I have done:

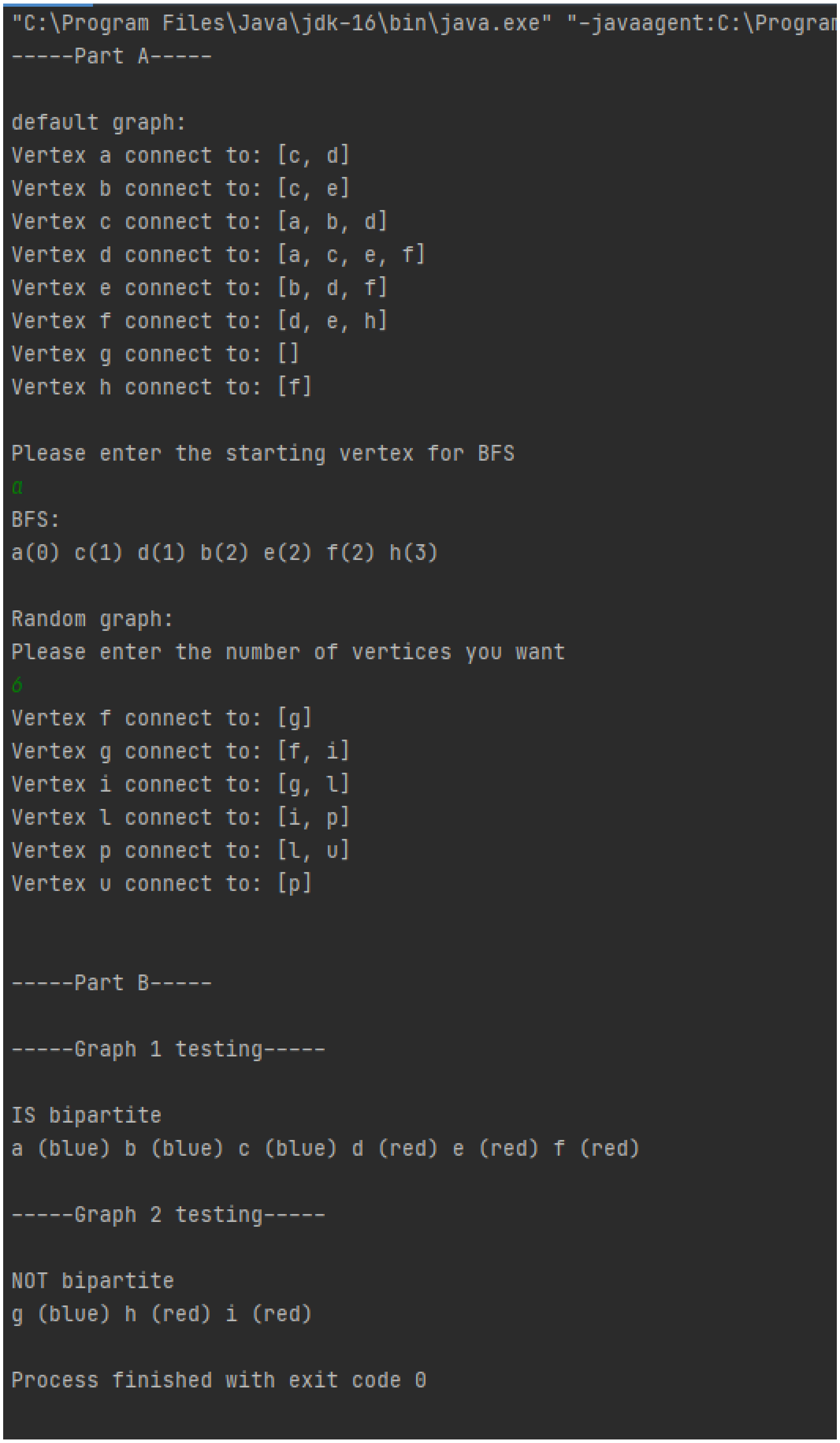
- Random Graph generator
- Implement graph using object of vertex and graph

```java
import java.util.*;
// Nhan Vo
// CECS 328- Lab#7
public class main {
   public static void main(String[] args){
       Scanner scan=new Scanner(System.in);
       //Part A:
       //Testing Graph
       System.out.println("-----Part A-----\n");
       System.out.println("default graph:");
       Graph graph=new Graph();
       Vertex a=new Vertex("a");
       Vertex b=new Vertex("b");
       Vertex c=new Vertex("c");
       Vertex d=new Vertex("d");
       Vertex e=new Vertex("e");
       Vertex f=new Vertex("f");
       Vertex g=new Vertex("g");
       Vertex h=new Vertex("h");

       graph.addVertex(a);
       graph.addVertex(b);
       graph.addVertex(c);
       graph.addVertex(d);
       graph.addVertex(e);
       graph.addVertex(f);
       graph.addVertex(g);
       graph.addVertex(h);

       graph.addEdge(a,c);
       graph.addEdge(a,d);
       graph.addEdge(b,c);
       graph.addEdge(b,e);
       graph.addEdge(c,d);
       graph.addEdge(d,e);
       graph.addEdge(d,f);
       graph.addEdge(f,e);
       graph.addEdge(f,h);

       graph.display();
```

```
        System.out.println("Please enter the starting vertex for
BFS");
        String n=scan.nextLine();
        Vertex input=null;
        outterloop:
        while(true){
            for(Vertex i:graph.getVertexSet()){
                if(i.getKey().equals(n)){
                    input=i;
                    break outterloop;
                }
            }
            System.out.println("Please enter the starting vertex for
BFS");
            n=scan.nextLine();
        }
        System.out.println("BFS:");
        graph.BFS(input);

        //Auto generating random graph from given number of vertices
        System.out.println("\nRandom graph:");
        System.out.println("Please enter the number of vertices you
want");
        String ver=scan.nextLine();
        while(!ver.matches("\\d+")){
            System.out.println("Your input is not an approriate
integer, Please enter the number of vertices you want again:");
            ver = scan.nextLine();
        }

        Graph newGraph=new Graph(Integer.parseInt(ver));
        newGraph.display();

        //Part B:
        System.out.println("\n-----Part B-----");
        System.out.println("\n-----Graph 1 testing-----");
        Graph bipartite= new Graph();

        Vertex a1=new Vertex("a");
        Vertex b1=new Vertex("b");
        Vertex c1=new Vertex("c");
        Vertex d1=new Vertex("d");
        Vertex e1=new Vertex("e");
        Vertex f1=new Vertex("f");
        bipartite.addVertex(a1);
        bipartite.addVertex(b1);
        bipartite.addVertex(c1);
        bipartite.addVertex(d1);
        bipartite.addVertex(e1);
        bipartite.addVertex(f1);

        bipartite.addEdge(a1,d1);
        bipartite.addEdge(c1,d1);
        bipartite.addEdge(c1,e1);
        bipartite.addEdge(b1,d1);
        bipartite.addEdge(b1,f1);

        bipartite.Explore();// Expected bipartite

        System.out.println("\n-----Graph 2 testing-----");
        Graph Notbipartite=new Graph();

        Vertex g1=new Vertex("g");
        Vertex h1=new Vertex("h");
        Vertex i1=new Vertex("i");

        Notbipartite.addVertex(g1);
        Notbipartite.addVertex(h1);
        Notbipartite.addVertex(i1);

        Notbipartite.addEdge(g1,h1);
        Notbipartite.addEdge(h1,i1);
        Notbipartite.addEdge(g1,i1);
        Notbipartite.Explore();// Expected NOT bipartite

    }

}

class Graph{
   private ArrayList<Vertex> vertexSet=new ArrayList<>();
   private int countEdge=0;


   public Graph(){
       //default constructor will create an empty graph
   }

   //Auto generate random graph constructor
```

```
    public Graph(int ver){
        countEdge=(int) ver*((ver-1)/2);// compute maximum number of
edges;
        String alphabet="abcdefgh";
        Random ran=new Random();
        char ch=alphabet.charAt(ran.nextInt(alphabet.length()));
        for(int i=1;i<ver+1;i++){
            this.addVertex(new Vertex(Character.toString(ch)));
            ch+=i;
        }
        for(int i=0;i<countEdge;i++){
            if(i<vertexSet.size()-1){
                this.addEdge(vertexSet.get(i), vertexSet.get(i + 1));
            }
        }
    }

    public void display(){

        String output="";
        for(int i=0;i<vertexSet.size();i++){
            output+="Vertex "+vertexSet.get(i).getKey()+" connect to:
"+vertexSet.get(i).getAdj()+"\n";
        }
        System.out.println(output);
    }

    //add vertex into a graph
    public void addVertex(Vertex v){

        vertexSet.add(v);

    }

    // addEdge method for undirected graph
    public void addEdge(Vertex source, Vertex sink){
        if(source.isNeighbor(sink) || sink.isNeighbor(source)){
            //do nothing since the edge already existed
        }
        else{
            source.addNeighbor(sink);
            sink.addNeighbor(source);
            countEdge++;
        }
    }

    public int getOrder(){

        return vertexSet.size();
    }

    public ArrayList<Vertex> getVertexSet(){
        return vertexSet;
    }

    public int getSize(){

        return countEdge;
    }
    //part A:
    public void BFS(Vertex start){
        Queue<Vertex> queue=new LinkedList<>();
        Vertex next_v=null;
        start.distance=0;
        start.parent=-1;
        queue.add(start);
        while(queue.size()>0){
            next_v=queue.poll();
            System.out.print(next_v+"("+next_v.distance+") ");
            for(int i=0;i<next_v.getAdj().size();i++){
                if(next_v.getAdj().get(i).parent==null){
                    queue.add(next_v.getAdj().get(i));
                    next_v.getAdj().get(i).parent=next_v;
                    next_v.getAdj().get(i).distance=
next_v.distance+1;
                }
            }
        }
        System.out.println(" ");
    }
    //part B:
    public void Explore(){
        for(Vertex i: vertexSet){
            i.color="gray";
        }
        vertexSet.get(0).color="blue";
        Is_bipartite( vertexSet.get(0));
        for(Vertex i:vertexSet){
            if(i.color.equals("gray")){
                Is_bipartite(i);
            }
        }

        for(Vertex i: vertexSet){
            System.out.print(i+" ("+i.color+")"+" ");
        }
        System.out.println(" ");

    }

    public void Is_bipartite(Vertex ve){
        Queue<Vertex> queue=new LinkedList<>();
        Vertex u=null;
        queue.add(ve);
        boolean isbipartite=true;
        outterloop:
        while(queue.size()>0) {
            u = queue.poll();
            //System.out.print(u+"("+u.color+") ");
            for (Vertex v : u.getAdj()) {
                if (v.color.equals("gray")) {
                    if (u.color.equals("blue")) {
                        v.color = "red";
                    } else if (u.color.equals("red")) {
                        v.color = "blue";
                    }
                    //setColor(v);
                    queue.add(v);

                } else if (u.color.equals(v.color)) {
                    //System.out.println(u+" ("+u.color+")"+" | "+ v+"
("+v.color+
                    System.out.println(" ");
                    System.out.println("NOT bipartite");
                    isbipartite=false;
                    break outterloop;

                }
            }
        }
        if(isbipartite!=false){
            System.out.println(" ");
            System.out.println("IS bipartite");
        }
    }
 }

 class Vertex{
    private String Key;
    private LinkedList<Vertex> adj= new LinkedList<>();
    public int distance;
    public Object parent=null;
    public String color=null;
    // default constructor that take a string as key for the vertex
    public Vertex(String k){

        Key=k;
    }

    public int getDegree(){

        return adj.size();
    }

    public boolean isNeighbor(Vertex v){
        for(int i=0;i<adj.size();i++){
            if(adj.get(i).getKey()==v.getKey()){
                return true;
            }
        }
        return false;
    }

    public String getKey(){

        return Key;
    }

    public void addNeighbor(Vertex v){
        adj.add(v);
    }

    public LinkedList<Vertex> getAdj(){

        return adj;
    }

    public String toString(){
        return Key;
    }

}
```

Output screenshot:



```
"C:\Program Files\Java\jdk-16\bin\java.exe" "-javaagent:C:\Progran
-----Part A-----

default graph:
Vertex a connect to: [c, d]
Vertex b connect to: [c, e]
Vertex c connect to: [a, b, d]
Vertex d connect to: [a, c, e, f]
Vertex e connect to: [b, d, f]
Vertex f connect to: [d, e, h]
Vertex g connect to: []
Vertex h connect to: [f]

Please enter the starting vertex for BFS
a
BFS:
a(0) c(1) d(1) b(2) e(2) f(2) h(3)

Random graph:
Please enter the number of vertices you want
6
Vertex f connect to: [g]
Vertex g connect to: [f, i]
Vertex i connect to: [g, l]
Vertex l connect to: [i, p]
Vertex p connect to: [l, u]
Vertex u connect to: [p]


-----Part B-----

-----Graph 1 testing-----

IS bipartite
a (blue) b (blue) c (blue) d (red) e (red) f (red)

-----Graph 2 testing-----

NOT bipartite
g (blue) h (red) i (red)

Process finished with exit code 0
```