1. Да се имплементира следниот метод кој го враќа бројот на појавување на стрингот `str` во колекцијата од колекција од стрингови:

```
public static int count(Collection<Collection<String>> c, String
    str)
```

   а. Да претпоставиме дека `Collection c` содржи `N` колекции и дека секоја од овие колекции содржи N објекти. Кое е времето на извршување на вашиот метод?

   б. Да претпоставиме дека е потребно 2 милисекунди да се изврши за $N = 100$. Колку ќе биде времето на извршување кога $N = 300$?

```
public static int count(Collection<Collection<String>> c,
    String str) {
    int count = 0;
    for (Collection<String> sub : c) {
        for (String s : sub) {
            if (s.equals(str)) {
                ++count;
            }
        }
    }
    return count;
}
```

2. Да се напише метод за печатење на колекција во обратен редослед со помош на `Collections API` но без употреба на `ListIterator`.

```
public static <T> void printReverse(Collection<? extends T>
    collection) {
    int size = collection.size();
    Object[] array = collection.toArray();
    for (int i = size - 1; i >= 0; --i) {
        System.out.println(array[i]);
    }
}
```

3. Методот `equals` кој е прикажан, враќа `true` ако две листи имаат иста големина и ако ги содржат истите елементи во ист редослед. Да претпоставиме дека `N` е големината на овие лист.

```
public boolean equals(List<Integer> lhs, List<Integer> rhs) {
    if(lhs.size( ) != rhs.size())
        return false;
    for(int i = 0; i < lhs.size(); i++)
        if(!lhs.get(i).equals(rhs.get(i))
            return false;
    return true;
}
```

   а. Кое е времето на извршување ако двете листи се `ArrayLists`?

   б. Кое е времето на извршување ако двете листи се `LinkedLists`?

в. Да претпоставиме дека за 4 секунди се извршува методот за две еднакво големи поврзани листи `LinkedList` со 10.000 елементи. Колку време ќе биде потребно за извршување со две еднакво големи поврзани листи со 50.000 елементи?

г. Објаснете со една реченица како да го направиме алгоритмот поефикасен за сите видови листи?

4. Методот `hasSpecial`, прикажан подолу, враќа `true` ако постојат два уникатни броеви во листата чија што сума е еднаква на некој трет број во листата. Да претпоставиме дека `N` е големината на листата.

```
// Returns true if two numbers in c when added together sum //
    to a third number in c.
public static boolean hasSpecial( List<Integer> c ) {
    for(int i = 0; i < c.size(); i++)
        for(int j = i + 1; j < c.size(); j++ )
            for(int k = 0; k < c.size(); k++ )
                if(c.get(i) + c.get(j) == c.get(k))
                    return true;
    return false;
}
```

а. Кое е времето на извршување ако листата е `ArrayList`?

б. Кое е времето на извршување ако листата е `LinkedList`?

в. Ако се потребни 2 секунди за извршување со листа со 1000 елементи, колку време ќе биде потребно за извршување на 3000 елементи. Да претпоставиме дека и во двата случаи методот враќа резултат `false`.

5. Ситото на Ерастотен (The Sieve of Erastothenes) е древен алгоритам за генерирање прости броеви. Да ја земеме следната листа со броеви:

```
2 3 4 5 6 7 8 9 10
```

Алгоритмот започнува со првиот прост број во листата, тоа е 2 и потоа ги изминува останатите елементи од листата, со тоа што ги отстранува сите броеви чии што множител е 2 (во овој случај, 4, 6, 8 и 10), со што остануваат 2 3 5 7. Го повторуваме овој процес со вториот прост број во листата, тоа е 3 и итерираме низ остатокот од листата и што ги отстрануваме броевите чии што множител е 3 (во овој случај 9), а остануваат 2 3 5 7.

Потоа ја повторуваме постапката со секој следен прост број, но не се отстрануваат елемнти, затоа што нема броеви чии што множители се 5 и 7. Сите броеви кои остануваат во листата се прости.

Да се имплементира алгоритмот со користење на `ArrayList` од цели броеви која што е иницијализирана со вредности од 2 до 100. Имплементацијата може да итерира низ листата со индекс од 0 од `size() - 1` за да го земе тековниот прост број, но треба да користи `Iterator` за да го скенира остатокот од листата и ги избрише сите елементи чии што множител е тековниот прост број. Отпечатете ги сите прости броеви во листата.

```
package edu.finki.np.av6;

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

```
import java.util.List;

public class ErastothenesSieve {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<Integer>();
        for (int i = 2; i <= 100; ++i) {
            list.add(i);
        }
        for (int i = 0; i < list.size(); ++i) {
            Iterator<Integer> iterator = list.listIterator(i +
                1);
            while (iterator.hasNext()) {
                if (iterator.next() % list.get(i) == 0) {
                    iterator.remove();
                }
            }
        }

        for (int i = 0; i < list.size(); ++i) {
            System.out.printf("%d ", list.get(i));
        }
    }
}
```

6. Парадоксот на роденден е дека постои изненадувачки голема веројатност дека двајца луѓе во иста просторија имаат ист роденден. Под ист роденден подразбираме дека се родени на ист ден во годината (ги игнорираме преспните). Да се напише програма која апроксимативно ја пресметува веројатноста дека 2-ца или повеќе луѓе во иста соба имаат ист роденден, за 2 до 50 луѓе во просторија.

   Програмата треба да го пресмета одговорот со симулација. Во повеќе обиди (пр. 5000), се доделува случаен роденден (пр. број од 1 - 365) на сите во собата. Роденденте се чуваат во `HashSet`. Како што се генерираат случајни родендени, со помош на методот `contains` на `HashSet` се проверува дали тој роденден е веќе назначен во собата. Ако е тоа случај, се зголемува еден бројач кој брои колку пати барем 2-ца во собата имаат ист роденден и се преминува на следниот обид.

   Излезот од вашата програма треба да биде во следниот облик. Броевите нема да бидат сосема точни затоа што се употребуваат случајни броеви.

```
For 2 people, the probability of two birthdays is about 0.002
For 3 people, the probability of two birthdays is about 0.0082
For 4 people, the probability of two birthdays is about 0.0163
...
For 49 people, the probability of two birthdays is about 0.9654
For 50 people, the probability of two birthdays is about 0.969
```

```
package edu.finki.np.av6;

import java.util.HashSet;
import java.util.Random;
import java.util.Set;

public class Birthdays {
    static final int NUM_TRIALS = 5000;
```

```
    public static void main(String[] args) {
        Random rand = new Random();

        for (int numPeople = 2; numPeople <= 50; ++numPeople) {
            int positiveEvents = 0;
            for (int j = 0; j < NUM_TRIALS; ++j) {
                int sameBirthday = 1;
                Set<Integer> room = new HashSet<Integer>();
                for (int i = 0; i < numPeople; ++i) {
                    int birthday = rand.nextInt(365) + 1;
                    if (!room.add(birthday)) {
                        sameBirthday++;
                    }
                }
                if (sameBirthday == 5) {
                    positiveEvents++;
                }
            }
            System.out.printf("Probability: %d : %f\n",
                numPeople, positiveEvents * 1.0
                    / NUM_TRIALS);
        }
    }
}
```

7. The text files `boynames.txt` and `girlnames.txt`, which are included in the source code for this book, contain lists of the 1,000 most popular boy and girl names in the United States for the year 2005, as compiled by the Social Security Administration. These are blank-delimited files where the most popular name is listed first, the second most popular name is listed second, and so on to the 1,000th most popular name, which is listed last. Each line consists of the first name followed by a blank space followed by the number of registered births in the year using that name. For example, the `girlnames.txt` file begins with

   ```
   Emily 25494
   Emma 22532
   ```

   This indicates that Emily is the most popular name with 25,494 registered namings, Emma is the second most popular with 22,532, and so on. Write a program that determines how many names are on both the boys' and the girls' list. Use the following algorithm:

   - Read each girl name as a `String`, ignoring the number of namings, and add it to a `HashSet` object.
   - Read each boy name as a `String`, ignoring the number of namings, and add it to the same `HashSet` object. If the name is already in the `HashSet`, then the add method returns false. If you count the number of false returns, then this gives you the number of common namings.
   - Add each common name to an `ArrayList` and output all of the common names from this list before the program exits.

   Repeat the previous problem except create your own class, `Name`, that is added to a `HashMap` instead of a `HashSet`. The `Name` class should have three private variables, a `String` to store the name, an integer to store the number of namings for girls,

and an integer to store the number of namings for boys. Use the first name as the key to the `HashMap`. The value to store is the `Name` object. Instead of ignoring the number of namings, as in the previous project, store the number in the `Name` class. Make the `ArrayList` a list of `Name` objects; each time you find a common name, add the entire `Name` object to the list. Your program should then iterate through the `ArrayList` and output each common name, along with the number of boy and girl namings.

```
package edu.finki.np.av6;

import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Scanner;

public class Names {
    public static void main(String[] args) {
        Scanner fileScanner = null;
        Map<String, Name> names = new HashMap<String, Name>();
        List<Name> repeating = new ArrayList<Name>();
        try {
            fileScanner = new Scanner(new FileReader("girlnames.
                txt"));
            while (fileScanner.hasNextLine()) {
                String line = fileScanner.nextLine();
                String[] parts = line.split(" ");
                String name = parts[0];
                int count = Integer.parseInt(parts[1]);
                Name nameObject = new Name(name, count, 0);
                names.put(name, nameObject);
            }
            fileScanner.close();
            fileScanner = new Scanner(new FileReader("boynames.
                txt"));
            while (fileScanner.hasNextLine()) {
                String line = fileScanner.nextLine();
                String[] parts = line.split(" ");
                String name = parts[0];
                int count = Integer.parseInt(parts[1]);
                if (names.containsKey(name)) {
                    Name nameObject = names.get(name);
                    nameObject.setCountBoys(count);
                    repeating.add(nameObject);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        for (Name name : repeating) {
            System.out.println(name);
        }
    }
}
```

```
class Name {
    String name;
    int countGirls;
    int countBoys;

    public Name(String name, int countGirls, int countBoys) {
        this.name = name;
        this.countGirls = countGirls;
        this.countBoys = countBoys;
    }

    public void setCountBoys(int countBoys) {
        this.countBoys = countBoys;
    }

    public String getName() {
        return name;
    }

    @Override
    public String toString() {
        return String.format("%s %d %d", name, countGirls,
            countBoys);
    }

}
```

8. In an ancient land, the beautiful princess Eve had many suitors. She decided on the following procedure to determine which suitor she would marry. First, all of the suitors would be lined up one after the other and assigned numbers. The first suitor would be number 1, the second number 2, and so on up to the last suitor, number n. Starting at the first suitor, she would then count three suitors down the line (because of the three letters in her name) and the third suitor would be eliminated from winning her hand and removed from the line. Eve would then continue, counting three more suitors, and eliminating every third suitor. When she reached the end of the line, she would reverse direction and work her way back to the beginning. Similarly, on reaching the first person in line, she would reverse direction and make her way to the end of the line. For example, if there were five suitors, then the elimination process would proceed as follows:

```
12345 Initial list of suitors; start counting from 1.
1245 Suitor 3 eliminated; continue counting from 4 and bounce from end
back to
Suitor 4 eliminated; continue counting back from 2 and bounce from
front back to 2.
15
Suitor 2 eliminated; continue counting forward from 5.
1
Suitor 5 eliminated; 1 is the lucky winner.
```

Write a program that uses an `ArrayList` or `Vector` to determine which position you should stand in to marry the princess if there are n suitors. Your program should use the ListIterator interface to traverse the list of suitors and remove a suitor. Be careful that your iterator references the proper object upon reversing direction at

the beginning or end of the list. The suitor at the beginning or end of the list should only be counted once when the princess reverses the count.

```
package edu.finki.np.av6;

import java.util.ArrayList;
import java.util.List;
import java.util.ListIterator;

public class LuckySuitor {
    public static int getWinner(int n) {
        List<Integer> positions = new ArrayList<Integer>();
        for (int i = 0; i < n; ++i) {
            positions.add(i + 1);
        }
        ListIterator<Integer> listIterator = positions.
            listIterator();
        boolean toRight = true;
        while (positions.size() != 1) {
            int last = -1;
            for (int i = 0; i < 3; ++i) {
                if (listIterator.hasNext() && toRight) {
                    last = listIterator.next();
                    if (!listIterator.hasNext()) {
                        toRight = false;
                        listIterator.previous();
                    }
                    //System.out.println("->: " + last);
                } else {
                    if (listIterator.hasPrevious()) {
                        last = listIterator.previous();
                        if (!listIterator.hasPrevious()) {
                            toRight = true;
                            listIterator.next();
                        }
                        //System.out.println("<-: " + last);
                    }
                }
            }
            //System.out.println("Remove: " + last);
            listIterator.remove();
            //System.out.println("DIR: " + (toRight ? "->" :
                "<-"));
        }
        return positions.get(0);
    }

    public static void main(String[] args) {
        System.out.println("Winner: " + LuckySuitor.getWinner(5)
            );
    }
}
```

9. In social networking websites, people link to their friends to form a social network. Write a program that uses `HashMaps` to store the data for such a network. Your program should read from a file that specifies the network connections for different usernames. The file should have the following format to specify a link:

`source_username friend_username`

There should be an entry for each link, one per line. Here is a sample file for five usernames:

```
iba java_guru
iba crisha
iba ducky
crisha java_guru
crisha iba
ducky java_guru
ducky iba
java_guru iba
java_guru crisha
java_guru ducky
wittless java_guru
```

In this network, everyone links to `java_guru` as a friend. `iba` is friends with `java_guru`, `crisha`, and `ducky`. Note that links are not bidirectional; wittless links with `java_guru` but `java_guru` does not link with wittless.

First, create a `User` class that has an instance variable to store the user's name and another instance variable that is of type `HashSet<User>`. The `HashSet<User>` variable should contain references to the User objects that the current user links to. For example, for the user `iba` there would be three entries, for `java_guru`, `crisha`, and `ducky`. Second, create a `HashMap<String,User>` instance variable in your main class that is used to map from a username to the corresponding `User` object. Your program should do the following:

- Upon startup, read the data file and populate the HashMap and HashSet data structures according to the links specified in the file.
- Allow the user to enter a name.
- If the name exists in the map, then output all usernames that are one link away from the user entered.
- If the name exists in the map, then output all usernames that are two links away from the user entered. To accomplish this in a general way, you might consider writing a recursive subroutine.

Do not forget that your `User` class must override the `hashCode` and `equals` methods.

```java
package edu.finki.np.av6;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Scanner;
import java.util.Set;

public class SocialNetwork {
    private Map<String, User> users;
```

```
    public SocialNetwork() {
        this.users = new HashMap<String, User>();
    }

    public void read(InputStream inputStream) {
        Scanner scanner = new Scanner(inputStream);
        while (scanner.hasNext()) {
            String line = scanner.nextLine();
            String[] parts = line.split(" ");
            String name1 = parts[0];
            String name2 = parts[1];
            User first = users.get(name1);
            User second = users.get(name2);
            if (first == null) {
                first = new User(name1);
                users.put(name1, first);
            }
            if (second == null) {
                second = new User(name2);
                users.put(name2, second);
            }
            first.addFriend(second);
        }
        scanner.close();
    }

    public void findFriends(String name, int n) {
        if (this.users.containsKey(name)) {
            this.users.get(name).findFriends(n);
        }
    }

    public void print() {
        for (String key : this.users.keySet()) {
            System.out.println("USER: " + key);
            System.out.println("FRIENDS: ");
            User u = this.users.get(key);
            for (User user : u.getFriends()) {
                System.out.println(user);
            }
        }
    }

    public static void main(String[] args) throws
        FileNotFoundException {
        InputStream inputStream = new FileInputStream(new File("
            friends.txt"));
        SocialNetwork socialNetwork = new SocialNetwork();
        socialNetwork.read(inputStream);
        // socialNetwork.print();
        Scanner scanner = new Scanner(System.in);
        String name = scanner.nextLine();
        socialNetwork.findFriends(name, 2);
    }
}
```

```
class User {
    private String name;
    private Set<User> friends;
    private static Set<String> found;

    public User(String name) {
        this.name = name;
        this.friends = new HashSet<User>();
    }

    public boolean addFriend(User friend) {
        return this.friends.add(friend);
    }

    public void findFriends(int n) {
        User.found = new HashSet<String>();
        User.found.add(this.name);
        this.findFriendsOfFriends(this.name, n);
    }

    private void findFriendsOfFriends(String name, int n) {
        if (n == 0) {
            for (User user : this.friends) {
                if (!User.found.contains(user.name)) {
                    System.out.println(user);
                }
            }
            return;
        }
        for (User user : this.friends) {
            if (!User.found.contains(user.name)) {
                user.findFriendsOfFriends(name, n - 1);
                User.found.add(user.name);
            }
        }
    }

    public Set<User> getFriends() {
        return this.friends;
    }

    @Override
    public String toString() {
        return this.name;
    }
}
```