Exercise 4

Objectives

In this exercise, you will learn how to use your classes to create objects and how to use the objects you created. In this exercise, you will learn one of the most important concepts in object-oriented programming: object references.

A: Passing Primitive Parameters to a Method [1 pt]

This application illustrates what happens when a parameter of a primitive type is modified within a method.

```
public class PrimitiveCall{

    public void test(){
        int x = 8;
        int y = 5;

        movePoint(x, y);

        System.out.println("(x, y) = (" + x + ", " + y + ")");
    }

    private void movePoint(int x, int y){
        x = 100;
        y = 120;
    }

    public static void main(String[] args){
        new PrimitiveCall().test();
    }
}
```

Explain what the values of x and y are and why their values have not changed (in `PrmitiveCall.txt`).

Submission Files	**Types**
`PrimitiveCall.txt`	`Texts in English`

B: Passing References to a Method [1 pt]

Predict the output of the following program ReferenceCall.java.

```
public class Point{
   private int x, y;

   public Point(int x, int y){
      this.x = x;
      this.y = y;
   }

   public void setX(int x){ this.x = x; }
   public void setY(int y){ this.y = y; }
   public int getX(){ return x; }
   public int getY(){ return y; }
}
```

```
public class ReferenceCall{

   public void test(){
      Point point = new Point(8, 5);

      movePoint(point);

      int x = point.getX();
      int y = point.getY();
      System.out.println("(x, y) = (" + x + ", " + y + ")");
   }

   private void movePoint(Point p){
      p.setX(100);
      p.setY(120);
      //(*)
   }

   public static void main(String[] args){
      new ReferenceCall().test();
   }
}
```

Also, predict the output if the following statement is added at the (*) position in ReferenceCall.java.

```
p = new Point(0, 0);
```

Execute the program and explain the process of obtaining each output (in `ReferenceCall.txt`).

Submission Files	Types
ReferenceCall.txt	Texts in English

C: Doubly Linked List [2 pt]

Your task is to create a SimpleList class, a double-directed linked list, and a SimpleNode class, an element of the SimpleList class. A SimpleNode object has an integer as its key. The SimpleList class has the following methods:

- insert(int *key*): inserts a new element with *key* at the beginning of the list.
- delete(int *key*): deletes the element corresponding to *key* from the list. Note, if more than one element of specified key is contained in the list, the first element reached when traversing from the beginning of the list should be deleted.
- printList(): prints out keys of elements currently stored in the list in order.

Elements of a SimpleList object consists of SimpleNode objects. A SimpleNode object has the key and references to the previous and next elements respectively. You should use the following template for the SimpleList class.

```
class SimpleList{
   private SimpleNode nil;

   SimpleList(){
      init();
   }

   void init(){
      nil = new SimpleNode();
      nil.setNext(nil);
      nil.setPrev(nil);
   }

   SimpleNode listSearch(int key){
      /* your code */
   }

   void printList(){
      /* your code */
   }

   void delete(int key){
      /* your code */
   }

   void insert(int key){
      /* your code */
   }
}
```

Please test your program by the following SimpleListApplication class.

```java
import java.util.Scanner;

class SimpleListApplication{
    public static void main(String[] args){
        new SimpleListApplication().run();
    }

    void run(){
        SimpleList list = new SimpleList();
        Scanner sc = new Scanner(System.in);

        int Q = sc.nextInt();
        for ( int i = 0; i < Q; i++ ){
            String command = sc.next();
            if ( command.equals("insert") ){
                int key = sc.nextInt();
                list.insert(key);
            } else if ( command.equals("delete") ) {
                int key = sc.nextInt();
                list.delete(key);
            } else if ( command.equals("print") ) {
                list.printList();
            }
        }
    }
}
```

If SimpleListApplication produces the corresponding output for the following sample input, it can be determined that SimpleNode and SimpleList are implemented correctly.

Sample Input	**Sample Output**
10 insert 8 insert 5 insert 2 insert 11 insert 6 delete 2 print insert 15 delete 6 print	6 11 5 8 15 11 5 8

16
insert 108
insert 100
insert 107
insert 100
insert 106
insert 100
print
delete 100
print
delete 100
print
insert 105
delete 105
delete 108
insert 109
print
	100 106 100 107 100 108
	106 100 107 100 108
	106 107 100 108
	109 106 107 100

Submission Files	Types
`SimpleNode.java`	`Java Class`
`SimpleList.java`	`Java Class`
`SimpleListApplication.java`	`Java Class`

D: Gas Station [2 pt]

The gas station at the White Tiger Service Area has *N* lanes, numbered from 1 to *N*. The first car in each lane is allowed to refuel.

When a car enters the gas station, it chooses the lane with the fewest cars in line and lines up at the end of the queue. If there is more than one such lane, they choose the lane with the lowest number. When the car has finished refueling, it will leave the lane and the car behind it will refuel. Once you have chosen a lane, you cannot move to another lane. Also, the order of the cars in the lane will not change.

Given the number of lanes, the cars that have entered, and the lanes that have finished refueling, write a program that outputs the numbers of the cars that have finished refueling in order. The entry information is given as the number of the car entering the stand, and the refueling end information is given as the lane number where the first car has finished refueling. It is assumed that no cars are lined up in any lane at the beginning.

The input is given in the following form.

N M
info_1
info_2
:
info_M

In the first line, the number of lanes *N* ($1 \le N \le 10$), and the number of information *M* ($2 \le M \le 10{,}000$) are given. In the following *M* lines, each information info_i is given. Each info_i is given in one of the following formats.

"`0` *lane*" or "`1` *car*"

If the first number is 0, it means that the first car in the lane whose number is *lane* ($1 \le lane \le N$) has finished refueling. If the first number is 1, it indicates that the car whose number is *car* ($1 \le car \le 9{,}999$) has entered the stand.

The input satisfies the following constraints.

- The numbers of all cars entering the gas station are different.
- There is always at least one piece of information where the first number is 0 or 1, respectively.
- For a lane with no cars, no information whose first number is 0 will be given.

For each refueling end information, output the number of the car that has finished refueling in a line. Here is sample input/output data.

Sample Input	Sample Output

2 7
1 999
1 1000
0 2
1 1001
1 1002
0 1
0 1
	1000
	999
	1002

If you like, you can validate your code by AOJ:
https://onlinejudge.u-aizu.ac.jp/challenges/sources/PCK/Final/0417?year=2019
To submit your code in Java to AOJ, note that the name of the class which includes the main method should be "Main", and you can have several classes in your code.

Submission Files	Types
`All classes you implemented`	`Java Class`

Summary

The concept of object references is one of the most important elements of object-oriented programming concepts. Did you understand how an object reference can be assigned as **a copy of a value** or passed to a function as **a copy of a value**?   We also learned how to manipulate arrays of objects through exercises in managing multiple queues.