2017학년도 1학기

JAVA 프로그래밍 II

514770-1
2017년 봄학기
4/12/2017
박경신

Lab #4 (확장 이미지 매니저 프로그램)

- 기존 요구사항 분석
  - Lab #3는 사용자가 지정한 영상파일이나 디렉토리안에 모든 영상파일을 **Photo 클래스** 객체로 관리함
  - **Lab #4는 List<Photo>를 가지고 interface, collection과 lambda를 이용한 다양한 기능을 사용**
- collections, interface, lambda(Java8), CSV/JSON
- Comparable interface, Comparator interface, sort, equals & hashCode, lambda & Stream, File

Lab #4_1 Interface

- **Lab#4_1에서는 Photo 클래스에 Comparable<Photo> 인터페이스를 구현한다.**
  - ***public int compareTo(Photo other) // fullPath 비교***
- **Lab#4_1에서는 Photo 클래스에 Comparator<Photo> 인터페이스를 구현한다.**
  - ***public static Comparator<Photo> ExtensionComparator = new Comparator<Photo>() // ext 비교***
  - ***public static Comparator<Photo> WidthComparator = new Comparator<Photo>() // width 비교***
  - ***public static Comparator<Photo> HeightComparator = new Comparator<Photo>() // height 비교***
- **그리고 Comparable과 Comparator를 사용한 sort를 구현한다.**
  - ***public static void sortBy(int mode) // mode에 따라 fullPath/ext/width/height로 정렬***
- Java2-lab4_1 폴더에 저장 후 제출

자바의 인터페이스

- 인터페이스(interface)
  - 모든 메소드가 추상 메소드인 클래스
- 인터페이스 선언
  - **interface** 키워드로 선언
  - ex) public **interface** SerialDriver {...}
- 인터페이스의 특징
  - 인터페이스의 메소드
    - public abstract 타입으로 생략 가능
  - 인터페이스의 상수
    - public static final 타입으로 생략 가능
  - 인터페이스의 객체 생성 불가
  - 인터페이스에 대한 레퍼런스 변수는 선언 가능

인터페이스

- 인터페이스 구성멤버
  - 상수필드 (constant field)
  - 추상메소드 (abstract method)
  - 디폴트메소드 (default method)
  - 정적메소드 (static method)

```
public interface 인터페이스명 {
    //상수(constant fields)
    타입 상수명 = 값;

    //추상 메소드(abstract method)
    리턴타입 메소드명(매개변수,..);

    //디폴트 메소드(default method)
    default 리턴타입 메소드명(매개변수,..) {
        ..내부구현..
    }

    //정적 메소드(static method)
    static 리턴타입 메소드명(매개변수,..) {
        ..내부구현..
    }
}
```

인터페이스 구현

- 구현 클래스 정의
  - 자신의 객체가 인터페이스 타입으로 사용할 수 있음
  - **implements** 키워드 사용
  - 여러 개의 인터페이스 동시 구현 가능
  - 상속과 구현이 동시에 가능

```
public class 클래스명 implements 인터페이스명 {
    //인터페이스에 선언된 추상 메소드의 실체 메소드 구현

}
```

- 추상 메소드의 실체 메소드를 작성하는 방법
  - 메소드의 선언부가 정확히 일치해야 함
  - 인터페이스의 모든 추상 메소드를 재정의하는 실체 메소드를 작성해야 함
    - 일부 추상메소드만 재정의할 경우, 구현 클래스는 추상 클래스

인터페이스 다중 구현

- **다중 인터페이스(multi-interface) 구현 클래스**
  - 구현 클래스는 다수의 인터페이스를 모두 구현
  - 객체는 다수의 인터페이스 타입으로 사용

```
public class 클래스명 implements 인터페이스명A,  인터페이스명B {
    //인터페이스A에 선언된 추상 메소드의 실체 메소드 구현
    //인터페이스B에 선언된 추상 메소드의 실체 메소드 구현
}
```

Comparable 인터페이스

- Comparable 인터페이스는 객체의 비교를 위한 인터페이스로 객체 간의 순서나 정렬을 하기 위해서 사용

```
public interface Comparable {
    // 이 객체가 다른 객체보다 크면 1, 같으면 0, 작으면 -1을 반환한다.
    int compareTo(Object other);
}
```

```
class Person implements Comparable {
    public int compareTo(Object other) {
        Person p = (Person)other;
        if (this.age == p.age) return 0;
        else if (this.age > p.age) return 1;
        else    return -1;
    }
}
```

Comparator 인터페이스

- Comparator 인터페이스는 다른 두 개의 객체를 비교하기 위한 인터페이스

```
public interface Comparator {
      // o1가 o2보다 크면 1, 같으면 0, 작으면 -1을 반환한다.
      int compare(Object o1, Object o2);
}
```

```
class AgeComparator implements Comparator {
    public int compare(Object o1, Object o2) {
        Person p1 = (Person)o1;
        Person p2 = (Person)o2;
        if (s1.age == s2.age) return 0;
        else if (s1.age > s2.age) return 1;
        else   return -1;
    }
}
```

Custom 클래스에 대한 sort 메소드 사용

- 개인적으로 만든 클래스에 대해서 컬렉션에 추가하고, Collections.sort 기능을 이용해서 정렬하고 싶다면 java.lang.Comparable 인터페이스를 구현해주어야 함

```
public interface Comparable<T> {
   int compareTo(T o);
}
```

- compareTo(T o) 메소드는 현 객체를 인자로 주어진 o와 비교해서 순서를 정한 후에 정수(int) 값을 반환함
  - 만약 현 객체가 주어진 인자보다 작다면 음수를 반환
  - 만약 현 객체가 주어진 인자와 동일하다면 0을 반환
  - 만약 현 객체가 주어진 인자보다 크다면 양수를 반환

Custom 클래스에 대한 sort 메소드 사용

- 객체 리스트의 정렬은 collections.sort() 메소드를 사용
- 만약 사용자 객체(Object)일 경우, 그 객체는 comparable interface를 구현해야 sort() 메소드가 동작한다.

```
List<Photo> pList = new ArrayList<Photo>();
pList.add(new Photo("C:/JAVA/IMG3.jpg"));
pList.add(new Photo("C:/JAVA/IMG1.png"));
pList.sort(null); // Comparable interface를 구현한 원소로 이루어진 컬렉션을 정렬
```

- 만약 사용자 객체(Object)에 특정 방식 sort을 원할 경우, comparator interface를 구현한 객체를 sort() 메소드에 전달해야 동작한다.

```
pList.sort(ExtensionComparator); // Comparator interface 객체를 전달하여 컬렉션을 정렬
public static Comparator<Photo> ExtensionComparator = new Comparator<Photo>() {
    public int compare(Photo p1, Photo p2) {
        return p1.getExt().compareTo(p2.getExt()); //ascending order
    }
};
```

Lab #4_2 equals & hashCode

- **Lab#4_2에서는 Photo 클래스에 equal 과 hashCode 메소드 재정의 (method override)를 한다.**
- **그리고 equals (동등성) 과 hashCode (동일성) 테스트 해보고 차이점을 살펴본다.**
- Java2-lab4_2 폴더에 저장 후 제출

ArrayList<E>

- ArrayList<E>의 특성
  - java.util.ArrayList, 가변 크기 배열을 구현한 클래스
    - <E>에서 E 대신 요소로 사용할 특정 타입으로 구체화
  - ArrayList에 삽입 가능한 것
    - 객체, null
    - 기본 타입(int, double)은 박싱/언박싱으로 Wrapper 객체로 만들어 저장
  - ArrayList에 객체 삽입/삭제
    - 리스트의 맨 뒤에 객체 추가
    - 리스트의 중간에 객체 삽입
    - 임의의 위치에 있는 객체 삭제 가능
  - 벡터와 달리 스레드 동기화 기능 없음
    - 다수 스레드가 동시에 ArrayList에 접근할 때 동기화되지 않음
    - 개발자가 스레드 동기화 코드 작성

HashMap<K,V>

- HashMap<K,V>
  - 키(key)와 값(value)의 쌍으로 구성되는 요소를 다루는 컬렉션
    - java.util.HashMap
    - K는 키로 사용할 요소의 타입, V는 값으로 사용할 요소의 타입 지정
    - 키와 값이 한 쌍으로 삽입
    - 키는 해시맵에 삽입되는 위치 결정에 사용 **(키는 중복이 허용안됨 – 중복시 마지막 키로 대체됨)**
    - 값을 검색하기 위해서는 반드시 키 이용 **(값은 중복이 허용됨)**
  - 삽입 및 검색이 빠른 특징
    - 요소 삽입 : put() 메소드
    - 요소 검색 : get() 메소드
  - 예) HashMap<String, String> 생성, 요소 삽입, 요소 검색

```
HashMap<String, String> h = new HashMap<String, String>();
h.put("apple", "사과"); // "apple" 키와 "사과" 값의 쌍을 해시맵에 삽입
String kor = h.get("apple"); // "apple" 키로 값 검색. kor는 "사과"
```

== & equals & hashCode

- **equals**는 두 객체의 **내용이 같은지** 동등성(equality)을 비교하는 연산자

```
Person p1 = new Person("Jason", 10);
Person p2 = new Person("Jason", 10);
Person p3 = p1;
// ==
if (p1 == p2) System.out.println("p1 == p2");
else System.out.println("p1 != p2"); //동일한 reference를 가리키지 않으므로 p1 != p2
if (p1 == p3) System.out.println("p1 == p3"); // 동일한 reference이므로 p1 == p3
else System.out.println("p1 != p3");
// equals
if (p1.equals(p2)) System.out.println("p1 equals p2"); // equals override 되어있으면 true
else System.out.println("p1 is not equal to p2"); // equals override 안되어있으면  false
```

== & equals & hashCode

- **hashCode**는 두 객체가 **같은 객체인지** 동일성(identity)을 비교하는 연산자

```
Map<Photo, Integer> photoMap = new HashMap<Photo, Integer>();
photoMap.put(p1, 1); // equals와 hashCode override 되어 있다면 p1과 p2는 동일
photoMap.put(p2, 2); // equals있어도 hashCode override되어 있지 않으면 p1과 p2는 다름
for (Map.Entry<Photo, Integer> entry : photoMap.entrySet()) {
    System.out.println("Photo : " + entry.getKey() + " Index : " + entry.getValue());
}
System.out.println("photoMap=" + photoMap.size()); // photomap=1
photoMap.remove(p1); // p1과 p2가 같은 hashCode를 가지므로 p1으로 p2 를 지움
for (Map.Entry<Photo, Integer> entry : photoMap.entrySet()) {
System.out.println("Photo : " + entry.getKey() + " Index : " + entry.getValue());
}
System.out.println("after remove photoMap=" + photoMap.size()); // photomap=0
```

Lab #4_3 lambda & Stream API (Java8)

- Lab#4_3에서는 List<Photo>에 람다식을 적용해 다양한 기능을 살펴본다.
- Stream API
  - stream() – stream을 생성
  - forEach() – stream 개별 요소마다 순회하며 개별 처리
  - map() – stream 개별 요소마다 다른 임의의 형태로 변환하여 새로운 stream 반환
  - filter() – stream 개별 요소마다 조건과 일치하는 모든 요소를 담은 새로운 stream 반환
  - reduce() – stream을 단일 요소로 반환
  - collect() – stream을 컬렉션 객체로 만들어서 반환
- 그리고 람다식을 사용한 convertIfTo(String ext, String format)과 resizeIfTo(String ext, int width, int height)를 구현한다.
- Java2-lab4_3 폴더에 저장 후 제출

Stream API

- 자바에서 **데이터 처리**를 위해서 대부분 **Collection**을 사용
- 복잡한 **데이터 처리 QUERY**를 표현하기 위하여 **Stream**을 사용
- **Stream과 collection의 차이**
  - 스트림은 요소들을 보관하지 않음. 요소들은 하부의 컬렉션에 보관되거나 필요할 때 생성.
  - 스트림 연산은 원본을 변경하지 않음. 대신 결과를 담은 새로운 스트림을 반환.
  - 스트림 연산은 lazy 처리됨 (즉, 결과가 필요하기 전에 실행되지 않음). 결과적으로 무한 스트림 생성가능.
- **Stream API의 대표적인 메소드**
  - stream() – stream을 생성
  - forEach() – stream 개별 요소마다 순회하며 개별 처리
  - map() – stream 개별 요소마다 다른 임의의 형태로 변환하여 새로운 stream 반환
  - filter() – stream 개별 요소마다 조건과 일치하는 모든 요소를 담은 새로운 stream 반환
  - reduce() – stream을 단일 요소로 반환
  - collect() – stream을 컬렉션 객체로 만들어서 반환

Stream API

```
Arrays.asList(5,3,4,7,2).stream();
Arrays.asList(5,3,4,7,2).stream().forEach(System.out::println); // 람다식보다 메소드참조 사용
Arrays.asList(5,3,4,7,2).stream().map(i-> i*i).forEach(System.out::println);
Arrays.asList(5,3,4,7,2).stream().filter(i-> i>3).forEach(System.out::println);
int sum = Arrays.asList(5,3,4,7,2).stream().reduce((i, j) -> i+j).get(); // ((((5+3)+4)+7)+2)
System.out.println("sum=" + sum); // sum=21
List<Integer> intList = Arrays.asList(5,3,4,7,2).stream().filter(i-> i>3).collect(Collectors.toList());
intList.forEach(System.out::println);
int result = Arrays.asList(5,3,4,7,2).stream().filter(i-> i>3).findFirst().orElse(null); // 3보다 큰 첫번째 요소 반환
System.out.println("result=" + result); // result=5
Arrays.asList(5,3,4,7,2).stream().filter(i-> i>3).findAny().ifPresent(System.out::println); // 3보다 큰 요소 반환 // 5
boolean result2 = Arrays.asList(5,3,4,7,2).stream().allMatch(i -> i>0); // 모두다 0보다 큰지
System.out.println("result2=" + result2); // true
```

Lab #4_4 ImageManager 클래스 작성

- Lab#4_4에서는 List<Photo>를 관리하는 ImageManager 클래스를 작성한다.
  - ImageManager 클래스는 photoList를 멤버로 하고, 다음 메소드를 포함한다.
    - private List<Photo> photoList = new ArrayList<Photo>(); // 이미지 리스트
    - *public ImageManager(String dirPath) // 디렉토리안의 모든 이미지를 리스트로 생성*
    - *public void addImagesInDirectory(String dirPath) // 디렉토리안의 모든 이미지를 리스트로*
    - *public void print() // 모든 이미지 리스트 정보 출력*
    - *public void sortBy(int mode) // 이미지리스트 정렬*
    - *public void convertIfTo(String ext, String format) // 이미지리스트에 ext가 있으면 format으로 변환*
    - *public void resizeIfTo(String ext, int width, int height) // 이미지리스트에 ext가 있으면 width x height 크기로 변환*
    - *public void removeIfFilenameContains(String name) // 이미지리스트에 name이 있으면 delete*
    - *public void removeIfExtEquals(String ext) // 이미지리스트에 ext이 있으면 delete*
- 그리고 ImageManager를 사용하여 모든 연산을 테스트한다.
- Java2-lab4_4 폴더에 저장 후 제출

Remove Objects from Collection while Iterating

- ArrayList는 remove(int index) 또는 remove(Object obj) 메소드를 제공함. 단 remove() 메소드는 ArrayList를 iterating하지 않은 경우에만 사용함.
- ArrayList에서 iterating하면서 remove() 해야할 경우, **Iterator**를 사용함.

```
ArrayList<String> list = new ArrayList<String>(Arrays.asList("a","b","c","d"));
for (int I = 0; i < list.size(); i++) {
    list.remove(i); // 원소가 삭제될 때 list의 사이즈가 줄면서 다른 원소들의 인덱스도 바뀜
}
for (String s : list) {
    list.remove(s); // ConcurrentModificationException 발생
}
Iterator<String> it = list.iterator();
while (it.hasNext()) {
    String s = it.next(); // Iterator의 next()가 remove()보다 먼저 호출되어야 함
    it.remove();
}
```

Lab #4_5 ImageFileManager 클래스 작성

- **Lab#4_5에서는 List<Photo>의 파일 (CSV, JSON) 입출력을 관리하는 ImageFileManager 클래스를 작성한다.**
  - *ImageFileManager 클래스는 다음 메소드를 포함한다.*
    - *public static List<Photo> loadCSV(String filename) // CSV 파일을 읽어서 List<Photo> 생성*
    - *public static saveCSV(String filename, List<Photo> photoList) // List<Photo> 에서 CSV 파일로 저장*
    - *public static List<Photo> loadJSON(String filename) // JSON 파일을 읽어서 List<Photo> 생성*
    - *public static saveJSON(String filename, List<Photo> photo:ist) // List<Photo>에서 JSON 파로 저장*
- Java2-lab4_5 폴더에 저장 후 제출

CSV (Comma-Separated Values) File

```
C:₩JAVA₩IMAGE₩IMG1.jpg,jpg,4032,3024
C:₩JAVA₩IMAGE₩IMG2.jpg,jpg,4032,3024
C:₩JAVA₩IMAGE₩IMG3.jpg,jpg,4032,3024
C:₩JAVA₩IMG1.jpg,jpg,4032,3024
C:₩JAVA₩IMG2.jpg,jpg,4032,3024
C:₩JAVA₩IMG3.jpg,jpg,4032,3024
C:₩JAVA₩IMG4.jpg,jpg,3024,3024
C:₩JAVA₩IMG5.jpg,jpg,3024,3024
C:₩JAVA₩PHOTO₩IMG4.jpg,jpg,3024,3024
C:₩JAVA₩PHOTO₩IMG5.jpg,jpg,3024,3024
```

JSON File (Indent Format)

```
 [{
  "fullPath" : "C:₩₩JAVA₩₩IMAGE₩₩IMG1.jpg",
  "width" : 4032,
  "height" : 3024,
  "ext" : "jpg"
}, {
  "fullPath" : "C:₩₩JAVA₩₩IMAGE₩₩IMG2.jpg",
  "width" : 4032,
  "height" : 3024,
  "ext" : "jpg"
},  // 중간 생략...
]
```

Jackson – Java JSON Library

- Jackson은 자바용 JSON library로, XML/CSV 등 다양한 형식의 데이터를 지원
  - jackson-core : low-level streaming API
  - jackson-annotations : annotations
  - jackson-databind : data binding (and object serialization)
- 사용법

```
ObjectMapper mapper = new ObjectMapper(); // ObjectMapper 객체 생성
```

  - **File/URL/String JSON -> Java Object**

```
Person aPerson = mapper.readValue(new File("data.json"), Person.class); // from File
value = mapper.readValue(new URL("http://some.com/some.json"), Person.class); // from URL
value = mapper.readValue("{\"name\":\"MyName\", \"age\":10}", Person.class); // from String
```

  - **Java Object -> File/Byte/String JSON**

```
Person  aPerson = new Person();
aPerson.name = "MyName"; aPerson.age= 10;
mapper.writeValue(new File("output.json"), aPerson); // to File
byte[] jsonBytes = mapper.writeValueAsBytes(myResultObject); // to Byte
String jsonString = mapper.writeValueAsString(myResultObject); // to String
```

과제 제출

- 사용자가 지정한 폴더(C:/JAVA)에는 서브 폴더(C:/JAVA/PHOTO)가 존재함
- 폴더 안에는 jpg, png, gif 포맷의 파일들이 존재함 (그 외의 다른 종류의 파일은 없음)
- Lab04_1 ~ Lab04_5와 보고서를 전체적으로 묶어서 e-learning에 과제 제출
- 각 Lab마다 본인이 추가로 작성한 코드와 설명을 중점적으로 보고할 것!