Lab 12 List Client

Goal

In this lab you will complete two applications that use the Abstract Data Type (ADT) list.

Resources

- Chapter 12: Lists
- *docs.oracle.com/javase/8/docs/api/* —API documentation for the Java `List` interface

In `javadoc` directory

- *ListInterface.html*—Interface documentation for the interface `ListInterface`

Java Files

- *AList.java*
- *CountingGame.java*
- *ListInterface.java*
- *Primes.java*

Introduction

The ADT list is one of the basic tools for use in developing software applications. It is an ordered collection of objects that can be accessed based on their position. Before continuing the lab you should review the material in Chapter 12. In particular, review the documentation of the interface *ListInterface.java*. While not all of the methods will be used in our applications, most of them will.

The first application you will complete implements a child's selection game. In the standard version of this game, a group of children gather in a circle and begin a rhyme. (One such rhyme goes "Engine, engine number nine, going down Chicago line. If the train should jump the track, will you get your money back? My mother told me to pick the very best one and you are not it.") Each word in the rhyme is chanted in turn by one person in the circle. The last person is out of the game and the rhyme is restarted from the next person. Eventually, one person is left and he or she is the lucky individual that has been selected. This application will read the number of players in the game and the rhyme from the keyboard. The final output will be the number of the selected person. You will use two lists in this application. The first will be a list of numbers representing the players in the game. The second will be a list of words in the rhyme.

The second application is one that computes prime numbers using the Sieve of Eratosthenes.

Pre-Lab Visualization

Counting Game

Suppose we have six players named 1, 2, 3, 4, 5, and 6. Further, suppose the rhyme has three words A, B, C. There will be five rounds played with one player eliminated in each round. In the following table, fill in the players next to the part of the rhyme they say in each round. Cross out the players as they are eliminated. The first round has been completed for you.

	Round 1	Round 2	Round 3	Round 4	Round 5
A	1				
B	2				
C	3				

 Eliminated Players: 1 2 ~~3~~ 4 5 6

Suppose we have five players named 1, 2, 3, 4, and 5. And the rhyme has six words A, B, C, D, E, F. There will be four rounds played. In the following table, fill in the players next to the part of the rhyme they say in each round. Cross out the players as they are eliminated.

	Round 1	Round 2	Round 3	Round 4
A				
B				
C				
D				
E				
F				

 Eliminated Players: 1 2 3 4 5

Primes

Suppose you are interested in finding the primes between 2 and 15 inclusive. The list of candidates is:

Candidates: 2 3 4 5 6 7 8 9 10 11 12 13 14 15

The algorithm proceeds in rounds. In each round a single prime is discovered and numbers that have that prime as a factor are eliminated.

Round 1:

Cross the first value off of the Candidates list and add it to the Primes list. Cross out any candidate that is divisible by the prime you have just discovered and add it to the composites list.

 *Primes*:

 *Composites*:

Round 2:

To make the operation of the algorithm clearer, copy the contents of the lists as they appear at the end of the previous round.

Cross the first value off of the Candidates list and add it to the Primes list. Cross out any candidate that is divisible by the prime you have just discovered and add it to the composites list.

 *Candidates*:

 *Primes*:

 *Composites*:

Round 3:

Again, copy the contents of the lists as they appear at the end of the previous round.

Cross the first value off of the Candidates list and add it to the Primes list. Cross out any candidate that is divisible by the prime you have just discovered and add it to the composites list.

 *Candidates*:

 *Primes*:

 *Composites*:

You can complete the other rounds if you wish, but most of the interesting work has been completed.

Finding Composites

The heart of this algorithm is removing the composite values from the candidates list. Let's examine this process more closely. In the first round, after removing the 2, the list of candidates is

Candidates: 3 4 5 6 7 8 9 10 11 12 13 14 15

How many values are in the list?

The first value to be examined is the 3. What is its index?

The second value to be examined is the 4. What is its index?

Since 4 is divisible by 2, we need to remove it from the list of candidates. What is the new list of candidates after the 4 is removed?

 *Candidates*:

The third value to be examined is the 5. What is its index?

The fourth value to be examined is the 6. What is its index?

Since 6 is divisible by 2, we need to remove it from the list of candidates. What is the new list of candidates after the 6 is removed?

 *Candidates*:

The fifth value to be examined is the 7. What is its index?

Can you simply loop over the indices from 1 to 13 to examine all the candidates?

Develop an algorithm to examine all the values in the candidates list and remove them if they are divisible by the given prime.

Directed Lab Work

Counting Game

Pieces of the `CountingGame` class already exist and are in *`CountingGame.java`*. Take a look at that code now if you have not done so already. Also before you start, make sure you are familiar with the methods available to you in the `AList` class (check *`ListInterface.html`*).

Step 1. Compile the classes `CountingGame` and `AList`. Run the `main` method in `CountingGame`.

Checkpoint: If all has gone well, the program will run and accept input. It will then generate a null pointer exception. The goal now is to create the list of players.

Step 2. Create a new `Alist<Integer>` and assign it to `players`.

Step 3. Using a loop, add new objects of type `Integer` to the players list.

Checkpoint: Compile and run the program. Enter 3 for the number of players. The program should print out { <1> <2> <3> } for the players list. The next goal is to do one round of the game. It will be encapsulated in the method doRhyme().

Step 4. Complete the `doRhyme()` method. Use the following algorithm.

For each word in the rhyme
    Print the word in the rhyme and the player that says it.
Print the name of the player to be removed.
Remove that player from the list.
Return the index of the player that will start the next round.

Step 5. `Call doRhyme(players, rhyme, position)` in `main` after the call to `getRhyme()`.

Step 6. Print out the new player list.

Checkpoint: Compile and run the program. Enter 6 for the number of players. Enter A B C for the rhyme. It should print out something similar to

Player 1: A
Player 2: B
Player 3: C
Removing player 3
The players list is { <1> <2> <4> <5> <6> }

Enter 5 for the number of players. Enter A B C D E F for the rhyme. Compare your result with your answers in the pre-lab. Reconcile any differences. The final goal is to do multiple rounds.

Step 7. Wrap the lines of code from the previous two steps in a `while` loop that continues as long as there is more than one player left.

Final checkpoint: Compile and run the program. Enter 6 for the number of players. Enter A B C for the rhyme. The players should be removed in the order 3, 6, 4, 2, 5. The winner should be player 1.

Enter 5 for the number of players. Enter A B C D E F for the rhyme. Compare your result with your answers in the pre-lab exercises. Reconcile any differences.

Primes

The skeleton of the `Primes` class already exists and is in *`Primes.java`*.

Step 1. Look at the skeleton in *`Primes.java`*. Compile `Primes`. Run the `main` method in `Primes`.

Checkpoint: If all has gone well, the program will run and accept input. It will then end. The goal now is to create the list of candidates.

Step 2. In `main` declare and create the candidates list. Add in the values.

Step 3. Print out the candidates list.

Checkpoint: Compile and run the program. Enter 7 for the maximum value. You should see the list { <2> <3> <4> <5> <6> <7> }. The next goal is to do a single round finding a prime in the candidates list.

Step 4. In `main` declare and create the primes and composites lists.

Step 5. Remove the first value from the primes list and remember it in a variable.

Step 6. Print out the prime that was discovered.

Step 7. Add it to the primes list.

Step 8. Print out all three lists.

Checkpoint: Compile and run the program. Enter 7 for the maximum value. The value 2 should be removed from the candidates list and added to the primes. Now all values that are divisible by the prime should be removed from the candidates list and added to the composites list. Next, this procedure will be encapsulated in the method getComposites().

Step 9. Refer to the pre-lab exercises and complete the `getComposites()` method. To determine if one integer value is divisible by another, you can use the modulus operator (`%` in Java).

Step 10. Between the code from steps 7 and 8, call `getComposites()`.

Checkpoint: Compile and run the program. Enter 15 for the maximum value. Compare the results with the pre-lab exercises. Reconcile any differences.

Just as in the counting game, a loop will be used to do the rounds.

Step 11. Wrap the code from steps 5 through 8 in a `while` loop that continues as long as the candidates list is not empty.

Final checkpoint: Compile and run the program. Enter 15 for the maximum value. Compare the results with the pre-lab exercises. Reconcile any differences.

Run the program with 100 as the maximum value. Carefully verify your results.

Post-Lab Follow-Ups

1. Modify the counting game program to compute the winning player for different numbers of players from 2 up to an input value. Figure out which player wins for a rhyme of length three for games with 2 to 20 players and record the results in a table. Can you discover a relation between the size of the player list, the length of the rhyme, and the winning player?

2. After a certain point in our iteration, all the remaining values in the candidates list are prime. Modify the program to just copy values directly when that point has been reached.

3. Write a program that will get a list of words and then remove any duplicates.

4. Write a program that will get two lists of words and will create a third list that consists of any words in common between the two input lists.

5. Write a program that will read in a list of words and will create and display two lists. The first list will be the words in odd positions of the original list. The second list will be all the remaining words.

6. Write a program that will get a list of integer values each of which is greater than one and determines if all the posible pairs of values from the list are relatively prime. Two values are relatively prime if they share no prime factors. For example, the values 10 and 77 are relatively prime, but 10 and 55 are not since the share 5 as a factor. If we look at the list {6, 25, 77} each of the pairs (6, 25), (6, 77), and (25,77) are relatively prime and it passes our test. On the other hand the list {6, 25, 14} will fail the test. While the pairs (6, 25) and (25, 14) are relatively prime, the pair (6,14) is not.