[디지털 컨버전스]
스마트 콘텐츠와 웹 융합 응용SW 개발자 양성과정

강사 : 이상훈
학생 : 임초롱

21회차

Collections.sort( ) : 정렬

링크 https://github.com/limcholong/LectureContents/blob/main/java/CholongLim/Day21/src/SortingTest.java

```java
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);
```

Collections.sort( ) :

Collections.sort( ) 의 매서드를 이용하여 정렬할 수 있다.

1. String :

```
[Apple, Ground, I, attack, hit, line, me, the, walk, you]
```

알파벳 순으로 정리가 되나, 대문자가 우선으로 정렬된다.

2. Integer :

```
[1, 2, 3, 4, 12, 77, 100, 345, 2342, 12323, 2342354]
```

작은 수부터 정렬된다.

➔ Add로 추가 가능한가?

```
[apple, banana, grape, ofcourse, orange, strawberry, watermelon]
```

가능하다.
add로 추가한 ofcourse가 추가되어 정렬되었다.

Thread

링크 https://github.com/limcholong/LectureContents/blob/main/java/CholongLim/Day21/src/AsynchronousPatternDoc.java

Thread 주요 개념

1. Thread를 활용하는 이유는 성능을 빠르게 만들기 위해서이다.
2. 비동기 패턴 (Asynchronous Pattern) 이란 전부 Thread를 기반으로 한다.
3. 자바 스크립트 또한 Multi Thread 모델을 자체적으로 지원한다.
4. Thread를 사용할 때는 Critical Section에 대한 방어가 중요하다. (데이터의 무결성)
5. 또한 Thread는 비동기 처리를 하기 때문에 데이터의 완전한 전송을 보장하지 못할 수 있다.

동기 처리와 비동기 처리의 차이점은 무엇일까?

Ex1 전화통화 = 동기처리

Why?

친구한테 전화를 걸었다. 친구가 통화 허용을 하지않으면 통화가 안된다.

Ex2 카카오톡 메시지 = 비동기 처리

Why?

상대방이 확인을 하던 하지않던 메시지는 보내진다.

(Thread는 비동기 처리를 지원하지만, Lock를 쓰는 그 순간은 동기처리이다.)

네트워크 (HTTP / HTTPS)

링크 https://github.com/limcholong/LectureContents/blob/main/java/CholongLim/Day21/src/NetworkUrlTest.java

HTTP (Hyper Text Transfer Protocol)

HTTP (Hyper Text Transfer Protocol)이란 서버 / 클라이언트 모델에 따라 데이터를 주고 받기 위한 프로토콜
즉, HTTP는 인터넷에서 하이퍼텍스트를 교환하기 위한 통신 규약으로 80번 포트를 사용하고 있다.
따라서 HTTP 서버가 80번 포트에서 요청을 기다리고 있으며, 클라이언트는 80번 포트로 요청을 보내게 된다.

HTTP는 암호화가 되지 않은 평문 데이터를 전송하는 프로토콜,
때문에 HTTP로 비밀번호나 주민등록번호 등을 주고 받으면 제 3자가 정보를 조회할 수 있었다.
이러한 문제를 해결하기 위해 HTTPS가 등장하게 되었다.

HTTPS (Hyper Text Transfer Protocol Secure)

Hyper Text Transfer Protocol over Secure Socket Layer, HTTP over TLS, HTTP over SSL, HTTP Secure
등으로 불리는 HTTPS는 HTTP에 데이터 암호화가 추가된 프로토콜이다.
HTTPS는 HTTP와 다르게 433번 포트를 사용하며,
네트워크 상에서 중간에 제 3자가 정보를 볼 수 없도록 공개키 암호화를 지원하고 있다.

60번 문제

링크 https://github.com/limcholong/LectureContents/blob/main/java/CholongLim/Day21/src/Quiz60Explain.java

```java
// 숫자값 2400, 5000, 그리고 아무 숫자나 8개 정도 추가한다.
// 이 난수들(총 8개)을 가지고 1000개의 데이터를 마구잡이로 생성한다.
// 각각의 데이터들이 몇 개씩 중복 되었는지 프로그래밍 해보도록 하자!
// 그리고 이 정보들을 정렬해보자!

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;
        //backUp =  { 2400,  5000, 1000, 200, 6000,
        //                77000, 434,  768,  20,  50 }

        for (Integer elem : arr) {
            frequencySet.add(elem);
            frequencyMap.put(elem, 0);
            // Set과 Map 각각에 arr값(elem) 셋팅
            // frequencyMap을 통해 cnt할 것, 초기 값 = 노카운트 = 0
        }

        System.out.println("frequencySet: " + frequencySet);
        System.out.println("frequencyMap: " + frequencyMap);
    }

    public void allocRandomFrequency (int num) {
//          int num  : 몇개의 데이터를 할 것인지
```

```java
    public void allocRandomFrequency (int num) {
//          int num  : 몇개의 데이터를 할 것인지
        for (int i = 0; i < num; i++) {
            int tmp = (int) (Math.random() * 10);
            // 0 ~ 9 (10개)
            int key = backUp[tmp];
            //int backUp[] = { 2400 , ...} stack 배열
            //int backUp[tmp] 하여 값을 랜덤화

            System.out.printf("%6d", key);

            if (i % 20 == 19) {
                // 20으로 나눴을때 19개가 남는 i번째에서 enter,
                // i가 0부터 시작이므로 20개씩 마다 enter
                System.out.println();
            }

            if (frequencySet.contains(key)) {
                // contains ( ) :  HashSet 내의 객체가 존재하는지 확인하는 코드
                // frequencySet 내의 특정 랜덤값이 존재한다면 (if)
                int cnt = frequencyMap.get(key);
                // get ( ) : key에 해당하는 value를 리턴해준다.
//                          key가 존재하지 않으면 null을 리턴한다.
                frequencyMap.put(key, ++cnt);
                // put ( ) : 인자로 key와 value를 받는다.
                //           전달된 인자는 HashMap에 key-value 관계로 저장된다.

                // frequencySet 내의 특정 랜덤값이 존재한다면 (if)
                // frequencyMap의 value값을 cnt++ 카운트해준다.
            }
```

60번 문제

링크 https://github.com/limcholong/LectureContents/blob/main/java/CholongLim/Day21/src/Quiz60Explain.java

```
63              /*
64              else {
65                  frequencyMap.put(key, 1);
66              }
67               */
68          }
69      }
70
71      public Map<Integer, Integer> getFrequencyMap() { return frequencyMap; }
74  }
75
76  public class Quiz60Explain {
77      public static void main(String[] args) {
78          int[] testSet = {
79                  2400,  5000, 1000, 200, 6000,
80                  77000, 434,  768,  20,  50
81          };
82
83          FrequencyChecker fc = new FrequencyChecker(testSet);
84
85          fc.allocRandomFrequency( num: 1000);
86
87          System.out.println(fc.getFrequencyMap());
88          //System.out.println(fc);
89      }
90  }
```

```
frequencySet: [2400, 6000, 768, 434, 50, 20, 5000, 1000, 200, 77000]
frequencyMap: {2400=0, 6000=0, 768=0, 434=0, 50=0, 20=0, 5000=0, 1000=0, 200=0, 77000=0}
   50  2400    20  6000  6000  2400    50  5000  5000    20  6000    50    20   768    20    50    20   434   434  2400
  434  2400   200  5000  1000    50    20    20  6000  5000  6000   434   200   768    50   768    50    20    50   434
  200   200   434   200   200  5000  1000  6000  6000    20 77000    20  1000   434  2400  6000   768 77000  2400    20
77000    20   434    20   434   768   768   434    50 77000    50  1000   768   768    20    20   434  6000   768  2400
 6000    50   200    20  5000    20  6000  1000   434   434   434  5000  6000  6000 77000 77000   200   434 77000 77000
77000  6000   434  6000  5000  6000  2400    20  2400    20  1000   434  2400    20   434  1000  6000  5000   768  6000
 1000  5000  1000  1000  1000    50  5000  5000    50   200   200   434  5000  5000 77000   768   768  2400   434  6000
77000  5000   200  1000   200   768  5000 77000   434 77000   200  6000    20  2400   434  1000  2400   434   768    50
  768    20   434   768    20    50  5000  1000    20    20  2400  6000 77000  2400  6000   768  1000  2400    50  5000
77000   434   768  1000    50   434   200    20   434  2400  2400  1000  5000   434  6000   768  6000    20 77000    50
```

```
{2400=96, 6000=91, 768=85, 434=125, 50=92, 20=109, 5000=110, 1000=106, 200=96, 77000=90}

Process finished with exit code 0
```