Exercise No.3
(Generics)

Q1

Implement MultiCollection class that contains two collections:List enables to store String objects, Map enables to store String values that corresponed to String keys. MultiCollection prepares three methods. addList(String s) stores the argument into List while addMap(String k, String v) stores the second argument that corresponds to the first argumet into Map. In addition, MultiCollection also prepares showAll() that displays the values stored in the List and Map as following example.

Also, implement Q31Main that contains main method in which you have to implement specifications described below.

1. Read one line from standard input, then continue step2 and step3 while the string does not equal "end"
2. If the string contains ",", split it into two parts, then use the left string as a key and the right string as a value that are stored into Map by using addMap.
3. If the string does not contain ",", store it into List by using addList.
4. If the string equals "end", invoke showAll() then finish the program.

An example of MultiCollection

```
public class MultiCollection {
    public void addList(String str) {
        .....
    }
    public void addMap(String key, String value) {
        .....
    }
    public void showAll() {
        .....
```

```
    }
}
```

An example of Q31Main

```
public class Q31Main {
    public static void main(String[] args) {
        MultiCollection mCollection = new MultiCollection();
        // implement the above algorithm

        mColleciton.showAll();
    }
}
```

Execution example

```
%java Q31Main
Input: test
Input: yahoo
Input: key1,value
Input: key2,value
Input: end
## List ##
test
yahoo
## Map ##
key1 : value
key2 : value
```

Q2

Implement the following methods in Shop class that uses Dictionary, Novel, Phone, IPhone and Xperia.

- bookDisplay
  receives Book and its extension as an argument, then invokes methods defined in Book (getIndex, getPageNum, getPrice).
- phoneDisplay
  receives Phone and its extension as an argument, then invokes methods defined in Phone (getName, getNumOfApp, getPrice).
- display
  receives List of any types and display each object in the List by detecting the type.

In this experiment, we assuem that display receives only List of Phone or Book. Note that, the List that is passwd as an argument of display needs to use type parameter.

Implement above methods so that we can get the same result as follows (including space and linefeed).

Shop.java

```
public class Shop {

    public static void main(String[] args) {
        List<Phone> plist = new ArrayList<Phone>();
        plist.add(new IPhone());
        plist.add(new Xperia());

        List<Book> blist = new ArrayList<Book>();
        blist.add(new Dictionary());
        blist.add(new Novel());

        Shop shop = new Shop();
        shop.bookDisplay(blist);
        System.out.println();
        shop.phoneDisplay(plist);

        System.out.println();
        shop.display(blist);
        System.out.println();
        shop.display(plist);
    }
```

```
    public void bookDisplay(....);
    public void phoneDisplay(....);
    public void display(....);

}
```

Phone.java

```
public class Phone {

    protected String name;
    protected int numOfApp;
    protected int price;

    public String getName() {
        return name;
    }

    public int getNumOfApp() {
        return numOfApp;
    }

    public int getPrice() {
        return price;
    }
}
```

IPhone.java

```
public class IPhone extends Phone {

    public IPhone() {
        name = "IPhone";
        price = 50000;
        numOfApp = 5000;
    }
}
```

Xperia.java

```
public class Xperia extends Phone{
```

```
    public Xperia() {
        name = "Xperia";
        price = 45000;
        numOfApp = 300;
    }
}
```

Book.java

```
public class Book {

    protected String index;
    protected int pageNum;
    protected int price;

    public String getIndex() {
        return index;
    }

    public int getPageNum() {
        return pageNum;
    }

    public int getPrice() {
        return price;
    }
}
```

Dictionary.java

```
public class Dictionary extends Book{

    public Dictionary() {
        index = "a, b, c, d, e";
        pageNum = 360;
        price = 4000;
    }
}
```

Novel.java

```
public class Novel extends Book {
```

```
    public Novel() {
        index = "1. Introduction, 2. Happening..";
        pageNum = 280;
        price = 760;
    }
}
```

Execution example

```
%java Shop
-Index
  a, b, c, d, e
  Total page = 360
  Price = 4000
-Index
  1. Introduction, 2. Happening..
  Total page = 280
  Price = 760

Name = IPhone
  App Num = 5000
  Price = 50000
Name = Xperia
  App Num = 300
  Price = 45000

Kind: book
  Index  = a, b, c, d, e
  Price  = 4000
Kind: book
  Index  = 1. Introduction, 2. Happening..
  Price  = 760

Kind: phone
  Name  = IPhone
  Price = 50000
Kind: phone
  Name  = Xperia
  Price = 45000
```

Q3

Implement PairList that stores a pair of objects as List fashion. PairList stores the pair the type of which can be specified by type parameter E1 and E2. Also this class contains get(int index) that returns an instance of Holder that uses the same type parameters. You also have to implement this Holder that contains showAll() method that displays the pairs it stores. Use these classes by executing Q33Main shown as follows.

An example of PairList

```
public class PairList<E1, E2> {

    public void add(....) {
        .....
    }
    public Holder<E1, E2> get(int index) {
        .....
    }
}
```

An example of Holder

```
public class Holder<V1, V2> {

    public Holder(....) {
        .....
    }
    public void showAll() {
        .....
    }
}
```

Q33Main and execution results

```
public class Q33Main {

    public static void main(String[] args) {
        PairList<String, Integer> list1 =
                  new PairList<String, Integer>();
        list1.add("key1", 10);
        list1.add("key2", 20);
        list1.add("key3", 30);
```

```
        Holder<String, Integer> holder = null;
        holder = list1.get(0);
        holder.showAll();

        holder = list1.get(1);
        holder.showAll();

        holder = list1.get(2);
        holder.showAll();
    }
}

%java Q33Main
v1:v2 = key1:10
v1:v2 = key2:20
v1:v2 = key3:30
```

Q4

Q34Main, as shown below, creates a list of HeavyWorker that implements Worker interface, and invokes workAll with the list as an argument. Then it will get and display a list of Result. However, we will find compile errors if we do not modify something in these probram. So, let's find the mistakes and fix them. At this time, we have two ways to do it. 1) find the way that modify Q34Main, 2) modify some classes without modifying Q34Main. Please answer these ways respectively.

Q34Main

```
import java.util.ArrayList;
import java.util.List;


public class Q34Main {

    public static void main(String[] args) {
        List<HeavyWorker> workerList =
                          new ArrayList<HeavyWorker>();
        for(int i = 0; i < 3; i++) {
            HeavyWorker heavyWorker =
                          new HeavyWorker("pre" + i, "suf" + i);
            workerList.add(heavyWorker);
        }

        List<Result<String>> resultList =
                                  WorkerUtil.workAll(workerList);
        for(Result<String> r : resultList) {
            System.out.println(r.getValue());
        }
    }
}
```

Worker

```
public interface Worker<V> {

    public V work();

}
```

Result

```
public class Result<V> {

    private V value;

    public void setValue(V value) {
        this.value = value;
    }

    public V getValue() {
        return value;
    }

}
```

HeavyWorker

```
public class HeavyWorker implements Worker<String> {

    protected String arg1;
    protected String arg2;

    public HeavyWorker(String arg1, String arg2) {
        this.arg1 = arg1;
        this.arg2 = arg2;
    }
    public String work() {
        return "Heavy : " + arg1 + arg2;
    }
}
```

WorkerUtil

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

public class WorkerUtil {

    public static <T> List<Result<T>>
                          workAll(Collection<Worker<T>> tasks) {
```

```
        List<Result<T>> resultList = new ArrayList<Result<T>>();

        for(Worker<T> t : tasks) {
            T value = t.work();
            Result<T> result = new Result<T>();
            result.setValue(value);
            resultList.add(result);
        }

        return resultList;
    }
}
```

Q5

Implement TinyList that is tiny version of List in Java. The specification is as follows.

- receives type parameter and can manipulate any type (e.g., Object).
- can add an object that is specified by type parameter.
- get(int index) extracts and returns an object the type of which is specified by type parameter.
- remove() removes and returns the last object the type of which is specified by type parameter.
- the maximum number that this TinyList can store is 10, then returns null if the index of get(int index) is below 0 or over 10.

Note that, you **cannnot** use collection framework (e.g., List).

Example of TinyList

```
public TinyList<E> {

    public E add(E element) {
    }

    public E get(int index) {
    }

    public E remove() {
    }

}
```