GTU Department of Computer Engineering
CSE 222/505 - Spring 2022

Homework 3 Report
Name : Md Sarwar Hossain
Student Number : 161044121

System Requirements:

This software is for small one street city planning. The buildings of the city are placed around the street. So the first thing the software requires a length for the street. Based on the length we can position the buildings of the city.

The city can have different kinds of building. We categorize the building into 3 groups. They are Houses, Markets and Offices. Besides these we also have playground. As playground is not owned by anyone we are still counting it as kind of building but not completely.

For the general kind of building , all of them length , height, owner , where they are positioned on the street. For the playground we will have just length but no owner or height. Besides these common properties, a particular type of building may have some other properties.

Suppose, a house is mainly known by it's owner , what is it's color ? Or how many rooms it has. When building the system, we will keep this thing in mind.

Like the house, A office is mainly known for what kind of service does it offer. Same way , A market is marked by it's opening and closing time.

All these properties of a building needs to handled accordingly. After we plan the city, We should have a nice view about , how it's gonna look. For that we need to plot the skyline silhouette.

We should also ensure that , when placing a building in the city , It should properly city with the aspect of the city.

All these things we need to keep in mind while building the system.

Problem Solution Approach:

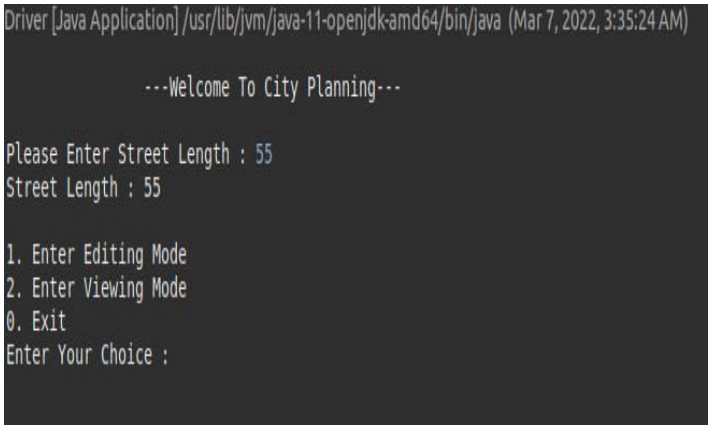
To fulfill the requirements of the system, I started from very root of the problem. That is the buildings. Since all the building have some sorts of similarities, I put them in a base abstract class.
These class will hold some of the basic features of a building , like length , height , position, owner.
Besides typical building , Playground is also under the building but with no owner and height. So I introduced constructor overload to deal with playground. Besides the position co-ordinate, the side of the street, where the building will be placed is also important. So for all the building , Side became a important factor while designing the system. The most challenging part, is the skyline silhouette printing. I decided to divide and conquer algorithm for this place. This works like same as

merger sort. So first we divide the buildings into two smaller sub section until there is no more sub section possible . Then from the smallest point we recursively merge them on basis of changing height. For displaying the silhouette, I used a char array.

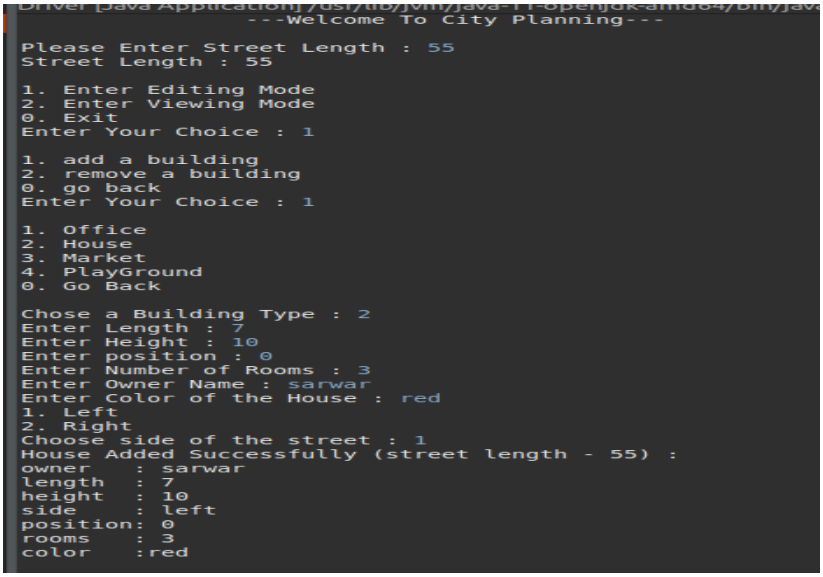
For better understanding , I introduced some enumeration like Building Type , Side. And finally to combine all the classes, I used a Street class like the main character.

Test Cases :

Case 1. Setting up street length
Expected Output: Street will have a length Result : pass



2. add a house
Expected Output : A new House will be created and added Result : pass



3. add a office
Expected Output : A new office will be created and added result :
pass

```
1. Office
2. House
3. Market
4. PlayGround
0. Go Back

Chose a Building Type : 1
Enter Length : 8
Enter Height : 5
Enter position : 5
Enter Owner Name : swr
Enter Job Type : unknown
1. Left
2. Right
Choose side of the street : 1
Office Added Successfully (street length - 55):
owner    : swr
length   : 8
height   : 5
side     : left
position: 5
job type: unknown
```

4. add a market
Expected Output : A new market will be created and added result :

```
1. Office
2. House
3. Market
4. PlayGround
0. Go Back

Chose a Building Type : 3
Enter Length : 10
Enter Height : 20
Enter position : 20
Enter Owner Name : swr
Enter Opening Time : morning 10
Enter Closing Time : evening 10
1. Left
2. Right
Choose side of the street : 1
Market Added Successfully(street length - 55) :
owner        : swr
length       : 10
height       : 20
side         : left
position     : 20
opening time: morning 10
closing time:evening 10
```

5. add a playground
Expected Output : A new playground will be created and added

```
1. Office
2. House
3. Market
4. PlayGround
0. Go Back

Chose a Building Type : 4
Enter Length : 10
Enter position : 30
1. Left
2. Right
Choose side of the street : 1
Playground Added Successfully (street length - 55) :
length   : 10
side     : left
position: 30
```

6. view list of building

Expected Output: All the buildings will be shown with by their focus point result : pass

```
1. Enter Editing Mode
2. Enter Viewing Mode
0. Exit
Enter Your Choice : 2

1. Remaining Length of Lands on the street
2. List of Buildings on the Left Side of the  Street
3. List of Buildings on the Right Side of the  Street
4. Number and ration of Length of Playgrounds on the street
5. Total Length Occupied By Market / House / Office
6. SkyLine Silhouette of the Street
0. Go Back
2
                --Building on Left Side--
id : 1  house owner : sarwar
id : 2  office job type : unknown
id : 3  closing time : evening 10
id : 4  building type : playground
```

7. remove a building

Expected Output : If a valid id is provided , the particular building should be removed result : pass

```
1. add a building
2. remove a building
0. go back
Enter Your Choice : 2
1. Left
2. Right
Choose side of the street : 1
                        ---Building on the left side of the street---
id : 1  house owner : sarwar
id : 2  office job type : unknown
id : 3  closing time : evening 10
id : 4  building type : playground
Enter Building Id to Remove(Press 0 to GoBack ) :
3
Building removed Successfully :
owner        : swr
length       : 10
height       : 20
side         : left
position     : 20
opening time: morning 10
closing time:evening 10
```

8. add a building with wrong value

Expected Output : It should give us error and building will not be added

result : pass

```
1. add a building
2. remove a building
0. go back
Enter Your Choice : 1

1. Office
2. House
3. Market
4. PlayGround
0. Go Back

Chose a Building Type : 1
Enter Length : 100
invalid length
```

9. remove a building with invalid id

Expected Output : It shouldn't remove any building as the didn't match

Result : pass

```
1. add a building
2. remove a building
0. go back
Enter Your Choice : 2
1. Left
2. Right
Choose side of the street : 1
                        ---Building on the left side of the street---
id : 1  house owner : sarwar
id : 2  office job type : unknown
id : 4  building type : playground
Enter Building Id to Remove(Press 0 to GoBack ) :
22
Failed to Remove Building. Carefully Type Building Id !!!
```

10.total length of street occupied by the markets, houses or offices

Expected Output : It should show all the space taken by market /house and office in details

Result : pass

```
1. Remaining Length of Lands on the street
2. List of Buildings on the Left Side of the  Street
3. List of Buildings on the Right Side of the  Street
4. Number and ration of Length of Playgrounds on the street
5. Total Length Occupied By Market / House / Office
6. SkyLine Silhouette of the Street
0. Go Back
5
On Left Side, Length Occupied By House : 7 Market : 0 Office : 8  Total: 15
On Right Side, Length Occupied By House : 0 Market : 0 Office : 0  Total: 0
```

11.invalid input handling

Expected Output : All forms of error should be caught

Result : pass

```
1. Office
2. House
3. Market
4. PlayGround
0. Go Back

Chose a Building Type : 1
Enter Length : 0
Enter Height : 20
Enter position : -1
invalid position !!!
```

12.adding building with invalid position
Expected Output: Out of bound positions are not allowed
Result : pass

```
1. Office
2. House
3. Market
4. PlayGround
0. Go Back

Chose a Building Type : 1
Enter Length : 0
Enter Height : 20
Enter position : -1
invalid position !!!
```

13.Remaining land
Expected Output : It should display the amount of unoccupied land on both side of the street
Result : pass

```
Choose side of the street : 1
Playground Added Successfully (street length - 55) :
length  : 10
side    : left
position: 0


1. Enter Editing Mode
2. Enter Viewing Mode
0. Exit
Enter Your Choice : 2

1. Remaining Length of Lands on the street
2. List of Buildings on the Left Side of the  Street
3. List of Buildings on the Right Side of the  Street
4. Number and ration of Length of Playgrounds on the street
5. Total Length Occupied By Market / House / Office
6. SkyLine Silhouette of the Street
0. Go Back
Enter your choice : 1
Remaining Length on Left Side : 45
Remaining Length on Right Side : 55
```

14.ratio of playground length
Expected Output: It should show you the number of playground and their ratio in terms of total area
Result : pass

```
1. Remaining Length of Lands on the street
2. List of Buildings on the Left Side of the  Street
3. List of Buildings on the Right Side of the  Street
4. Number and ratio of Length of Playgrounds on the street
5. Total Length Occupied By Market / House / Office
6. SkyLine Silhouette of the Street
0. Go Back
Enter your choice : 4
On Left Side - Number of PlayGround : 1 Ratio of Length Compared to Street : 0.18181818181818182
On Right Side - Number of PlayGround : 0 Ratio of Length Compared to Street : 0.0
```

15.building details
expected Output: It should show you all the information about a particular buildings
result : pass

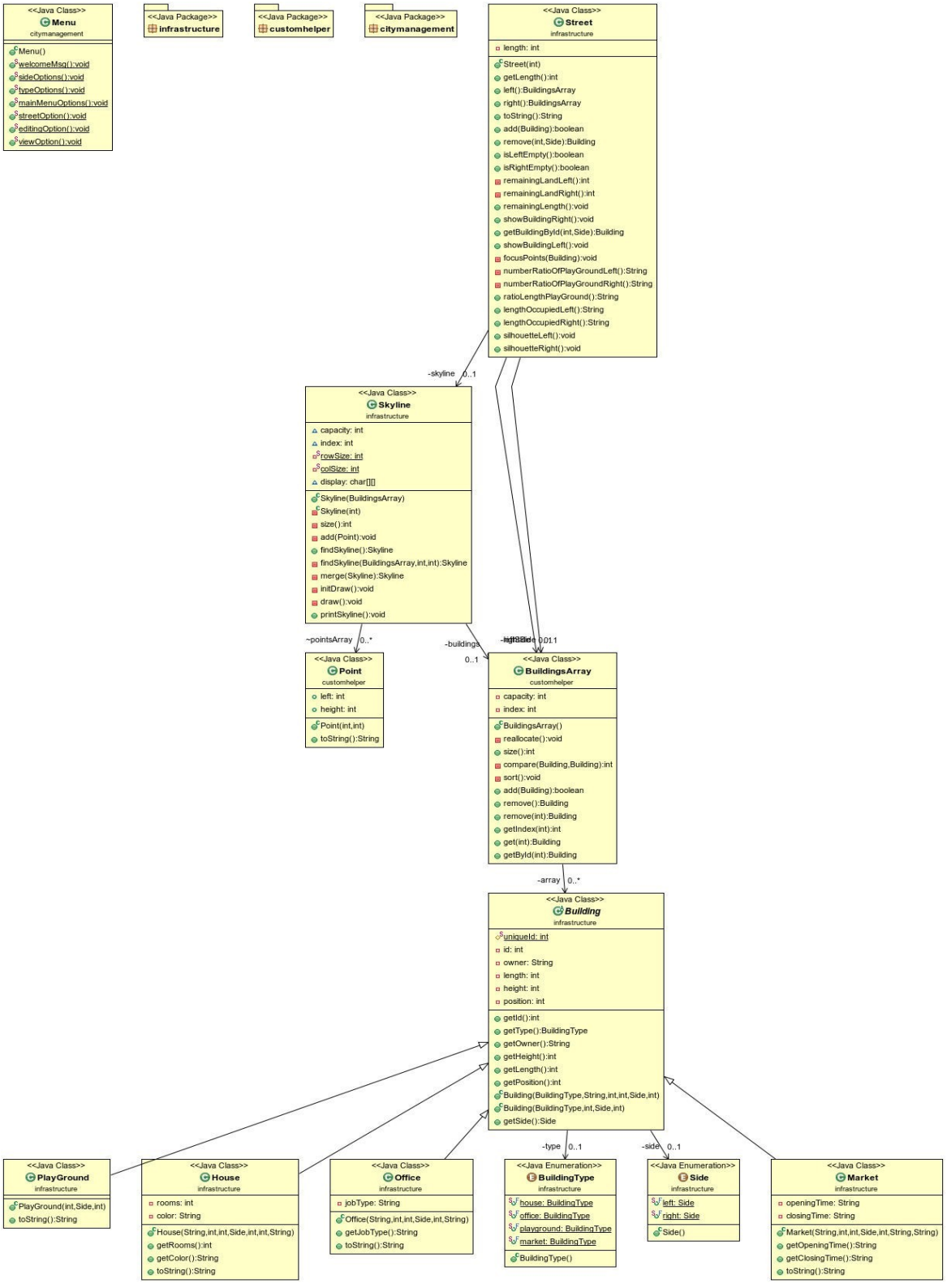
```
1. Remaining Length of Lands on the street
2. List of Buildings on the Left Side of the  Street
3. List of Buildings on the Right Side of the  Street
4. Number and ratio of Length of Playgrounds on the street
5. Total Length Occupied By Market / House / Office
6. SkyLine Silhouette of the Street
0. Go Back
Enter your choice : 2
                --Building on Left Side--
id : 1  building type : playground

Do you want to see detail of a property(y/n) : y
Enter a building id : 1
length  : 10
side    : left
position: 0
```

16.silhouette display
Expected Output:
It should display a skyline silhouette in the console.
Result : pass. This shape was printed on the basis of current building cordinates

```
1. Remaining Length of Lands on the street
2. List of Buildings on the Left Side of the  Street
3. List of Buildings on the Right Side of the  Street
4. Number and ratio of Length of Playgrounds on the street
5. Total Length Occupied By Market / House / Office
6. SkyLine Silhouette of the Street
0. Go Back
Enter your choice : 6
1. Left
2. Right
Choose side of the street : 1
            ---Silhouette Display of Left Side of Street---

                  ***********
                  *         *
                  *         *             ******
                  *         *             *    *
                  *         *             *    *
********  ***********         *         ******    ******
*       *  *                  *         *              *
*       *  *                  *         *              *
*       *  *                  *         *              *
*       *  *                  *         *              *
*       ****                  *         *              *
*                             *         *              *
*                             *         *              *
*                             *         *              *
*                             *         *              *
*                             ***********              *

1. Remaining Length of Lands on the street
```

Previous Homework Analysis:

Class Diagram :



Time Complexity:

```
private void reallocate() {
	this.capacity = this.capacity *2;
	Building[] tempArray = new Building[this.capacity];

	//copying the old array to new array
	for(int i = 0; i < size(); i++) {
		tempArray[i]=array[i];
	}

	//reference back to main array
	array = tempArray;

}

Time Complexity: O(n)
```

```
public boolean add(Building building) {
	if(this.size() == this.capacity) {
		try {
			reallocate();----------------------O(n)
		} catch (Exception e) {
			System.out.println("failed to add more building");
			return false;
		}
	}
	array[index++] = building;
	return true;
}

Time Complexity : O(n). Best case is Ω(1) if no reallocation needed.
```

```
/**
*
* @return removes the last element in the buildings array
*/
public Building remove() {
	Building temp = array[this.size()-1];
	this.index--;
	return temp;
}

Time Complexity : O(1)
```

```
/**
 *
 * @param index is an integer value
 * @return returns Building at the particular index if index if valid, otherwise
 *           throws exception
 */
public Building remove(int pos) {
      if(pos < 0 || pos >= this.size()) {

            throw new IndexOutOfBoundsException();
      }

      Building temp = array[pos];

      for(int i = pos; i < this.size()-1; i++) {
            array[i] = array[i+1];
      }
      this.index--;
      return temp;
}

Time Complexity : O(n)

/**
 *
 * @param id is a integer value refers to id of a Building
 * @return return a integer index corresponds to the building id
 */
public int getIndex(int id) {
      for(int i = 0; i < this.size(); i++) {
            if(array[i].getId() == id) {
                  return i;
            }
      }
      return -1;
}

Time Complexity : O(n)

/**
 *
 * @param index takes a integer value
 * @return returns a Building at a given index if index is valid, else throws
 *           exception
 */
public Building get(int index) {
      if(index < 0 || index >= this.size()) {
            throw new IndexOutOfBoundsException();
      }
      return array[index];
}
Time Complexity : O(n)


/**
 *
 * @param id is a integer value, represents the id of a Building
 * @return returns a Building if a Building is found by a given id
 */
public Building getById(int id) {
      for(int i = 0; i < this.size(); i++) {
            if(array[i].getId() == id) {
            return array[i];
```

```
            }
      }
      return null;
}

Time Complexity : O(n)



/**
 *
 * @param building is of type Building
 * @return returns true if building is added else false
 */
public boolean add(Building building) {
      if(building.getLength() <= (this.length - building.getPosition())) {
            if(building.getSide() == Side.left) {
                  return leftSide.add(building); ---------O(n)
            }else {
                  return rightSide.add(building);---------O(n)
            }
      }
      return false;
}

Time Complexity : O(n)


/**
 *
 * @param buildingId is of type integer
 * @param side is of type Side
 * @return returns the building that has been removed
 */
public Building remove(int buildingId,Side side) {
      if(side == Side.left) {
            int index = left().getIndex(buildingId);
            try {
                  return left().remove(index); ---------O(n)
            } catch (Exception e) {
                  return null;
            }
      }else {
            int index = right().getIndex(buildingId);
            try {
                  return right().remove(index);---------O(n)
            } catch (Exception e) {
                  return null;
            }
      }
}

Time Complexity: O(n)


/**
 *
 * @return returns a integer value
 */
private int remainingLandLeft() {
      if(left().size()!=0) {
            Building last = left().get(left().size()-1);
            return this.length - last.getLength()+last.getPosition();
      }
      return this.length;
```

```
}

Time Complexity : O(1)

/**
 *
 * @return returns a integer value
 */
private int remainingLandRight() {
      if(right().size()!=0) {
            Building last = right().get(right().size()-1);
            return this.length - last.getLength()+last.getPosition();
      }
      return this.length;
}

Time Complexity : O(1)


/**
 * prints the remaining unoccupied space
 */
public void remainingLength() {
      System.out.println("Remaining Length on Left Side :
      "+remainingLandLeft());
      System.out.println("Remaining Length on Right Side : "+
      remainingLandRight());
}

Time Complexity : O(1)


/**
 *
 * @param bd is of type Building
 */
private void focusPoints(Building bd) {
      if(bd instanceof House) {
            System.out.println("id : "+bd.getId()+"\thouse owner : "+
            bd.getOwner());
      }else if (bd instanceof Office) {
            Office office  = (Office)bd;
            System.out.println("id : "+bd.getId()+"\toffice job type :
            "+office.getJobType());
      }else if (bd instanceof Market) {
            Market market = (Market)bd;
            System.out.println("id : "+bd.getId()+"\tclosing time :
            "+market.getClosingTime());
      }else  if (bd instanceof PlayGround){
            System.out.println("id : "+bd.getId()+"\tbuilding type :
            playground");
      }
}
Time Complexity : O(1)

      /**
       * prints the building on the right side of the street
       */
      public void showBuildingRight() {
            if(right().size() == 0) {
                  System.out.println("No Building Found ");
                  return;
            }
            for(int i =0; i < right().size(); i++) {
                  Building bd = right().get(i);
```

```
            focusPoints(bd);-----------O(1)
      }
}
```

Time Complexity: O(n)

```
/**
 * prints the buildings on the left side of the street
 */
public void showBuildingLeft() {
      if(left().size()==0) {
            System.out.println("No Building Found");
            return;
      }
      for(int i =0; i < left().size(); i++) {
            Building bd = left().get(i);
            focusPoints(bd);
      }
}
```

Time Complexity: O(n)

```
/**
 *
 * @return returns a String with information about playground of right
   side of the street
 */
private String numberRatioOfPlayGroundRight() {
      int count =0;
      double totalLength =0;
      for(int i =0 ; i < right().size(); i++) {
            if(right().get(i) instanceof PlayGround) {
                  count++;
                  totalLength+=right().get(i).getLength();
            }
      }
      double ratio = totalLength/(double)this.length;
      return "On Right Side - Number of PlayGround : "+count+" Ratio of
      Length Compared to Street : "+ ratio;
}
```

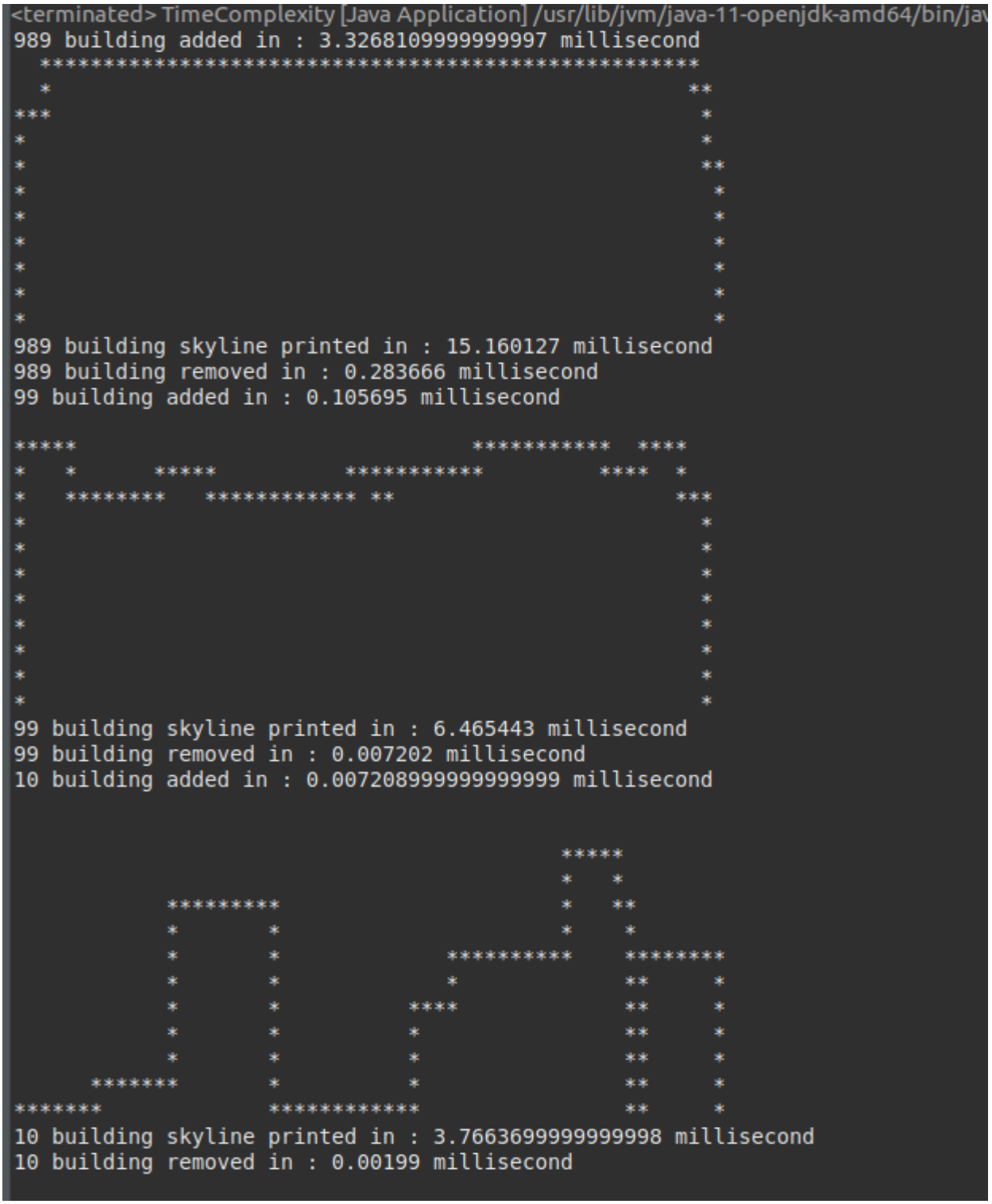
Time Complexity: O(n)

```
/**
 * @return returns a Strin g value
 */
public String lengthOccupiedLeft() {
      int houseLength =0;
      int officeLength =0;
      int marketLength =0;
      for(int i =0; i < left().size(); i++) {
```

```
                  if(left().get(i) instanceof House) {
                        houseLength+= left().get(i).getLength();
                  }else if (left().get(i) instanceof Office) {
                        officeLength+= left().get(i).getLength();
                  }else if (left().get(i) instanceof Market) {
                        marketLength+= left().get(i).getLength();
                  }
            }
            return "On Left Side, Length Occupied By House : "+houseLength+"
            Market : "+marketLength+" Office : "+officeLength+"  Total: "+
            (houseLength+marketLength+officeLength);
      }

      Time Complexity : O(n)



/**
* @param st this function will take a Point as parameter
*/
private void add(Point st){
      if (index > 0 && pointsArray[index - 1].height == st.height)
            return;
      if (index > 0 && pointsArray[index - 1].left == st.left) {
            pointsArray[index - 1].height = Math.max(pointsArray[index -
            1].height, st.left);
            return;
      }
      pointsArray[index] = st;
      index++;
}

Time complexity : O(1)



/* @param other is of type Skyline
 * @return returns a instance of Skyline class
*/
private  Skyline merge(Skyline other){
      Skyline res = new Skyline(this.index+ other.size());
      int h1 = 0, h2 = 0;
      int i = 0, j = 0;
      while (i < this.index && j < other.size()) {
            if (this.pointsArray[i].left < other.pointsArray[j].left) {
                  int x1 = this.pointsArray[i].left;
                  h1 = this.pointsArray[i].height;
                  int maxh = Math.max(h1, h2);
                  res.add(new Point(x1, maxh));
                  if(rowSize < maxh) {
                        rowSize = maxh;
                  }
                  i++;
            }
            else {
                int x2 = other.pointsArray[j].left;
                h2 = other.pointsArray[j].height;
                int maxh = Math.max(h1, h2);
                res.add(new Point(x2, maxh));
                if(rowSize < maxh) {
                  rowSize = maxh;
                }
                j++;
            }
```

```
        }
        while (i < this.index) {
            if(rowSize < pointsArray[i].height) {
                  rowSize = pointsArray[i].height;
            }
            res.add(pointsArray[i]);
            i++;
        }
        while (j < other.size()) {
            if(rowSize < other.pointsArray[j].height) {
                  rowSize = other.pointsArray[j].height;
            }
            res.add(other.pointsArray[j]);
            j++;
        }
        return res;
    }
Time Complexity : Θ (nlogn)


    /**
     * printing the skyline
     */
    public void printSkyline() {
      draw();
      for(int i =0; i < rowSize; i++) {
            for(int j =0; j < colSize; j++) {
                  System.out.print(display[i][j]);
            }
            System.out.println();
      }
    }

      Time Complexity: O(n^2)



      /**
       * displays the silhouette for the left side of the street
       */
      public void silhouetteLeft() {
            if(left().size()!=0) {
                  skyline = new Skyline(left());
                  skyline= skyline.findSkyline();  --------Θ(nlogn)
                  skyline.printSkyline(); -----------------O(n^2)
            }else
                  System.out.println("No Building Found");
      }

      Time Complexity: O(n^2+nlogn) / O(n^2)
```

Time Complexity Compared To Theoretical Time:



Adding building has a time complexity of O (n) or linear time.
For adding 10, 100 and 1000 building the time grew linearly.

For linear growth,

10 building should take 10 unit,
100 building should take 10*10 unit,
1000 building should take 10*10*10 unit of time

From the actual time running time we can see,

For 10 building   : .0072 millisecond
For 100 building : .1056 millisecond
For 1000 building : 3.326 millisecond

which are pretty close to the expected result.

For the remove last method that I used to here in time complexity analysis has a time complexity of O(1). For constant time complexity, we are expecting to have same time for all size.

From the actual time running time we can see,

For 10 building   : .0019 millisecond
For 100 building : .0072 millisecond
For 1000 building : .28   millisecond

the difference is negligible.

For Printing the skyline silhouette, The time complexity is O(n^2). So time growth for displaying the skyline silhouette is quadratic.

Now let’s see actual running time,

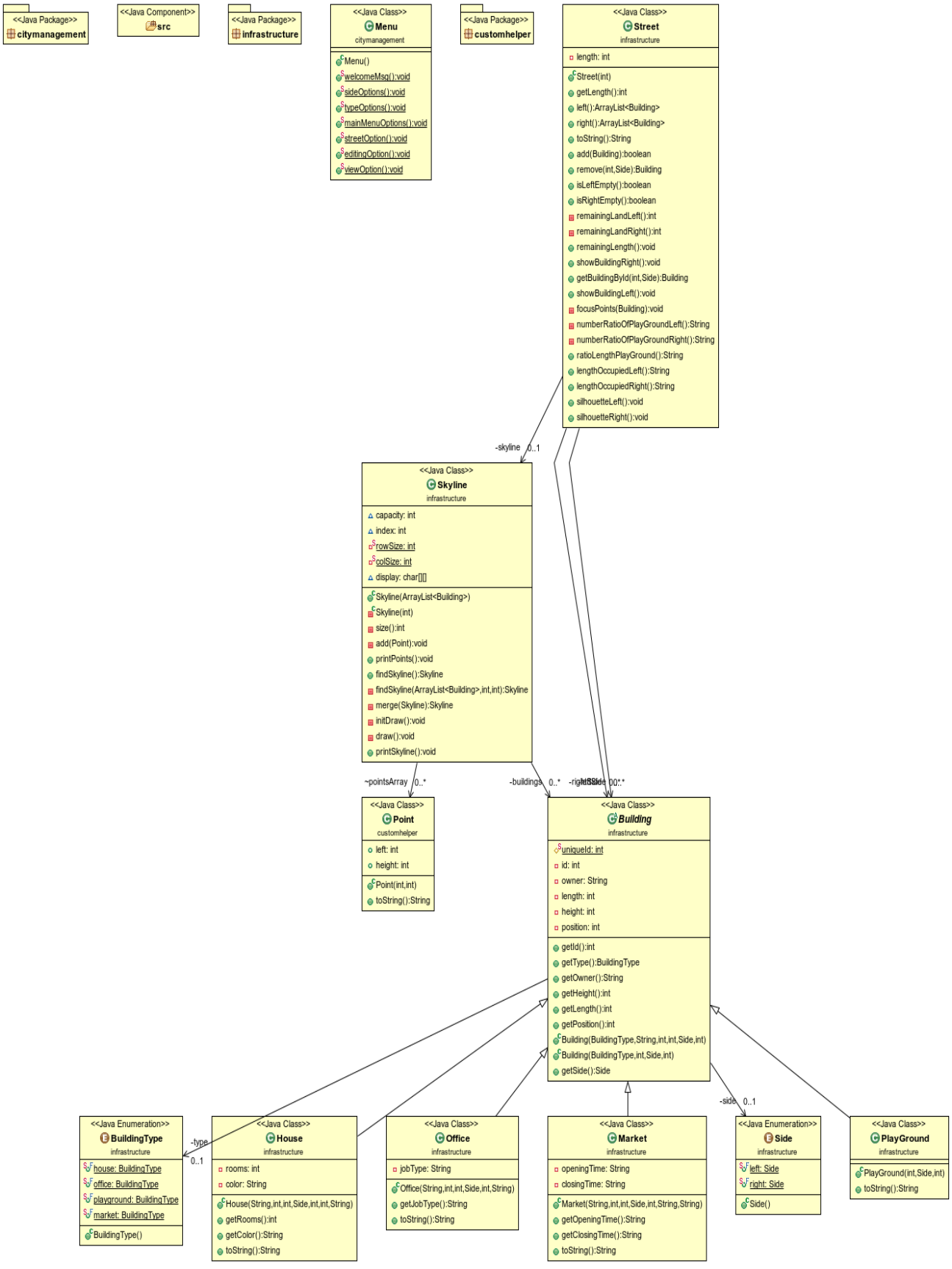
For 10 building   : 3.76 millisecond
For 100 building : 6.46 millisecond
For 1000 building : 15.16   millisecond

from the result it’s obvious that it has quadratic time growth as the result almost double after increasing the size by 10 times.

Version one analysis (ArrayList) :

Problem solutions approach:
The system requirements stayed the same. For this part I just replaced the regular array with a arraylist. The functionality of the program stayed same. Since My previous homework had the same time complexity as the arraylist, the time complexity also didn’t change.

Class Diagram:



Time Complexity Analysis :

Adding building has a time complexity of O (n) or linear time in worst case and O(1) for the best case.
Since I am using arraylist and there is a possibility of reallocaion.
Otherwise it's constant time.
For adding 10, 100 and 1000 building the time grew linearly.

For linear growth,

10 building should take 10 unit,
100 building should take 10*10 unit,
1000 building should take 10*10*10 unit of time

From the actual time running time we can see,

For 10 building   : .0011 millisecond
For 100 building : .07 millisecond
For 1000 building : 6 millisecond

For the 10 and 100 building we can see, the time is almost constant. As the size is small so the reallocation is done fast. But for 1000 building the time is much more. Which is almost linear.
So we can clearly visualize the time complexity from best and worst case scenerio.

For the removing element from arbitrary location the time complexity is O(n). However, here I used clear method of arraylist which is a constant time operation.

From the actual time running time we can see,

For 10 building   : .006 millisecond
For 100 building : .005 millisecond
For 1000 building : .03 millisecond

the difference is negligible.So it's constant time.

For Printing the skyline silhouette, The time complexity is O(n^2). So time growth for displaying the skyline silhouette is quadratic.

Now let's see actual running time,

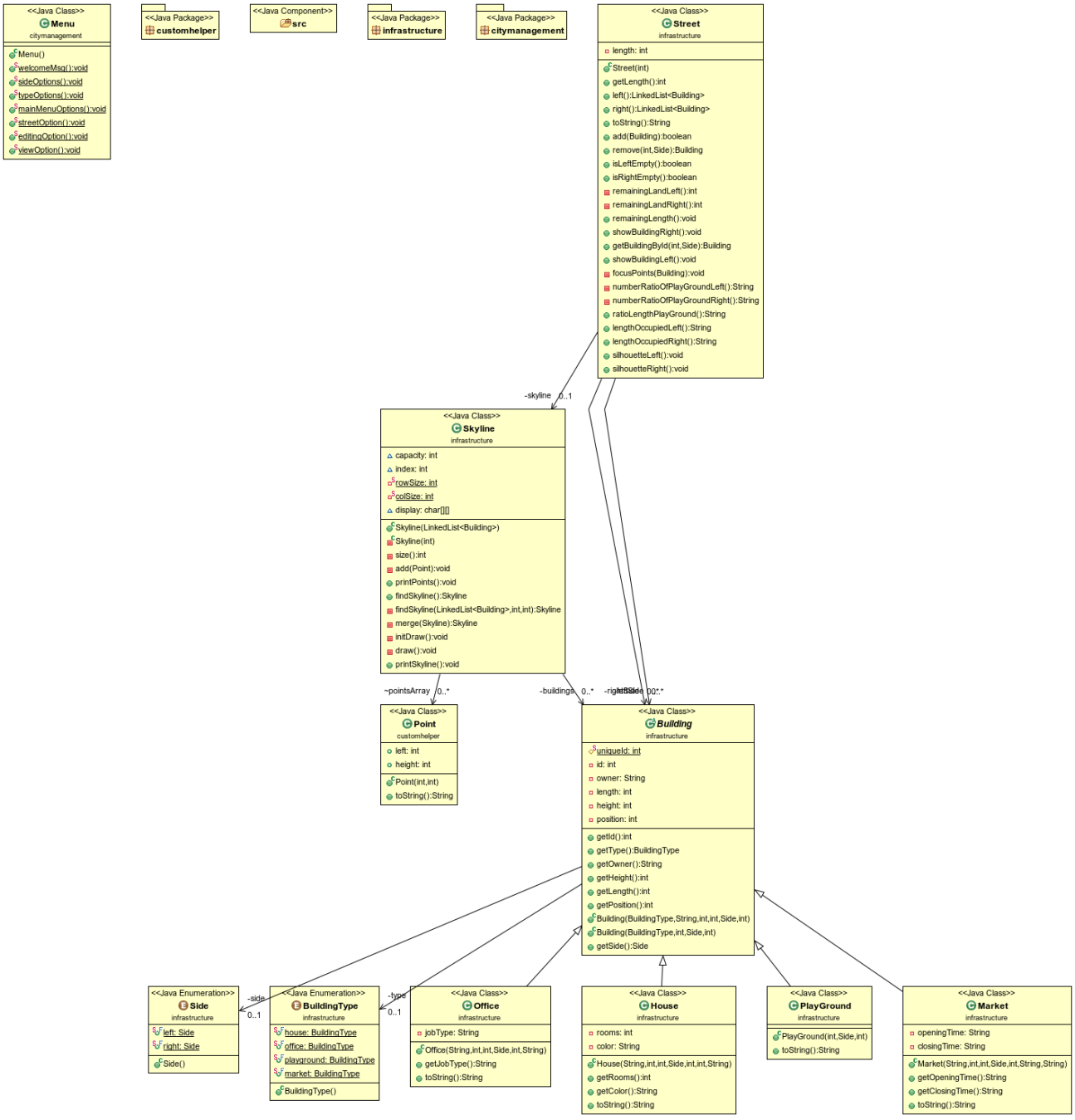
For 10 building   : 5.57 millisecond
For 100 building : 7.21 millisecond
For 1000 building : 15.70   millisecond

from the result it's obvious that it has quadratic time growth as the result almost double after increasing the size by 10 times.

Moreover, the time complexity is almost same as the time complexity of previous homework for displaying silhouette section. So there wasn't any visible change in time as both the implementation is pretty same.

Vesion Two Analysis (LinkedList) :

Class Diagram:



Problem Solution Approach :

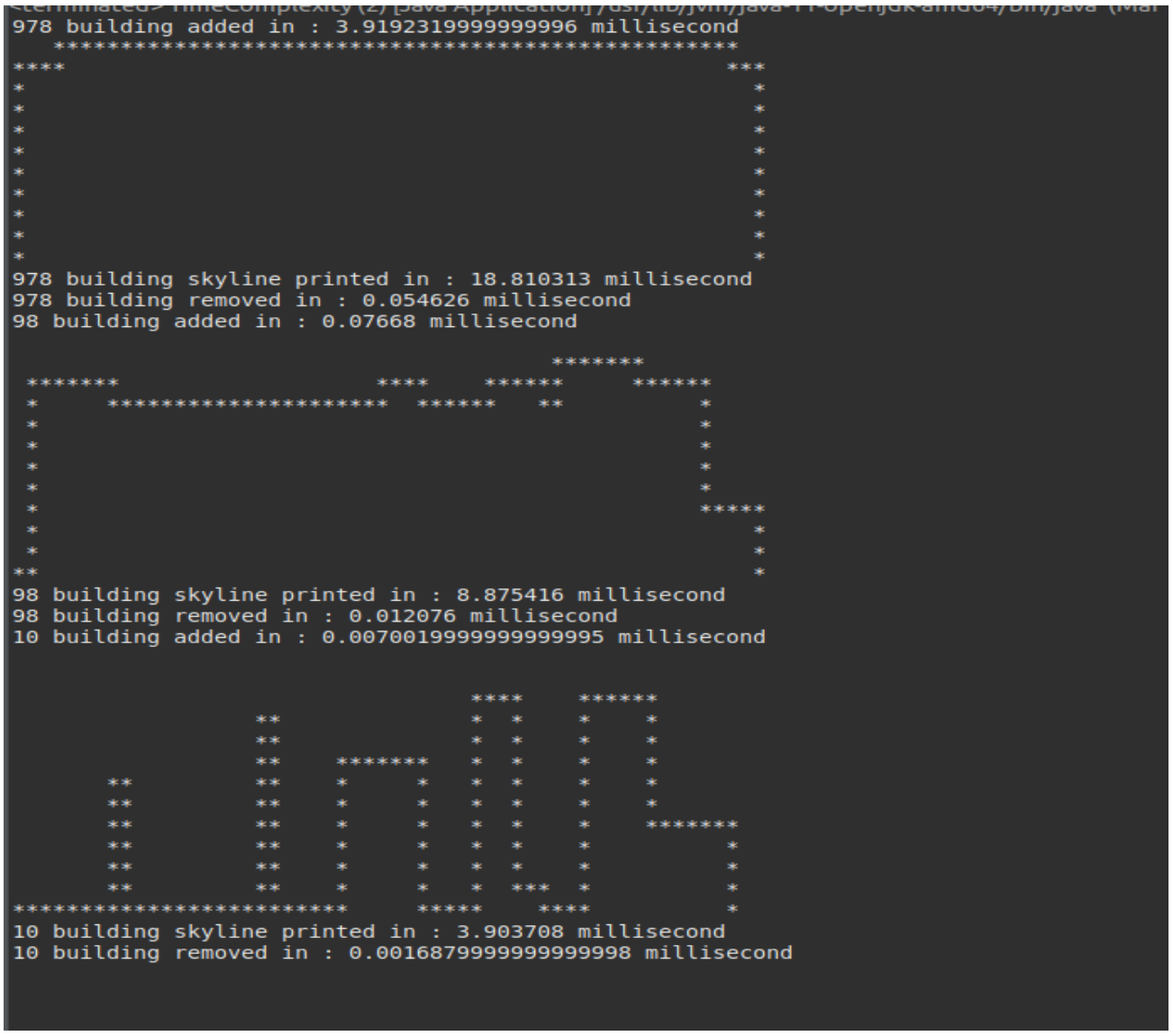
I replaced ArrayList with Java Collection LinkedList for holding the buildings.

Time Complexity Analysis :

```
/**
*
* @param building is of type {@link Building}
* @return returns true if building is added else false
*/
public boolean add(Building building) {
	if(building.getLength() <= (this.length - building.getPosition())) {
		if(building.getSide() == Side.left) {
			return leftSide.add(building);
		}else {
			return rightSide.add(building);
		}
	}
	return false;
}
```

Since we are using java linked list and adding at the end or from has constant time complexity. So the add method will have constant time complexity.
Time Comlexity: O(1)

Running Time vs Theoretical Time:



Adding building has a constant Time complexity O(1).

For Constant time, For any size it should take almost the same time.
Since I am using a for loop to add the building which will take liner time.
If we ignore that for

From the actual time running time we can see,

For 10 building   : .007 millisecond
For 100 building : .12 millisecond
For 1000 building : 3.9 millisecond

Since I am using a for loop to add the building which will take liner time.
If we ignore the loop it’s obvious that it has constant time complexity as the difference are very small.

For the removing element from arbitrary location the time complexity is O(n).
However, here I used clear method of LinkedList which is a constant time operation.

From the actual time running time we can see,

For 10 building   : .0016 millisecond
For 100 building : .005 millisecond
For 1000 building : .05 millisecond

the difference is negligible.So it’s constant time.

For Printing the skyline silhouette, The time complexity is O(n^2). So time growth for displaying the skyline silhouette is quadratic.

Now let’s see actual running time,

For 10 building   : 3.9 millisecond
For 100 building : 8.8 millisecond
For 1000 building : 18.8   millisecond

from the result it’s obvious that it has quadratic time growth as the result almost double after increasing the size by 10 times.

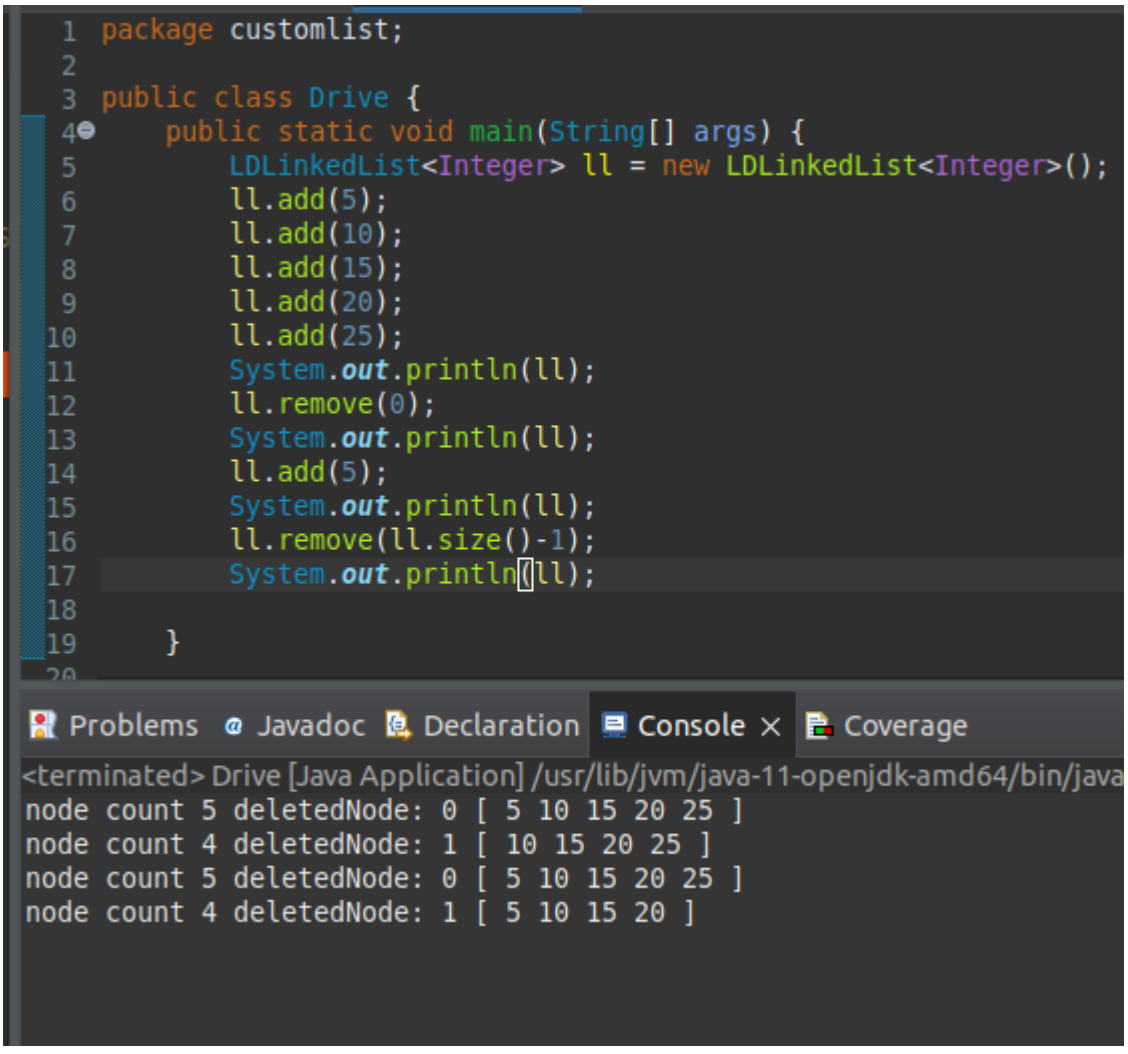
Moreover, the time complexity is almost same as the time complexity of ArrayList homework for displaying silhouette section. So there wasn’t any visible change in time as both the implementation is pretty same.

Version Three analysis (LDLinkedList) :

Problem Solution Approach :

For implementing the lazy linkedlist I used the mechanism of marking to generate to separate the original list. If a node is requested to get removed I marked with a boolean value. If next time I want to add the same value then instead of creating a node I just unmarked that value. This saves us some time. When the deleted and created node becomes equal I removed the marked node permanently. This way I achieved the lazy linked list. I compared value to understand if a node is already there. To get the full advantage of it the Class using LDLinkedList should override equals method.

Test Case:



After 5 gets removed,when I added 5 it just changed the marking.

Time Complexity Analysis :

```
private boolean activateLazyAdd(E data) {
            Node temp = head;
            while(temp != null) {
                  if(temp.data.equals(data) && temp.markDeleted) {
                        temp.markDeleted = false;
                        nodeCount++;
                        deletedNodeCount--;
                        return true;
                  }
                  temp = temp.next;
            }
            return false;
      }
```

Time Complexity : O(n)

```
      @Override
      public boolean add(E data) {

            if(activateLazyAdd(data)) {----------O(n)
                  return true;
            }
            try {
                   Node newNode = new Node(data);
                    if(head == null) {
                        head = tail = newNode;
                        head.prev = null;
                        tail.next = null;
                    }
                    else {
                        tail.next = newNode;
                        newNode.prev = tail;
                        tail = newNode;
                        tail.next = null;
                    }
                nodeCount++;
                  return true;
            } catch (Exception e) {
                  return false;
            }
      }
```

Time Complexity : Best Case O(1) , worst case O(n)

```
@Override
public void clear() {
      nodeCount = 0;
      deletedNodeCount = 0;
}
```

Time Complexity : O(1)

```
private void deleteNode(Node del)
    {
        if (head == null || del == null) {
            return;
        }
        if (head == del) {
            head = del.next;
        }
        if (del.next != null) {
            del.next.prev = del.prev;
        }
        if (del.prev != null) {
            del.prev.next = del.next;
        }
        return;
    }
```

Time Complexity: O(1)

```
      @Override
      public E remove(int index) {
            if((nodeCount-1) == deletedNodeCount) {
                  deleteMarked();
            }
            if(index < 0 || index>=size()) {
                  throw new IndexOutOfBoundsException();
            }
            int count = 0;
            Node temp = head;
            while(temp!=null) {
                  if(!temp.markDeleted) {
                        if(count == index) {
                              temp.markDeleted = true;
                              deletedNodeCount++;
                              nodeCount--;
                              return temp.data;
                        }
                        count++;
                  }
                  temp = temp.next;
            }
            return null;
      }
```

Time Complexity : O(n)

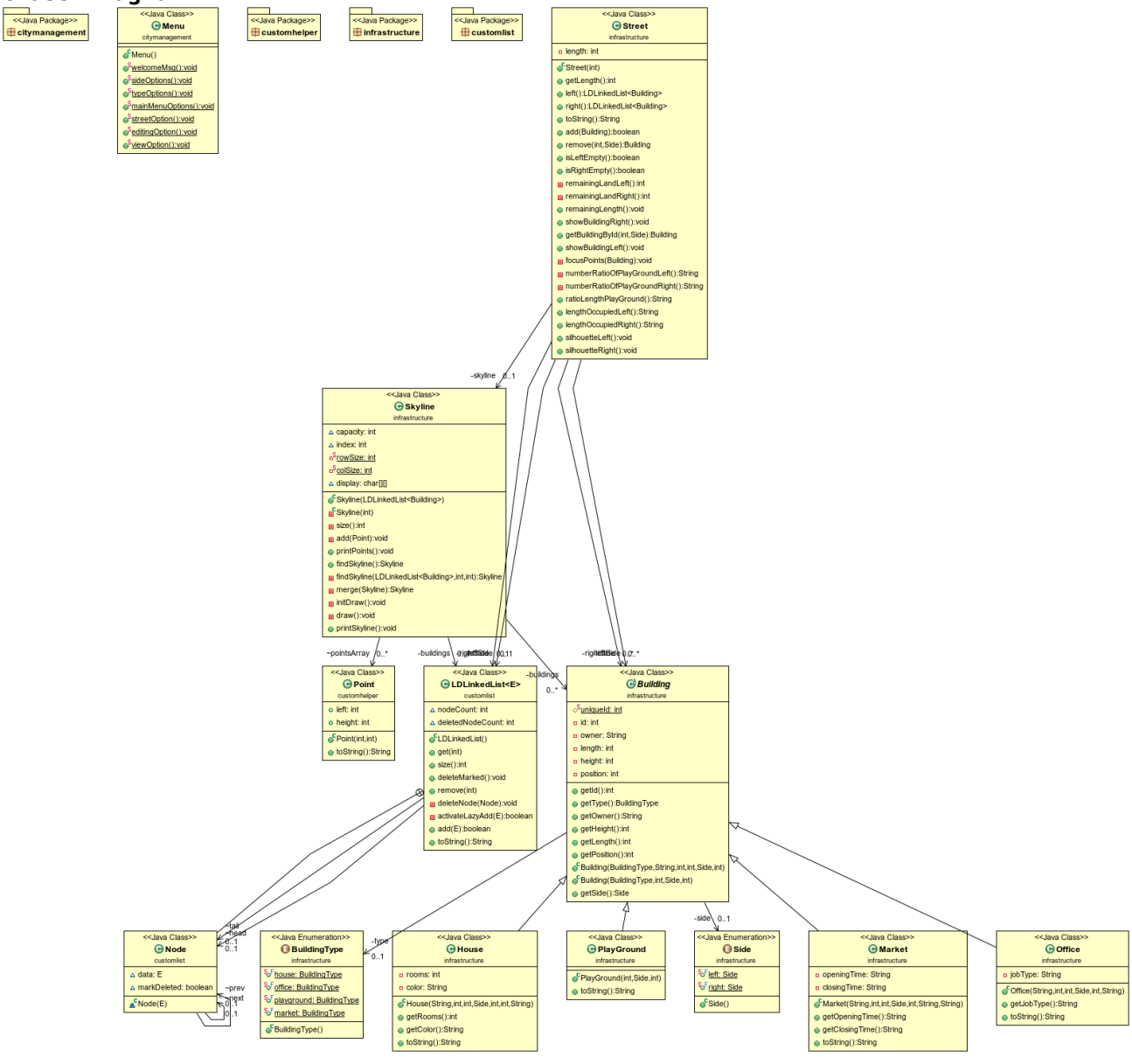
```
public void deleteMarked() {
      Node temp = head;
      while(temp!=null) {

            if(temp.markDeleted) {
                  deleteNode(temp);
            }
            temp = temp.next;
      }
      deletedNodeCount = 0;
}
```

Time Complexity: O(n)

```
	@Override
	public E get(int index) {
		int count = 0;
		Node temp = head;
		while(temp!=null) {
			if(!temp.markDeleted) {
				if(count == index) {
					return temp.data;
				}
				count++;
			}
			temp = temp.next;
		}
		return null;
	}
```

Time Complexity : O(n)

Class Diagram:



Running Time vs Theoretical Time:

```
<terminated> TimeComplexity (3) [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/j
980 building added in : 9.30307 millisecond
*****************************************************
*                                                  ***
*                                                    *
*
*
*
*
*
*
*
*
*
980 building skyline printed in : 23.991999 millisecond
980 building removed in : 0.007954 millisecond
98 building added in : 0.055827999999999996 millisecond

       **       ****    *********        ******
       *******  *  **** *       *   ***  *    *******
  ******     ****  ** ***       ***** ****          *
  *   **                                           *
  *   **                                           *
***                                                *
*                                                  *
*                                                  *
*                                                  ***
*                                                    *
*                                                    *
98 building skyline printed in : 8.700168 millisecond
98 building removed in : 0.001584 millisecond
9 building added in : 0.007494 millisecond


      **********
      *        *               **********
      *        *               *        *
      *        *    *********** *        *
      *        *    *         * *        **
      *        ******         * *         *
      *        **             * *         *
      *        **             * *         *
      *        **             * *         ******
      *        **             * *              *
*******        **             ***
9 building skyline printed in : 3.250607 millisecond
9 building removed in : 0.001137 millisecond
```

Adding building has best case constant Time O(1) and worst case is O(n) since we need to go through the marked building to add from deleted node.
For adding 10, 100 and 1000 building the time grew linearly.

For linear growth,

10 building should take 10 unit,
100 building should take 10*10 unit,
1000 building should take 10*10*10 unit of time

From the actual time running time we can see,

For 10 building   : .0011 millisecond
For 100 building : .007 millisecond
For 1000 building : .007 millisecond

For the remove last method that I used to here in time complexity analysis has a time complexity of O(1). For constant time complexity, we are expecting to have same time for all size.

From the actual time running time we can see,

For 10 building   : .011 millisecond
For 100 building : .013 millisecond
For 1000 building : .   millisecond

the difference is negligible.

For Printing the skyline silhouette, The time complexity is O(n^2). So time growth for displaying the skyline silhouette is quadratic.

Now let’s see actual running time,

For 10 building   : 3.25 millisecond
For 100 building : 8.7 millisecond
For 1000 building : 23.9   millisecond

from the result it’s obvious that it has quadratic time growth as the result almost double after increasing the size by 10 times.