GIT Department of Computer Engineering
CSE 222/505 - Spring 2022
Homework 3 Report

Emre YILMAZ
1901042606

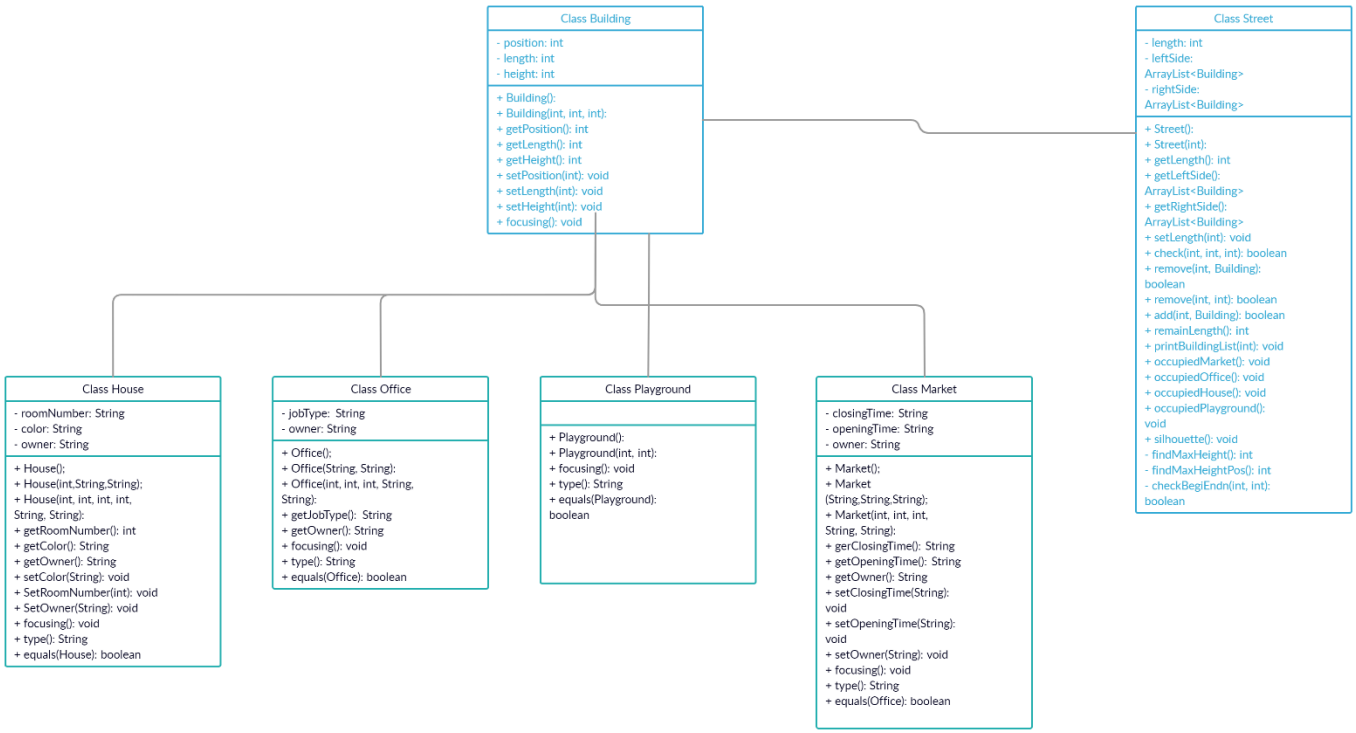
1. SYSTEM REQUIREMENTS

Main requirements -> The system is a street design program. Street must have 2 side (left-right) and the user can set the length at the beginning of the program. At the beginning of the program, there is no building in the street. There are 4 building type. User can add any of these four building types to any side of the street and can set the properties of those buildings. Buildings can have common and distinctive features. In addition to add/remove process, the user can access some information such as the remaining free land, the land occupied by the buildings, the properties of the buildings etc. User can display the skyline silhouette of the street.

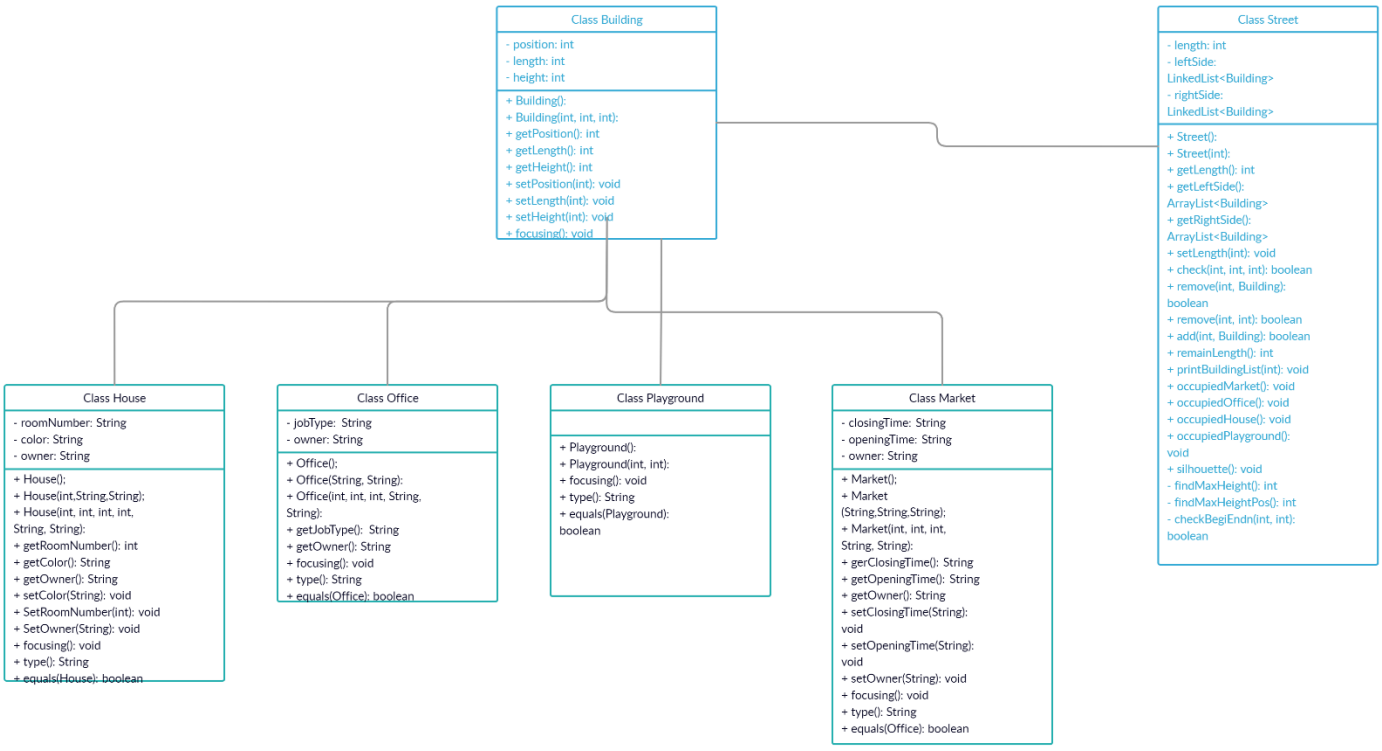
Additional requirements: There must be 4 different systems. The differences of the systems are that the type of data structure that holds the data that is in the building type is different from each other. One of the systems is array, one is linked list, one is ArrayList, and one is LDLinkedList designed by me. LinkedList must be designed using Lazy Delation. It means that buildings removed from the street should continue to be kept on another second list, and this list should be checked first when adding a new building. If the building to be added is included in this second list, no new object should be created. These 4 programs should be tested and running time must be measured to compare data structures.

2. CLASS DIAGRAM

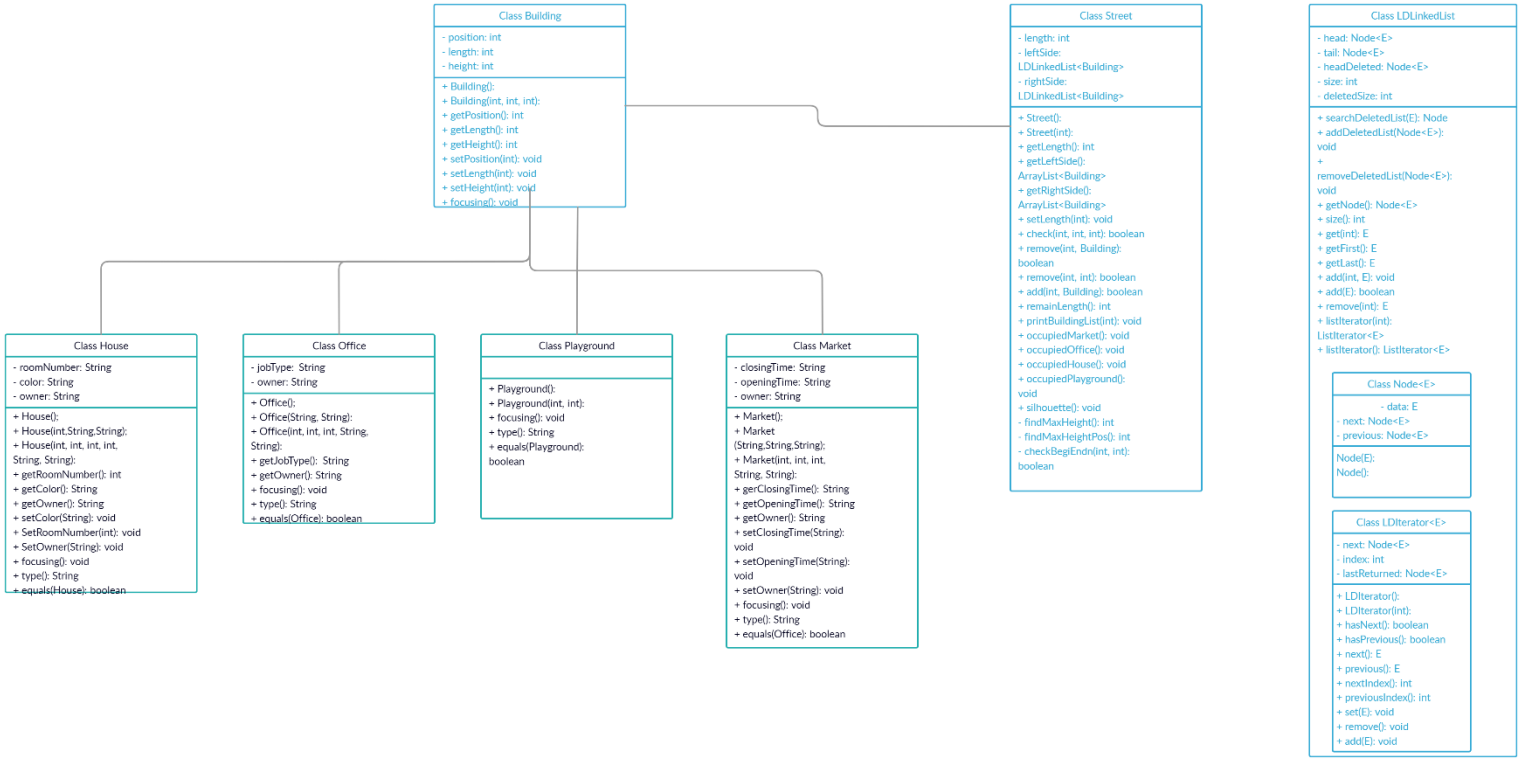
ArrayList Part:



LinkedList Part:



LDLinkedList Part:



3. PROBLEM SOLUTION APPROACH

General Solution (HW1 BASED) -> In this assignment, we were asked to design a street where buildings can be added and removed to it, and the properties of the buildings it owns can be displayed. Since there are 2 sides to the street, I created a class called Street and defined 2 classic Java arrays. I have also created a class called Building to represent generic buildings. One of the arrays in the Street class was set up as the Building type to hold the buildings on the left and the other was set up to hold the buildings on the right.

When adding a new building to one of these arrays, it was first checked for the validness of the input and checked that it coincided with another building. During the addition process, buildings were added to the end of the arrays. I have defined 4 custom classes of the building name type such as House, Office, Market that inherit the Building class. Thus, buildings with appropriate parameters in specific building types such as House and Office could be added to the street. After these operations, viewing operations in PDF have become very easy. However, the silhouette extraction function was challenging for this assignment. In the Street class, during the top-down silhouette extraction process, many private auxiliary functions were used, such as the function that calculates longest height in a specific position, function to find out if a position is at the top of the position where it is located, functions to find out if a position is the starting or ending position of any building etc.

HW2 Based Solution -> It is very easy to switch from using array to using java LinkedList and ArrayList. It is just changed some function calls. The main thing of this assignment is to write our own linked list and our own iterator. So, I had no difficulty using Linked List and ArrayList.

Linked List Part -> First of all, it should be noted that the iterator has been used wherever it is available. For this reason, the run-time duration of most functions has not changed compared to the first version of the assignment, and the add-remove functions have been reduced from o(n) to o(1) since the nodes is add to tail of the LinkedList

ArrayList Part -> Complexity is no different from the first version of the assignment. This system can be considered as using the most correct form of the first version of the homework. (HW1)

LDLinkedList Part -> LDLinked List is designed as double LinkedList. *Node<E>* inner class is used to hold data. The LD*Iterator<E>* inner class has been implemented. In

addition to the classic linked list structure, the second linked list variable that deleted nodes are kept has been added. This deleted list variable is treated as Single Linked List variable to make implementation easier. Again, in this second list, the functions that will search, add and remove have been implemented. Functions other than this are essential functions that must be present in a linked list. When adding a new object, this second list is searched first, and if an element was found, it moved to the main linked list. An iterator was not used to see the complexity difference better when adding/removing a building. (The iterator inner class was implemented and used only for testing in the remove function. It can be checked). Due to the search operations in the second list and the deliberate absence of an iterator, the run-time is expected to be high.

3.1 THEORITICAL TIME COMPLEXITIES

- Add() function is:
  - O(n) in ArrayList
  - O(1) in LinkedList
  - O(n) in LDLinkedList (because of searching the second list)
  - O(n) in HW1
- Remove() function is:
  - O(n) in ArrayList
  - O(1) in LinkedList (because of using Iterator)
  - O(n) in LDLinkedList (because of using Iterator)
  - O(n) in HW1
- Printing the street
  - $O(n^3)$
- Check() function
  - $O(n^2)$ in ArrayList
  - O(n) in LinkedList (because of using iterator)
  - $O(n^3)$ in LDLinkedList (because of NOT using iterator)
  - $O(n^2)$ in HW1
- RemainLength() function
  - $O(n^2)$ in ArrayList
  - O(n) in LinkedList
  - $O(n^2)$ in LDLinkedList (because of NOT using iterator)
  - $O(n^2)$ in HW1
- PrintBuildingList function
  - O(n) in ArrayList
  - O(n) in LinkedList (because of using iterator)
  - $O(n^2)$ in LDLinkedList (because of NOT using iterator)
- O(n) in HW1

- Occupied functions
  - O(n) in ArrayList
  - O(n) in LinkedList (because of using iterator)
  - $O(n^2)$ in LDLinkedList (because of NOT using iterator)
  - O(n) in HW1

General complexities excluding printing street:

Linked List implementation: O(n)

ArrayList implementation: $O(n^2)$

LDLinkedList implementation: $O(n^3)$

Hw1 implementation: $O(n^2)$

General complexities including printing street:

Linked List implementation: $O(n^3)$

ArrayList implementation: $O(n^3)$

LDLinkedList implementation: $O(n^3)$

Hw1 implementation: $O(n^3)$

4. TEST CASES

- Firstly, we have to measure the run-time. To do this, I created data structures with 10,100,1000 elements, respectively. After the insertion process, I performed the viewing operations. after that, I removed all the elements again with remove(). (These operations were repeated in all 3 files and taken to the comment line in the Driver codes.)

After measuring run-time, the operations below are performed for testing in all 3 files.

• 2 House, 2 Office, 2 Market and 1 Playground objects are created.

• The position first House object is set as -3. It is an invalid value and the program warns us.

• First Office (that is named office1) and the second Market (that is named market2) are set to coincide with other buildings. Again, the program warns us about coinciding and does not the add the buildings to arrays

• It is printed that the total remaining length in the street It is printed .

• Buildings on the left side and buildings on the right side are printed with their position information separately.

• Total length on the street that is occupied by buildings is printed for each different Building types.

• Number of playgrounds and its ratio to the street are printed.

• Some focusing methods are tested.

• Then, printed the silhouette of the street.

• Removed an element from the building array.

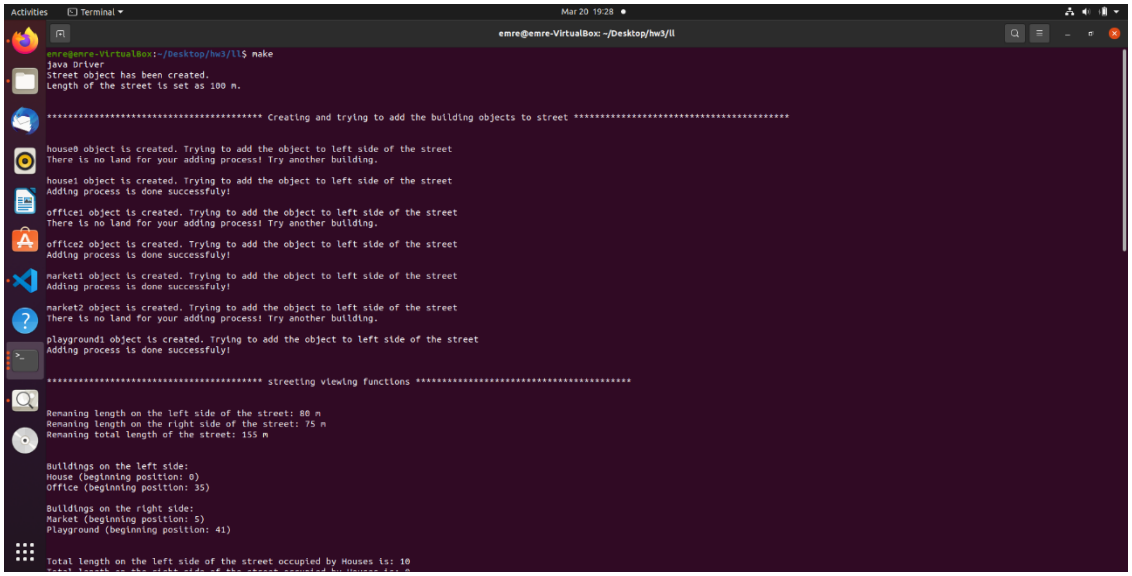
• Then, it is printed the silhouette of the array, again.

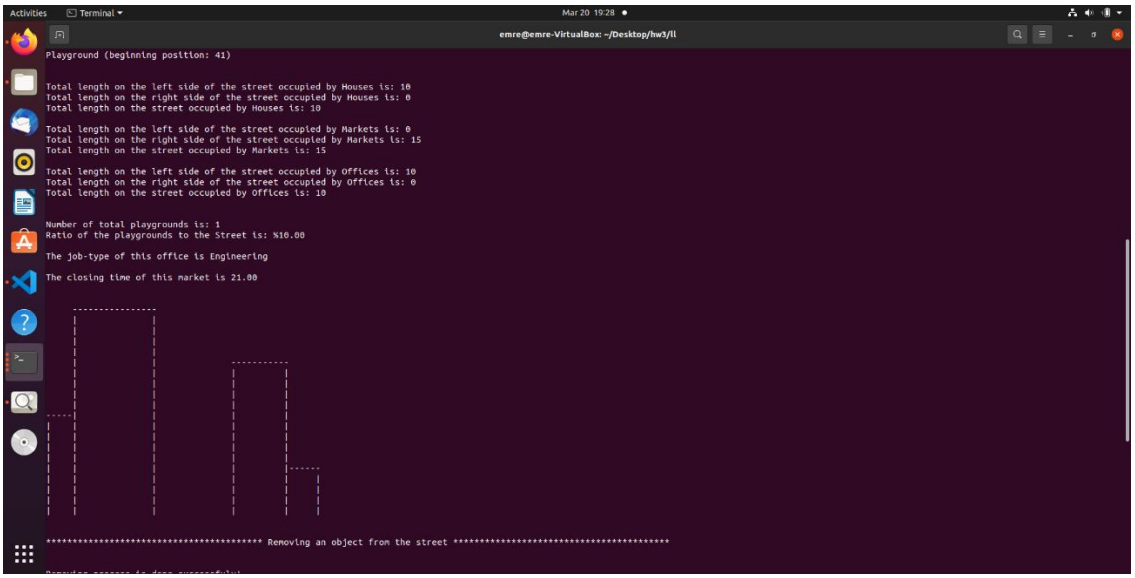
- In addition, the operation of re-adding a node that was previously deleted to the list cannot be tested because the functions are not interactive with the user. But it is observed that the *second deleted list* makes program slower.

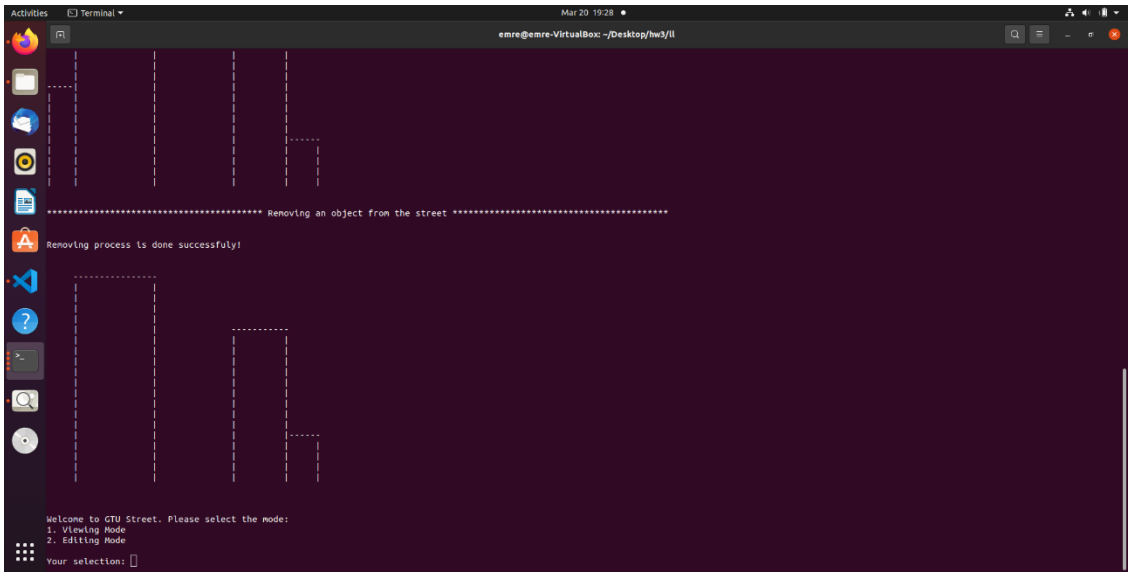
• **Finally**, a menu appears for user. You can try all the function yourselves. You can reach the view and edit menu, you can check their all the operations.

• In the interactive test where the user uses the menu, all inputs have been checked with exception handling.

5. RUNNING AND RESULTS







5.2 **Testing Complexities**

```java
import java.util.InputMismatchException;
import java.util.Scanner;

/**
 * Driver class.
 * @author Emre Yilmaz - 1901042606
 * @version 1.0
 * @since 2022 March
 */
public class Driver
{
    /**
     * Main function
     * @param args -
     */
    Run | Debug
    public static void main(String[] args)
    {
        long start = System.currentTimeMillis(); // Start the clock to measure run-time

        Street street = new Street(10000); // Creating street object

        for(int i=0;i<30000;i=i+3)
        {
            street.add(1,new House(i,3,10,3,"red","Emre"));
        }

        for(int i=3;i<30000;i=i+3)
        {
            street.add(2,new House(i,3,15,3,"red","Emre"));
        }

```

```java
    // streeting viewing functions
    System.out.printf("\n\n************************************* streeting viewing functions *************************************\n\n");
    street.remainLength();
    street.printBuildingList(3);
    street.occupiedHouse();
    street.occupiedMarket();
    street.occupiedOffice();
    street.occupiedPlayground();
    street.getLeftSide().get(1).focusing();
    street.getRightSide().get(0).focusing();
    street.silhouette();

    System.out.printf("\n\n************************************* Removing an object from the street *************************************\n\n");
    if(!street.remove(1, 0))
        System.out.printf("\nThe building is cannot found! Try again\n");
    else
        System.out.printf("\nRemoving process is done successfuly!\n");

    street.silhouette(); // Print the street's silhouette, again


    for(int i=0;i<30000;i=i+3)
    {
        street.remove(1,i);
    }

    for(int i=3;i<30000;i=i+3)
    {
        street.remove(2,i);
    }

    long end = System.currentTimeMillis();
    System.out.println("\n\nThe run-time is: "+(end-start));
```

ArrayList with 10.000 input:

```
The run-time is: 19141
```

ArrayList with 1000 input:

```
The run-time is: 1859
```

ArrayList with 100 input:

```
The run-time is: 257
```

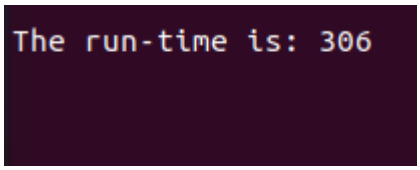
LinkedList with 10.000 input:

```
The run-time is: 26721
```

LinkedList with 1000 input:

```
The run-time is: 3435
```

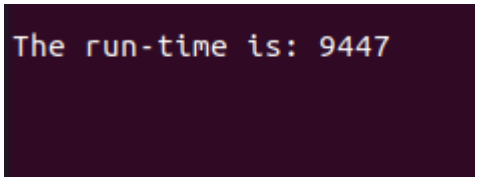
LinkedList with 100 input:



LDLinkedList with 10.000 input:

Cannot be calculate since the time is too long

LDLinkedList with 1000 input:



LDLinkedList with 100 input: