```
/ Author: Jose Enriquez
// Email: jose.enriquez@okstate.edu
// 4/30/2021
// Graph used to in creating a simulated social network

import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;
import java.util.*;

public class Graph {

    // add User as key, and adjacent Users as values in set
    private HashMap<User, Set<User>> adjacencyLists = new HashMap<>();

    public void addVertex(User user) {
        if (this.adjacencyLists.containsKey(user)) {
            throw new IllegalArgumentException();
        }
        this.adjacencyLists.put(user, new HashSet<>());

    }


    /**
     * Adds edge from source to destination
     * @param source of edge
     * @param destination of egde
     * @return true if edge was added.
     */
    public boolean addEdge(User source, User destination) {
        if (!this.adjacencyLists.containsKey(source) ||
!this.adjacencyLists.containsKey(destination)) {
            //One or both of the parameters do not exist.
            throw new IllegalArgumentException("One or both of the parameters do not
exist.");
        }
        if(this.adjacencyLists.get(source).contains(destination)){
            //Source is already connected to destination.
            throw new IllegalArgumentException("Source is already connected to
destination.");
        }

        //add edge from source to destination
        return this.adjacencyLists.get(source).add(destination);
    }

    /**
     * gets the followers of a user.
     * @param User to get the followers of.
     * @return HashSet<User> container of followers
     */
    public HashSet<User> getFolowers(User name){
        HashSet<User> followers = new HashSet<User>();
        Iterator<Map.Entry<User, Set<User>>> iter =
adjacencyLists.entrySet().iterator();

        while(iter.hasNext()){
            //Iterates through each entry in the map.
            Map.Entry<User, Set<User>> entry = (Map.Entry<User,
Set<User>>)iter.next();
            if(entry.getValue().contains(name)){
                //A vertex contains an edge to the user.
                //someone is following th user.
                followers.add(entry.getKey());
            }
        }
        return followers;
    }

    /**
     * gets the users a given user is following.
     * @param name
     * @return HashSet<User> container of Users the given user is following.
     */
    public HashSet<User> getFollowing(User name) {
        return new HashSet<User>(this.adjacencyLists.get(name));
    }

    /**
     * @return true if there is an edge from source -> destination
     */
    public boolean isFollower(User source, User destination) {
        if (!this.adjacencyLists.containsKey(source) ||
!this.adjacencyLists.containsKey(destination)) {
            throw new IllegalArgumentException();
        }
        return this.adjacencyLists.get(source).contains(destination);
    }

    /**
     * Remove an edge from a source to a destination
     * @param source source of an edge
     * @param destination destination of an edge
     * @return true if edge was removed.
     */
    public boolean removeEdge(User source, User destination) {
        if (!this.adjacencyLists.containsKey(source)) {
            return false;
```

```java
        }

        return this.adjacencyLists.get(source).remove(destination);
    }

    /**
     * Removes a given vertex from the graph.
     * @param vertex The vertex to remove
     * @return true if the vertex was found and removed
     */
    public boolean removeVertex(User vertex) {
        if (!this.adjacencyLists.containsKey(vertex)) {
            //vertex was not found
            return false;
        }

        //remove vertex from graph
        this.adjacencyLists.remove(vertex);

        for ( Set<User> adjacencies : this.adjacencyLists.values()) {
            //remove edges to the vertex.
            adjacencies.remove(vertex);
        }
        return true;
    }

    /**
     * removes all vertexes and edges from graph
     */
    public void clear() {
        this.adjacencyLists.clear();
    }

    /**
     * @return int the number of vertexes
     */
    public int size() {
        return this.adjacencyLists.size();
    }

    /**
     * Checks if the graph is empty or not
     * @return true if empty
     */
    public boolean isEmpty() {
        return this.adjacencyLists.isEmpty();
    }

    //used to test
    public static void main(String[] args) {
        User a = new User("jo", "A1212");
        Graph graph = new Graph();
        graph.addVertex(a);
        System.out.println(graph.adjacencyLists.containsKey(a));

        User b = new User("jo", "A2222");
        graph.addVertex(b);
        System.out.println(graph.adjacencyLists.containsKey(b));

        graph.addEdge(a, b);

    }
}
```