July 21st, 2020

Exercises for Programming Practice 2

Note:
- ✓ Do not create “module-info.java”, when you create a Java Project.
- ✓ Do not set Package name in the window “New Java Class”.
- ✓ Do not use the title of exercise for “contents of the program”. Think about “contents of the program” yourself.

The deadline for submitting the programs is 18:00 on July 23rd, 2020.

The class “SetGraph.java” can be obtained from “Week 12” of Resource page, “Programming Practice 2” course, manaba+R.

Exercise 26 (file name “Graph.java”)

Create a Java program “Graph.java” to make an undirected graph using HashMap. The key is the name of the vertex and the value is an ArrayList that stores the vertices next to the vertex. In the constructor of “Graph.java”, an emplty ArrayList is set as a value for each key (the name of the vertex). There are two methods in “Graph.java” as follows:

- ✓ `public void addEdge(String v, String w)`
  When the names (v and w) of the two vertices are given, the adjacent vertex of each vertex is set.

- ✓ `public ArrayList<String> getNext(String node)`
  When the name of the vertex is given, ArrayList representing the neighboring vertex of the given vertex is returned.

The “SetGraph.java” shown in Fig. 1 creates the graph shown in Fig. 2 and displays the adjacent points of each vertex shown in Fig. 3. Use “SetGraph.java” to verify the “Graph.java” works correctly.

```java
import java.util.ArrayList;

public class SetGraph {
    String[] vertices= {"A", "B", "C", "E", "F", "G", "I", "J", "K"};

    public static void main(String[] args) {
        new SetGraph();
    }

    public SetGraph() {
        Graph gf = new Graph(vertices);

        gf.addEdge("A", "B");
        gf.addEdge("A", "E");
        gf.addEdge("A", "F");
        gf.addEdge("B", "C");
        gf.addEdge("B", "F");
        gf.addEdge("C", "G");
        gf.addEdge("E", "F");
        gf.addEdge("E", "I");
        gf.addEdge("F", "I");
        gf.addEdge("G", "J");
        gf.addEdge("G", "K");
        gf.addEdge("I", "J");
        gf.addEdge("J", "K");

        for(String node: vertices) {
            ArrayList<String> nodes = gf.getNext(node);
            System.out.println(node + " >> " + nodes);
        }
    }
}
```

Fig. 1 “SetGraph.java”.

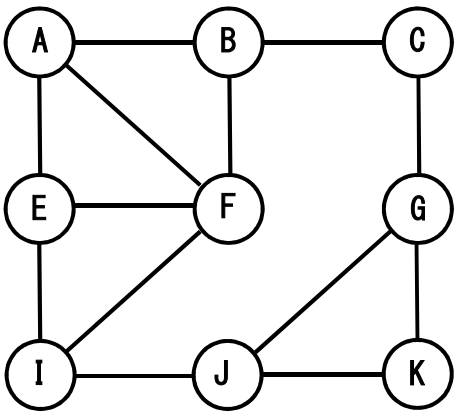

Fig. 2 The graph created by “SetGraph.java”.

The outline of “Graph.java” is as follows:

```
/* comments */

import java.util.ArrayList;
import java.util.HashMap;

public class Graph {
    HashMap<String, ArrayList<String>> vertex = new HashMap<String,
ArrayList<String>>();

    public Graph(String[] nodes) {
        for(String node: nodes) {
            vertex.put(node, new ArrayList<String>());
        }
    }
    public void addEdge(String v, String w) {

        // Complete this part

    }

    public ArrayList<String> getNext(String node){

        // Complete this part

    }
}
```

```
A >> [B, E, F]
B >> [A, C, F]
C >> [B, G]
E >> [A, F, I]
F >> [A, B, E, I]
G >> [C, J, K]
I >> [E, F, J]
J >> [G, I, K]
K >> [G, J]
```

Fig. 3 The expected output of “SetGraph.java”.

Exercise 27 (file name “Exercise27.java”)

Create a Java program “Exercise27.java” that implements the depth-first search algorithm shown in Fig. 4. The variable “gf” is substituted for an instance of the created graph. The variable “u” is the name of the vertex currently being processed. The variable “discovered” is an ArrayList that stores vertices that have already passed.

The outline of “Exercise27.java” is shown in Fig. 5. The process of creating a graph is the same as “SetGraph.java”.

```
public void dfs(Graph gf, String u, ArrayList<String> discovered)
(1) Initialize ArrayList<String> “nodes” at the vertices next to vertex “u”.
(2) Obtain Iterator<String> “iterator” to cycle through the contents of “nodes”.
(3) While there is an element in “iterator,” perform the followings:
    (a) Substitute the next element in “iterator” to the String variable “node”
    (b) If “discovered” does not contain “node”,
        ① Append “node” to “discovered”
        ② Display “node” name with a blank at the back
        ③ Call dfs(gf, node, discovered)
```

Fig. 4 The Depth-First Search Algorithm.

```
/* comments */

import java.util.ArrayList;
import java.util.Iterator;

public class Exercise27 {
    String[] vertices= {"A", "B", "C", "E", "F", "G", "I", "J", "K"};

    public static void main(String[] args) {
        new Exercise27();
    }

    public Exercise27() {
        ArrayList<String> discovered = new ArrayList<String>();
        String startNode = "A";
        Graph gf = new Graph(vertices);

        gf.addEdge("A", "B");
        gf.addEdge("A", "E");
        gf.addEdge("A", "F");
        gf.addEdge("B", "C");
        gf.addEdge("B", "F");
```

Fig. 5 The outline of Exercise27.java”.

```
        gf.addEdge("C", "G");
        gf.addEdge("E", "F");
        gf.addEdge("E", "I");
        gf.addEdge("F", "I");
        gf.addEdge("G", "J");
        gf.addEdge("G", "K");
        gf.addEdge("I", "J");
        gf.addEdge("J", "K");

        for(String node: vertices) {
            ArrayList<String> nodes = gf.getNext(node);
            System.out.println(node + " >> " + nodes);
        }

        discovered.add(startNode);
        System.out.print(startNode + " ");
        dfs(gf, startNode, discovered);
    }

    public void dfs(Graph gf, String u, ArrayList<String> discovered) {
        ArrayList<String> nodes = gf.getNext(u);
        Iterator<String> iterator = nodes.listIterator();
        while(iterator.hasNext()) {

            // Complete this part

        }
    }
}
```

Fig. 5 (continue).

Expected Output:

```
A >> [B, E, F]
B >> [A, C, F]
C >> [B, G]
E >> [A, F, I]
F >> [A, B, E, I]
G >> [C, J, K]
I >> [E, F, J]
J >> [G, I, K]
K >> [G, J]
A B C G J I E F K
```

Exercise 28 (file name “Exercise28.java”)

Create a Java program “Exercise28.java” that implements the breadth-first search algorithm shown in Fig. 6. The variable “gf” is substituted for an instance of the created graph. The variable “u” is the name of the vertex currently being processed. The variable “discovered” is an ArrayList that stores vertices that have already passed.

public void bfs(Graph gf, String u, ArrayList<String> discovered)

(1) Create ArrayList<String> “level”, and String variable “vertex”.

(2) Append “u” to ArrayList<String> “level”.

(3) While the size of “level” is greater than 0, perform the followings:

  (A) Display “level”.

  (B) Create ArrayList<String> “next_level”

  (C) Obtain Iterator<String> “iteratorL” to cycle through the contents of “level”

  (D) While there is an element in “iteratorL,” perform the followings:

    ① Initialize ArrayList<String> “nodes” at the vertices next to the element in “iteratorL”.

    ② Obtain Iterator<String> “iteratorN” to cycle through the contents of “nodes”

    ③ While there is an element in “iteratorN,” perform the followings:

      (a) Substitute the next element in “iteratorN” to the variable “vertex”

      (b) If “discovered” does not contain “vertex”

        ✓ Append “vertex” to “discovered”

        ✓ Append “vertex” to “next_level”

  (E) Substitute “next_level” to “level”

Fig. 6 The Breadth-First Search Algorithm.

The outline of “Exercise28.java” is as follows:

```
/* comments */

import java.util.ArrayList;
import java.util.Iterator;

public class Exercise28 {
    String[] vertices= {"A", "B", "C", "E", "F", "G", "I", "J", "K"};

    public static void main(String[] args) {
        new Exercise28();
    }

    public Exercise28() {
        ArrayList<String> discovered = new ArrayList<String>();
        String startNode = "A";
        Graph gf = new Graph(vertices);

        gf.addEdge("A", "B");
        gf.addEdge("A", "E");
        gf.addEdge("A", "F");
        gf.addEdge("B", "C");
        gf.addEdge("B", "F");
        gf.addEdge("C", "G");
        gf.addEdge("E", "F");
        gf.addEdge("E", "I");
        gf.addEdge("F", "I");
        gf.addEdge("G", "J");
        gf.addEdge("G", "K");
        gf.addEdge("I", "J");
        gf.addEdge("J", "K");

        for(String node: vertices) {
            ArrayList<String> nodes = gf.getNext(node);
            System.out.println(node + " >> " + nodes);
        }

        discovered.add(startNode);
        bfs(gf, startNode, discovered);
    }

    public void bfs(Graph gf, String u, ArrayList<String> discovered) {
        ArrayList<String> level = new ArrayList<String>();
        String vertex;

        // Complete this part

    }
}
```

Expected Output:

```
A >> [B, E, F]
B >> [A, C, F]
C >> [B, G]
E >> [A, F, I]
F >> [A, B, E, I]
G >> [C, J, K]
I >> [E, F, J]
J >> [G, I, K]
K >> [G, J]
[A]
[B, E, F]
[C, I]
[G, J]
[K]
```