선언해주고
백업할 배열을 선언해줌

포이치구문으로 arr의 내용을 elem에 넘겨주고
그 내용을 해시셋과 해시맵에 넘겨준다

게터 프리퀀시맵의 값을 반환

Tmp는 0~9까지의 번호를 랜덤으로 받는다
나온번호를 인덱스번호로하여 값을 키에 저장한다


```java
public class Prob60 {
    public static void main(String[] args) {
        int[] testSet = {2400, 5000, 1000, 200, 6000, 7700, 8600, 42, 2, 13};

        FrequencyChecker fc = new FrequencyChecker(testSet);

        fc.allocRandomFrequency( num: 1000);

        System.out.println(fc);
        System.out.println(fc.getFrequencyMap());
    }
}
```

객체 생성 후

값을 입력해서

랜덤으로 발생시켜 중복이 몇번이 되는지 확인할 수 있다.

```
frequencySet : [2400, 6000, 2, 7700, 5000, 1000, 200, 8600, 42, 13]
frequencyMap : {2400=0, 6000=0, 2=0, 7700=0, 5000=0, 1000=0, 200=0, 8600=0, 42=0, 13=0}
FrequencyChecker@4554617c
{2400=101, 6000=94, 2=104, 7700=106, 5000=104, 1000=103, 200=98, 8600=95, 42=96, 13=99}
```