Object – Oriented Programming

LAB #11. Generics and the ArrayList

ArrayList Class

- Java의 standard library class (java.util.ArrayList)

- 동적 데이터 구조
  - 아이템이 리스트에서 추가되고 삭제될 수 있음
  - 아이템을 추가하거나 제거할 때 길이가 늘어나거나 줄어든다

ArrayList vs Array

- Java의 일반적인 array는 초기 크기가 고정되어 있기 때문에 정적 데이터 구조이다.

- ArrayList는 프로그램이 실행되는 동안 Array의 길이를 변경할 수 있다는 점을 제외하면, 배열과 동일한 용도로 사용됨.

ArrayList Class

- ArrayList 클래스는 array를 private instance variable로 사용하여 구현됨

```
public class ArrayList<E> extends AbstractList<E>
        implements List<E>, RandomAccess, Cloneable, java.io.Serializable
{
    private static final long serialVersionUID = 8683452581122892189L;

    /**
     * The array buffer into which the elements of the ArrayList are stored.
     * The capacity of the ArrayList is the length of this array buffer.
     */
    private transient Object[] elementData;

    /**
```

- Array가 꽉 차면 더 큰 배열을 만들고, 새로운 배열에 데이터를 옮긴다.

Using ArrayList

- ArrayList의 base type은 class type (또는 다른 reference type)이어야 한다.
- Primitive type (int, char, double, ...) 은 저장될 수 없다.
- Primitive type을 사용하려면 wrapper class로 boxing되어야 한다.

- ArrayList를 설정하려면
  - ArrayList 패키지를 import 하고
    - import java.util.ArrayList;
  - 기본 베이스 타입을 지정한다.
    - ArrayList<BaseType> aList = new ArrayList<BaseType>();
    - 초기 크기를 지정할 수도 있다.

      ex) ArrayList<String> aList = new ArrayList<String>(20);

ArrayList Notation

- 일반적인 Array의 경우 [ ] 을 사용하여 Array의 특정 인덱스에 있는 항목에 쉽게 접근 가능
  - String str = myArray[i];
  - myArray[i] = “Hello”;
- ArrayList는 [ ] 표기법을 제공하지 않음

  -> 대신 다음 메소드들을 사용
  - get(index) 가져오기
  - add(object) 추가하기
  - set(index, object) 바꾸기

ArrayList Notation

- ArrayList에 처음 항목을 추가하려면 add()를 사용한다.

- *myArrayList.add("Goodbye");*
  myArrayList.add("cruel");
  myArrayList.add("world");

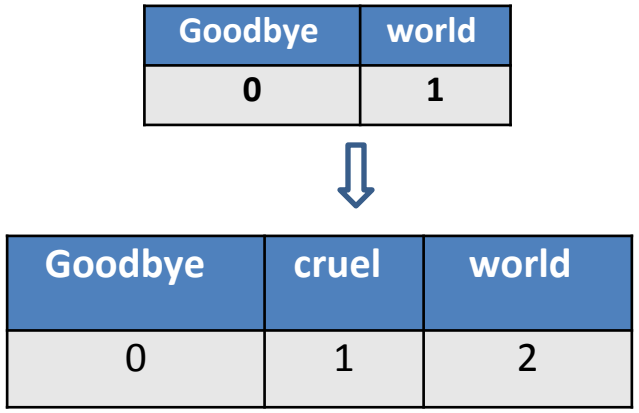
  -> 순차적으로 항목이 추가됨

Goodbye	cruel	world
0	1	2

ArrayList Notation

- add() 는 추가 매개 변수를 받아서 원하는 index에 add 할 수도 있다. (overload)
  - ex) add(index, object);

```
myArrayList.add("Goodbye");
myArrayList.add("world");
myArrayList.add(1,"cruel");
```



ArrayList Notation

- size() 메소드는 ArrayList에 저장된 element의 수를 반환한다.

Goodbye	cruel	world
0	1	2

- System.out.println(myArrayList.size());

3

ArrayList Class

Method	Description
ArrayList<*Base_Type*> (int initialCapacity)	입력된 정수 크기의 비어있는 Base_Type ArrayList 생성
ArrayList<*Base_Type*>()	10 크기의 비어있는 Base_Type ArrayList 생성
set(int index, *Base_Type* newElement)	index 위치에 있는 element를 newElement로 변경
get(int index)	index 위치에 있는 element를 반환
add(*Base_Type* newElement)	newElement를 ArrayList에 추가
add(int index, *Base_Type* newElement)	index 위치에 newElement를 추가
remove(int index)	index 위치에 있는 element를 리스트에서 삭제하고, 그 element를 반환
removeRange(int fromIndex, int toIndex)	fromIndex ≤ index < toIndex 에 해당하는 index의 element들을 삭제
clear()	element를 전부 삭제하고 ArrayList의 size를 0으로 만듬
contains(Object target)	ArrayList에 target이 존재하면 true를 반환
indexOf(Object target)	target과 동일한 첫번째 element의 index를 반환
isEmpty()	ArrayList가 비어있다면 true를 반환
size()	ArrayList의 element 개수를 반환

For each loop

- Array와 마찬가지로, for-each loop를 사용하여 collection (ex) ArrayList)의 모든 요소에 접근할 수 있음

```
for(variable : collection) {
    Statements;
}
```

For each loop

```java
import java.util.ArrayList;

public class ArrayListTest {
    public static void main(String[] args){
        //create the ArrayList
        ArrayList<Integer> aList = new ArrayList<Integer>();

        //Load objects into the list
        for(int i = 0; i < 20; i++){
            aList.add(new Integer(i));
        }
        System.out.println("Loaded integers into list.");

        //using the the for-each loop data can be retrieved
        for (Integer itger:aList){
            System.out.println(itger.intValue());
        }
    }
}
```

```
Loaded integers into list.
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

Generics

- 일반적인 코드를 작성하고, 이 코드를 **다양한 타입**에 대하여 재사용하는 기법
- 클래스에서 사용할 타입을 클래스 외부에서 설정
- 타입 사용에 대한 유연함 + 컴파일 타임에 **타입에 대한 안정성을 보장**받을 수 있다

Generic Class

- Type parameter가 있는 클래스 정의
  - Type parameter는 클래스 선언부의 클래스 이름 뒤 “〈 〉” 안에 작성됨
    - *Public class myClass〈T〉 { }*
    - *Public class myClass〈T, U, ...〉 { }*
- Type parameter의 type은 항상 reference 형식이어야 한다.
  - Primitive type (int, double, char, ...) 은 불가능

Defining Generic Class

Display 14.4 A Class Definition with a Type Parameter

```
1  public class Sample<T>
2  {
3      private T data;

4      public void setData(T newData)
5      {
6          data = newData;
7      }

8      public T getData()
9      {
10         return data;
11     }
12 }
```

T is a parameter for a type.

Generic Class

```
1  class Test<T>{
2      private T t;
3
4      public void set(T t) {
5          this.t = t;
6      }
7
8      public T get() {
9          return t;
10     }
11 }
```

```
1  public static void main(String[] args) {
2
3      Test<String> test = new Test();
4      test.set("test");
5      System.out.println(test.get());
6  }
```

실행 결과 : test

Generic Methods

- 일반적으로 generic 클래스가 정의되면
  해당 generic 클래스의 메소드 정의에 type parameter를 사용할 수 있다.

- 일반 클래스에서도 generic 메소드는 선언 가능하다.

- generic 메소드의 type parameter는 해당 메소드에만 적용된다. (not class)

Generic Methods

- 선언 시

  public static <T> T genMethod(T a)

  public <T> int genMethod2(T[ ] b)

  -> T 배열을 parameter로 받고 T 타입을 반환

- 호출 시

  String s = Class.<String>genMethod(c);

  int i = object.genMethod2(d); // 인자(d)의 타입에 따라 추론

Generic Methods

```
public class Utility{
    //…

    public static <T> T getMidpoint(T[] a){
        return a[a.length/2];
    }

    public static <T> T getFirst(T[] a){
        return a[0];
    }
    //…
}

String midString = Utility.<String>getMidpoint(b);
double firstNumber = Utility.<Double>getFirst(c);
```

caution: generic 클래스에서의 static method는
class와 별도의 타입 파라미터를 설정해야 함 (의존 관계 모순)

실습

- Hub, Gyeonggi, Gangwon, ServiceManagement, Logistics 클래스를 생성한다.

- Hub 클래스
  - Gyeonggi와 Gangwon 클래스는 Hub 클래스를 부모 클래스로 둔다.
  - Hub 의 모든 instance 변수들은 private로 선언하고 모두 getter/setter를 생성한다.
  - 아래 4개의 instance variables를 생성한다.
    - int number(박스 시리얼 번호), String description(박스 객체의 설명),
      String area(배송 허브 위치), double price_per_box(박스 개당 배송 단가)
  - toString 메소드를 아래의 형식으로 String 형을 반환하게끔 overriding 한다.

Box Number : 10005
Description : GaPyeong#1
Area : Gyeonggi
Price per box : 3150.0

실습

- Gyeonggi 클래스

```
public static int init_num = 10000;
public static String init_area = "Gyeonggi";
public static double init_price_per_box = 3000;
```

  - 3개의 static 변수를 가진다.
  - 생성자는 **description** 값만 받고, area와 price_per_box 각각 init_area 과 init_price_per_box를 그대로 사용하며 number 는 init_num 을 1씩 증가시킨다.
    - 가장 처음에 추가되는 번호는 10001부터 시작한다.

- Gangwon 클래스

```
public static int init_num = 20000;
public static String init_area = "Gangwon";
public static double init_price_per_box = 4000;
```

  - 3개의 static 변수를 가진다.
  - 생성자는 **description** 값만 받고, area와 price_per_box 각각 init_area 과 init_price_per_box를 그대로 사용하며 number 는 init_num 을 1씩 증가시킨다.
    - 가장 처음에 추가되는 번호는 20001부터 시작한다.

실습

- ServiceManagement 클래스
  - 총 5개의 static generic method를 생성한다. (제공되는 메소드 1개(`raiseAll`), 구현 해야하는 메소드 4개)

```java
public static <T extends Hub> int findIndexByNum(ArrayList<T> tList, int num) {
	// 택배 박스 번호로 찾기
	// 찾으면 해당 인덱스 반환, 찾지 못하면 -1 반환
	return -1;
}

public static <T extends Hub> T raisePerBox(T t, double rate) {
	// 택배 박스 개당 요금 변경, e.g. 1.05 == 5% 상승
	return t;
}
```

실습

- ServiceManagement 클래스
  - 클래스 상단에 필요한 모듈을 가져온다.

```
import java.lang.reflect.Field;
```

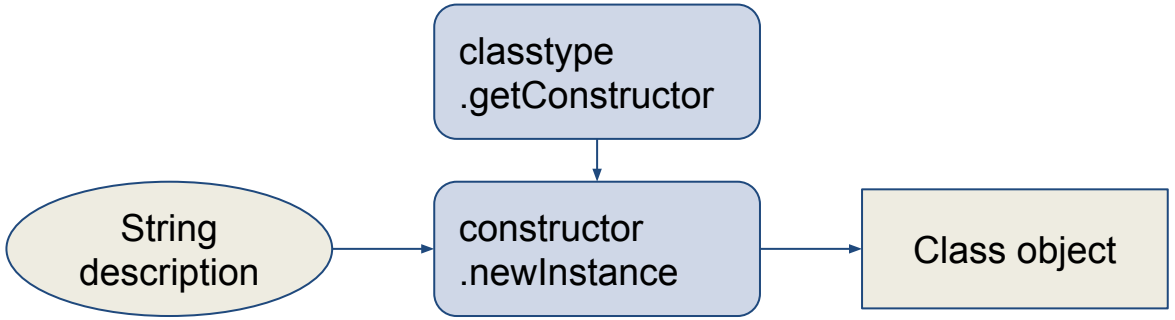
```
public static <T extends Hub> ArrayList<T> raiseAll(Class<T> c, ArrayList<T> tList, double rate) {
	// tList의 모든 박스 요금 변경, e.g. 1.05 == 5% 상승
	for(T elem:tList) {
		// 아래 메소드는 자신이 정한 메소드 이름에 맞게 변경
		elem.setPricePerBox(rate*elem.getPricePerBox());
	}

	// 해당 지역 허브의 기본 택배 요금 변경
	try {
		Field f = c.getField("init_per_box");
		double value = f.getDouble(null);
		f.setDouble(null, value*rate); // no need to be an object, because it is static variable.
	} catch (NoSuchFieldException | SecurityException | IllegalArgumentException | IllegalAccessException e) {
		e.printStackTrace();
	}
	return tList;
}
```

실습

- ServiceManagement 클래스
  - 박스를 각 배송지 Hub로 배분하는 packageBoxes generic 메소드를 만든다.
  - 주어진 description 배열을 각 Hub의 인스턴스로 캡슐화 한다.

```
public static <T extends Hub> void packageBoxes(String[] descriptions, Class<T> classtype, ArrayList<T> tList) {
/*
 * 1. descriptions 배열의 모든 항목을 각 Hub별 인스턴스 생성자를 이용해 캡슐화한다.
 * 2. 이 때, 인자로 받은 classtype 매개변수로 getConstructor, newInstance 메소드를 호출할 수 있다.
 * 3. tList에는 해당 Hub 타입을 가진 인스턴스만 담는다.
 * 4. try/catch를 사용해, 위 메소드를 이용하기 위해 필요한 exception 처리를 수행한다.
 */
}
```



실습

- ServiceManagement 클래스
  - 배송지 Hub를 변경하는 changeHub generic 메소드를 만든다.

```
public static <T extends Hub, U extends Hub> void changeHub(
    ArrayList<T> tList, int index, Class<U> classtype, ArrayList<U> uList
) {
    try {
    // tList에서 index를 통해 객체를 하나 가져온다.
    // getDescription 메소드를 이용해 description 정보를 가져온다.
    // 해당 정보를 이용해 U classtype으로 새로운 객체를 생성한다.
    // tList에서 원래 객체를 null로 세팅한다.
    // 새로 만든 객체는 지연 배송에 따른 할인으로 -10%를 적용한다. (0.9)
    // uList에 새로 만든 객체를 추가한다.
    } catch () { // packageBoxes 메소드와 동일한 방식으로 exception 처리를 한다.
        e.printStackTrace();
    }
}
```

실습

- Logistics 클래스
  - main method를 가진 클래스이다.

실습

```
public static void main(String[] args) {
	String[] new_gyeonggi_boxes = {"SuWon#1","SeongNam#1","YongIn#1","GoSeong#1","GaPyeong#1"};
	String[] new_gangwon_boxes = {"Gangneung#1","Wonju#1"};
	ArrayList<Gyeonggi> gyList = new ArrayList<>();
	ArrayList<Gangwon> gaList = new ArrayList<>();

	Scanner scan = new Scanner(System.in);

	// Gyeonggi 와 Gangwon 의 box String 목록을 모두 인스턴스화해서 ArrayList에 넣는다. (tip: ClassName.class)

	System.out.println("*** Oh, delivery price has been increased!! ***");
	// Gyeonggi 와 Gangwon 의 배송 요금을 모두 5% 인상한다.

	System.out.println("Which box is important in Gangwon-area?");
	// 콘솔 입력을 통해 정수값 id를 받아서 Gangwon arraylist에서 index를 찾는다.
	if(index == -1) {
		System.out.println("nothing");
	} else {
		System.out.println("The box \""+gaList.get(index).getDescription()+"\" is important! be careful!\n");
		// 반환된 index의 해당 box 요금을 20% 인상한다. (payment for risk-위험물 배송 추가 요금)
	}
```

실습

```
        System.out.println("Which box has the wrong area in Gyeonggi-area?");
        // 콘솔 입력을 통해 정수값 id를 받아서 Gyeonggi arraylist에서 index를 찾는다.

        if(index == -1) {
              System.out.println("nothing");
        } else {
              System.out.println("The box \"" + gyList.get(index).getDescription()
                                          + "\" is actually has to go to Gangwon! late! hurry up!\n");
              // gyList에서 반환된 index에 해당하는 box의 목적 허브를 "Gangwon"으로 변경한다.
              // 지연 배송에 대한 discount로 box의 요금을 10% 할인한다. (0.9)
        }

        for(Gyeonggi g : gyList) {
              System.out.println(g+"\n");
        }
        for(Gangwon j : gaList) {
              System.out.println(j+"\n");
        }
        scan.close();
}
```

실습

- 실행화면

```
*** Oh, delivery price has been increased!! ***
Which box is important in Gangwon-area?
20001
The box "Gangneung#1" is important! be careful!

Which box has the wrong location in Gyeonggi-area?
10004
The box "GoSeong#1" is actually has to go to Gangwon! late! hurry up!

-------- Delivery List for Gyeonggi --------
Box Number : 10001
Description : SuWon#1
Area : Gyeonggi
Price per box : 3150.0

Box Number : 10002
Description : SeongNam#1
Area : Gyeonggi
Price per box : 3150.0

Box Number : 10003
Description : YongIn#1
Area : Gyeonggi
Price per box : 3150.0

null

Box Number : 10005
Description : GaPyeong#1
Area : Gyeonggi
Price per box : 3150.0

-------- Delivery List for Gangwon --------
Box Number : 20001
Description : Gangneung#1
Area : Gangwon
Price per box : 5040.0

Box Number : 20002
Description : Wonju#1
Area : Gangwon
Price per box : 4200.0

Box Number : 20003
Description : GoSeong#1
Area : Gangwon
Price per box : 3780.0
```

다음주 실습 시간 퀴즈 :

- Generics
- ~~File IO~~
- ~~Exception~~
- Interface

이 중에서 볼 예정

번외

- https://jjjwodls.github.io/java/2020/02/02/02-Java-Generic-Exam.html

  Generic을 이용한 스택 만들기