GTU Department of Computer Engineering

CSE 222 / 505 – Spring 2022

Homework 3

Ahmet USLUOGLU

1801042602

1 – System Requirement

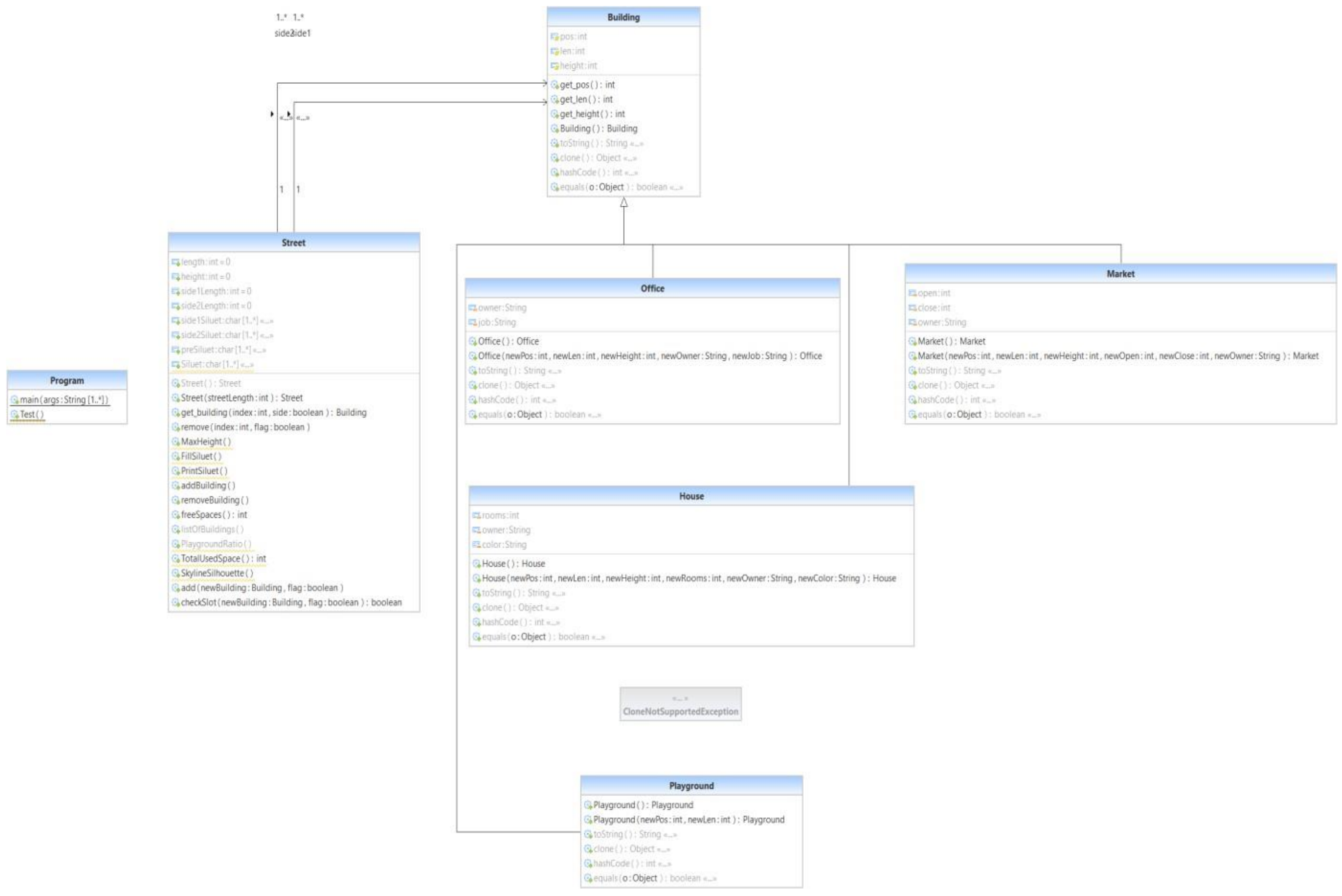
Operating System must have JDK (Java Development Kit) 11 and JRE (Java Runtime Environment) 11 or higher.

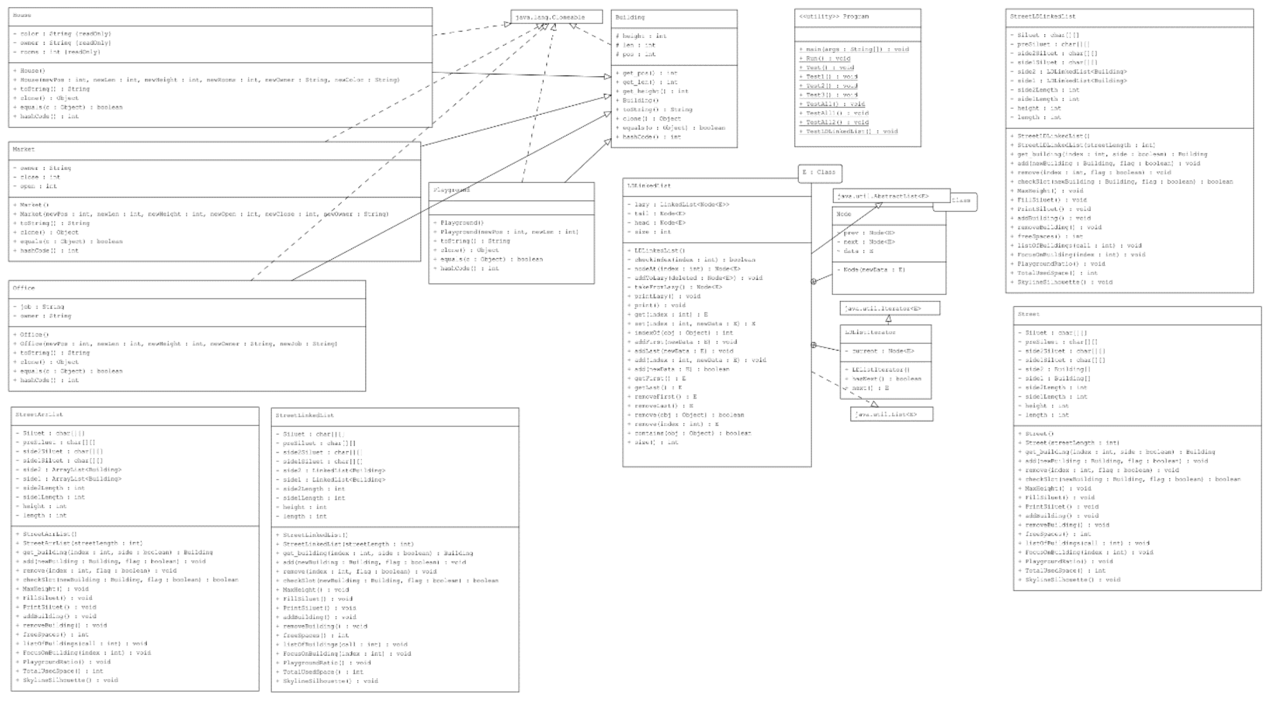
There should be enough space for storing data’s.

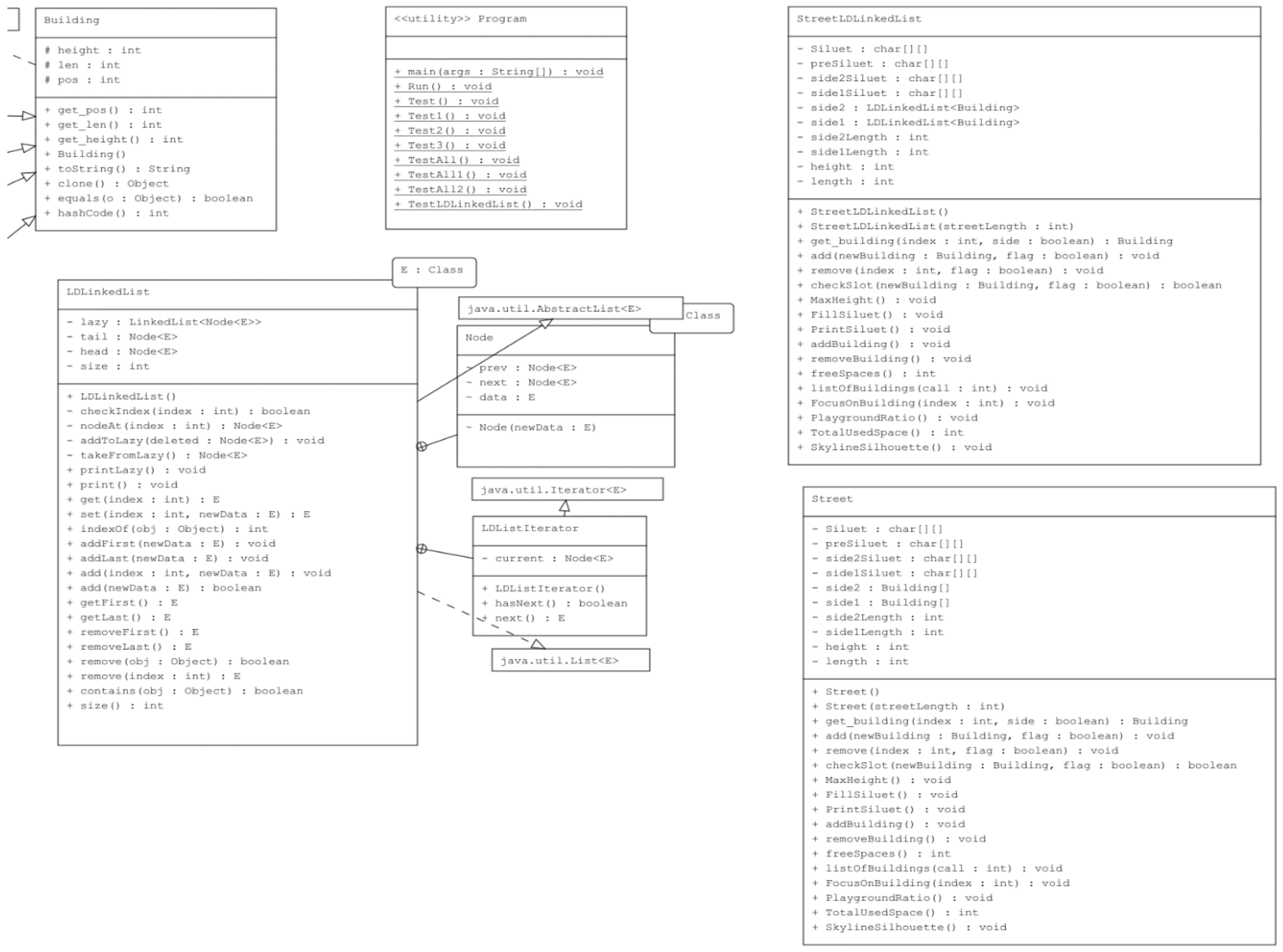
User must create a Street with a fixed length at the start of the program. User must enter correct input values while adding a new building.

2 – Class Diagrams

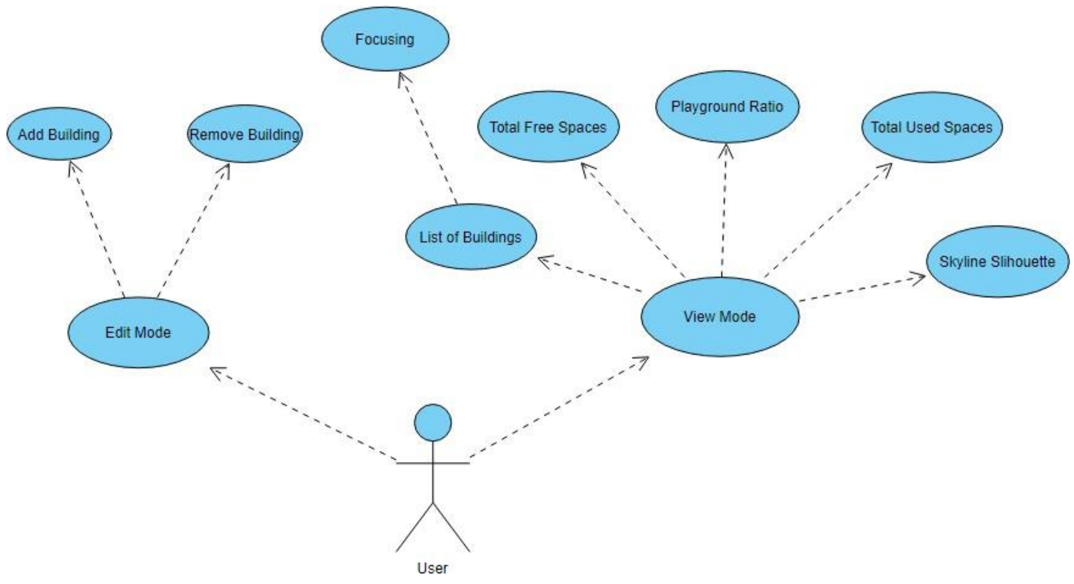
*Higher Resolution Version of the Class Diagram is in the files.


1..* 1..*
side2side1
«...» «...»
1 1
Building
pos:int
len:int
height:int
get_pos() : int
get_len() : int
get_height() : int
Building() : Building
toString() : String «...»
clone() : Object «...»
hashCode() : int «...»
equals(o : Object) : boolean «...»
Street
length:int = 0
height:int = 0
side1Length:int = 0
side2Length:int = 0
side1Siluet:char [1..*] «...»
side2Siluet:char [1..*] «...»
preSiluet:char [1..*] «...»
Siluet:char [1..*] «...»
Street() : Street
Street(streetLength : int) : Street
get_building(index : int, side : boolean) : Building
remove(index : int, flag : boolean)
MaxHeight()
FillSiluet()
PrintSiluet()
addBuilding()
removeBuilding()
freeSpaces() : int
listOfBuildings()
PlaygroundRatio()
TotalUsedSpace() : int
SkylineSilhouette()
add(newBuilding : Building, flag : boolean)
checkSlot(newBuilding : Building, flag : boolean) : boolean
Program
main(args : String [1..*])
Test()
Office
owner:String
job:String
Office() : Office
Office(newPos : int, newLen : int, newHeight : int, newOwner : String, newJob : String) : Office
toString() : String «...»
clone() : Object «...»
hashCode() : int «...»
equals(o : Object) : boolean «...»
Market
open:int
close:int
owner:String
Market() : Market
Market(newPos : int, newLen : int, newHeight : int, newOpen : int, newClose : int, newOwner : String) : Market
toString() : String «...»
clone() : Object «...»
hashCode() : int «...»
equals(o : Object) : boolean «...»
House
rooms:int
owner:String
color:String
House() : House
House(newPos : int, newLen : int, newHeight : int, newRooms : int, newOwner : String, newColor : String) : House
toString() : String «...»
clone() : Object «...»
hashCode() : int «...»
equals(o : Object) : boolean «...»
«...»
CloneNotSupportedException
Playground
Playground() : Playground
Playground(newPos : int, newLen : int) : Playground
toString() : String «...»
clone() : Object «...»
hashCode() : int «...»
equals(o : Object) : boolean «...»


House
- color : String {readOnly}
- owner : String {readOnly}
- rooms : int {readOnly}
+ House()
+ House(newPos : int, newLen : int, newHeight : int, newRooms : int, newOwner : String, newColor : String)
+ toString() : String
+ clone() : Object
+ equals(o : Object) : boolean
+ hashCode() : int
java.lang.Cloneable
Market
- owner : String
- close : int
- open : int
+ Market()
+ Market(newPos : int, newLen : int, newHeight : int, newOpen : int, newClose : int, newOwner : String)
+ toString() : String
+ clone() : Object
+ equals(o : Object) : boolean
+ hashCode() : int
Playground
+ Playground()
+ Playground(newPos : int, newLen : int)
+ toString() : String
+ clone() : Object
+ equals(o : Object) : boolean
+ hashCode() : int
Office
- job : String
- owner : String
+ Office()
+ Office(newPos : int, newLen : int, newHeight : int, newOwner : String, newJob : String)
+ toString() : String
+ clone() : Object
+ equals(o : Object) : boolean
+ hashCode() : int
StreetArrList
- Siluet : char[][]
- preSiluet : char[][]
- side2Siluet : char[][]
- side1Siluet : char[][]
- side2 : ArrayList<Building>
- side1 : ArrayList<Building>
- side2Length : int
- side1Length : int
- height : int
- length : int
+ StreetArrList()
+ StreetArrList(streetLength : int)
+ get_building(index : int, side : boolean) : Building
+ add(newBuilding : Building, flag : boolean) : void
+ remove(index : int, flag : boolean) : void
+ checkSlot(newBuilding : Building, flag : boolean) : boolean
+ MaxHeight() : void
+ FillSiluet() : void
+ PrintSiluet() : void
+ addBuilding() : void
+ removeBuilding() : void
+ freeSpaces() : int
+ listOfBuildings(call : int) : void
+ FocusOnBuilding(index : int) : void
+ PlaygroundRatio() : void
+ TotalUsedSpace() : int
+ SkylineSilhouette() : void
StreetLinkedList
- Siluet : char[][]
- preSiluet : char[][]
- side2Siluet : char[][]
- side1Siluet : char[][]
- side2 : LinkedList<Building>
- side1 : LinkedList<Building>
- side2Length : int
- side1Length : int
- height : int
- length : int
+ StreetLinkedList()
+ StreetLinkedList(streetLength : int)
+ get_building(index : int, side : boolean) : Building
+ add(newBuilding : Building, flag : boolean) : void
+ remove(index : int, flag : boolean) : void
+ checkSlot(newBuilding : Building, flag : boolean) : boolean
+ MaxHeight() : void
+ FillSiluet() : void
+ PrintSiluet() : void
+ addBuilding() : void
+ removeBuilding() : void
+ freeSpaces() : int
+ listOfBuildings(call : int) : void
+ FocusOnBuilding(index : int) : void
+ PlaygroundRatio() : void
+ TotalUsedSpace() : int
+ SkylineSilhouette() : void
Building
<<utility>> Program
StreetLDLinkedList
LDLinkedList
E : Class
java.util.AbstractList<E>
Class
Node
java.util.Iterator<E>
LDListIterator
java.util.List<E>
Street


Building
# height : int
# len : int
# pos : int
+ get_pos() : int
+ get_len() : int
+ get_height() : int
+ Building()
+ toString() : String
+ clone() : Object
+ equals(o : Object) : boolean
+ hashCode() : int
<<utility>> Program
+ main(args : String[]) : void
+ Run() : void
+ Test() : void
+ Test1() : void
+ Test2() : void
+ Test3() : void
+ TestAll() : void
+ TestAll1() : void
+ TestAll2() : void
+ TestLDLinkedList() : void
StreetLDLinkedList
- Siluet : char[][]
- preSiluet : char[][]
- side2Siluet : char[][]
- side1Siluet : char[][]
- side2 : LDLinkedList<Building>
- side1 : LDLinkedList<Building>
- side2Length : int
- side1Length : int
- height : int
- length : int
+ StreetLDLinkedList()
+ StreetLDLinkedList(streetLength : int)
+ get_building(index : int, side : boolean) : Building
+ add(newBuilding : Building, flag : boolean) : void
+ remove(index : int, flag : boolean) : void
+ checkSlot(newBuilding : Building, flag : boolean) : boolean
+ MaxHeight() : void
+ FillSiluet() : void
+ PrintSiluet() : void
+ addBuilding() : void
+ removeBuilding() : void
+ freeSpaces() : int
+ listOfBuildings(call : int) : void
+ FocusOnBuilding(index : int) : void
+ PlaygroundRatio() : void
+ TotalUsedSpace() : int
+ SkylineSilhouette() : void
E : Class
LDLinkedList
- lazy : LinkedList<Node<E>>
- tail : Node<E>
- head : Node<E>
- size : int
+ LDLinkedList()
- checkIndex(index : int) : boolean
- nodeAt(index : int) : Node<E>
- addToLazy(deleted : Node<E>) : void
- takeFromLazy() : Node<E>
+ printLazy() : void
+ print() : void
+ get(index : int) : E
+ set(index : int, newData : E) : E
+ indexOf(obj : Object) : int
+ addFirst(newData : E) : void
+ addLast(newData : E) : void
+ add(index : int, newData : E) : void
+ add(newData : E) : boolean
+ getFirst() : E
+ getLast() : E
+ removeFirst() : E
+ removeLast() : E
+ remove(obj : Object) : boolean
+ remove(index : int) : E
+ contains(obj : Object) : boolean
+ size() : int
java.util.AbstractList<E>
Class
Node
~ prev : Node<E>
~ next : Node<E>
~ data : E
~ Node(newData : E)
java.util.Iterator<E>
LDListIterator
- current : Node<E>
+ LDListIterator()
+ hasNext() : boolean
+ next() : E
java.util.List<E>
Street
- Siluet : char[][]
- preSiluet : char[][]
- side2Siluet : char[][]
- side1Siluet : char[][]
- side2 : Building[]
- side1 : Building[]
- side2Length : int
- side1Length : int
- height : int
- length : int
+ Street()
+ Street(streetLength : int)
+ get_building(index : int, side : boolean) : Building
+ add(newBuilding : Building, flag : boolean) : void
+ remove(index : int, flag : boolean) : void
+ checkSlot(newBuilding : Building, flag : boolean) : boolean
+ MaxHeight() : void
+ FillSiluet() : void
+ PrintSiluet() : void
+ addBuilding() : void
+ removeBuilding() : void
+ freeSpaces() : int
+ listOfBuildings(call : int) : void
+ FocusOnBuilding(index : int) : void
+ PlaygroundRatio() : void
+ TotalUsedSpace() : int
+ SkylineSilhouette() : void

2.1 – Use Case Diagrams



3. Problem - Solution Approach

Problem:

1 -) Designing a small Street planning software with the feature of Silhouette of Skyline. Street consists of 2 sides. Each side has free spaces to construct a building. Street's Length is fixed, and space is limited. Construct buildings that do not conflicts with otherbuildings at the same side.

2-) Implementing above problem's solution with 4 types of data structures which are arrays, array lists, linked lists and custom linked lists (LDLinkedList). Compare all data structure's running time, complexity and theoretical time.

Solution:

1-) The buildings that can be constructed (House, Office, Market, Playground) on the Street inherited from a Building class which has length, position and height properties.

I have then created a Street class to oversee all related functions that will be needed to construct buildings on the street.

I have stored the buildings in 2 separate arrays respective to each side of the Street.

Each side of the Street has 1 Building array as container to store buildings and 1 char array to store silhouette of the street.

After creating 2 different char arrays with silhouettes i have combined them to 1as preSlihouette array by only adding the parts where both of the arrays are empty.

The Empty points in the preSlihouette array is not finished. I have reversed the array and only printed the walls of the buildings at the outer most side. Thus Skyline Silhouette is completed.

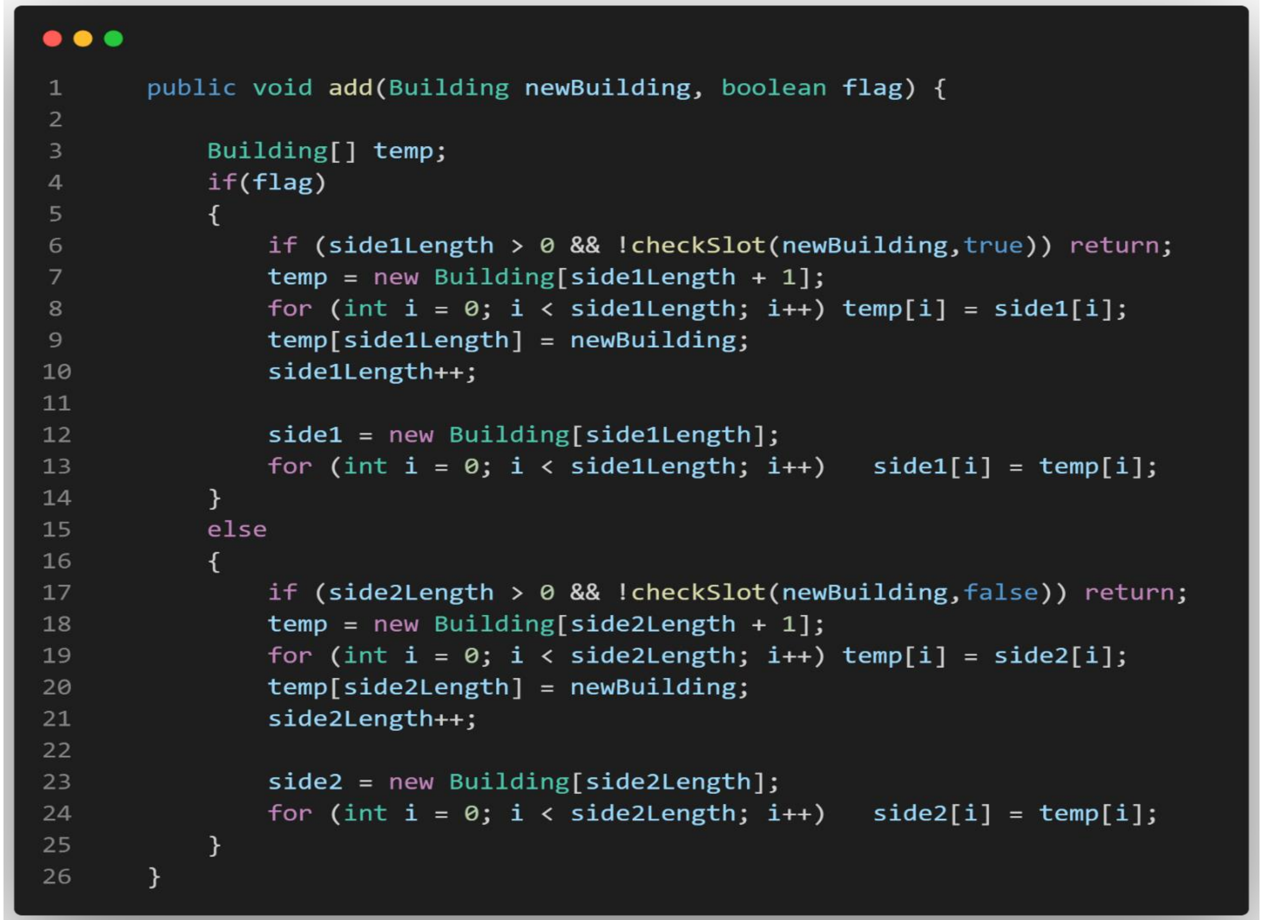
Other functions are simple calculation functions that sums up total lengths, empty spaces, used spaces etc.

Exceptions are handled in case of invalid values for inputs are given.

2-) Using above solution I have created 3 other street classes that uses array list, linked list and LDlinked list as container types for buildings. After creating total of 4 different street class for each of the data structure I have tested their running times with different size of buildings, calculated their complexity and compared their experimental times.

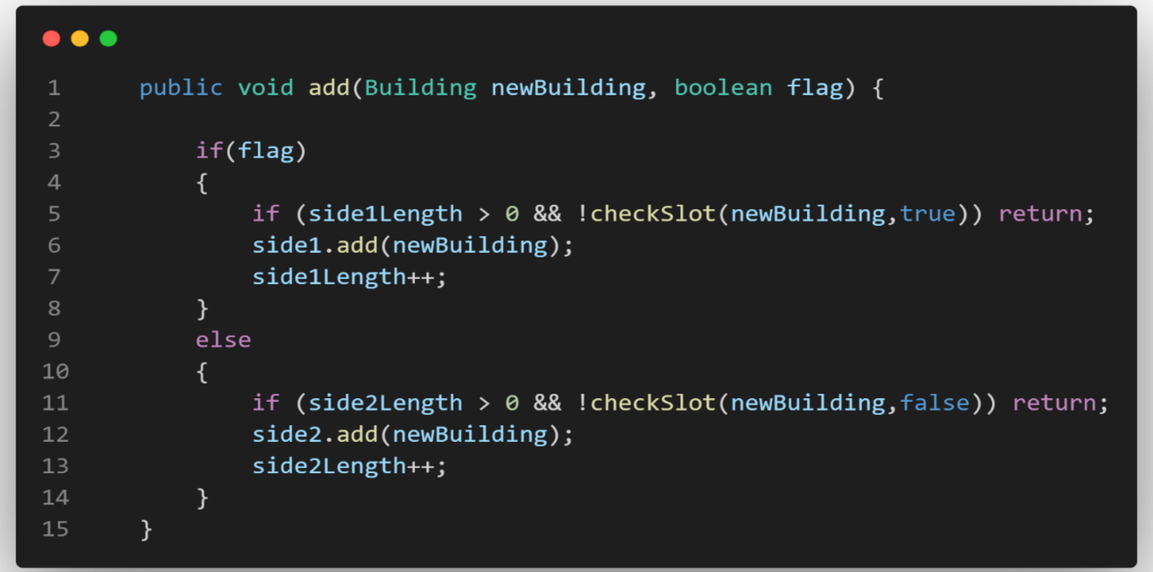
Example of Add Building Function with Different Data Structures

- Using Arrays



```java
    public void add(Building newBuilding, boolean flag) {

        Building[] temp;
        if(flag)
        {
            if (side1Length > 0 && !checkSlot(newBuilding,true)) return;
            temp = new Building[side1Length + 1];
            for (int i = 0; i < side1Length; i++) temp[i] = side1[i];
            temp[side1Length] = newBuilding;
            side1Length++;

            side1 = new Building[side1Length];
            for (int i = 0; i < side1Length; i++)   side1[i] = temp[i];
        }
        else
        {
            if (side2Length > 0 && !checkSlot(newBuilding,false)) return;
            temp = new Building[side2Length + 1];
            for (int i = 0; i < side2Length; i++) temp[i] = side2[i];
            temp[side2Length] = newBuilding;
            side2Length++;

            side2 = new Building[side2Length];
            for (int i = 0; i < side2Length; i++)   side2[i] = temp[i];
        }
    }
```

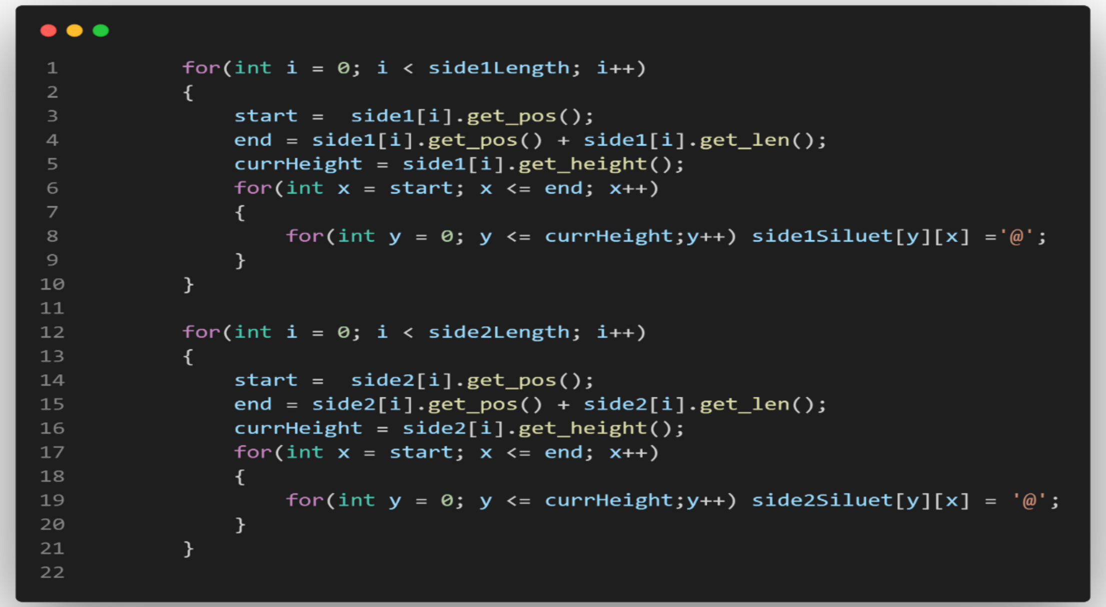
- Using Array list, Linked list, LDLinked list



```java
    public void add(Building newBuilding, boolean flag) {

        if(flag)
        {
            if (side1Length > 0 && !checkSlot(newBuilding,true)) return;
            side1.add(newBuilding);
            side1Length++;
        }
        else
        {
            if (side2Length > 0 && !checkSlot(newBuilding,false)) return;
            side2.add(newBuilding);
            side2Length++;
        }
    }
```

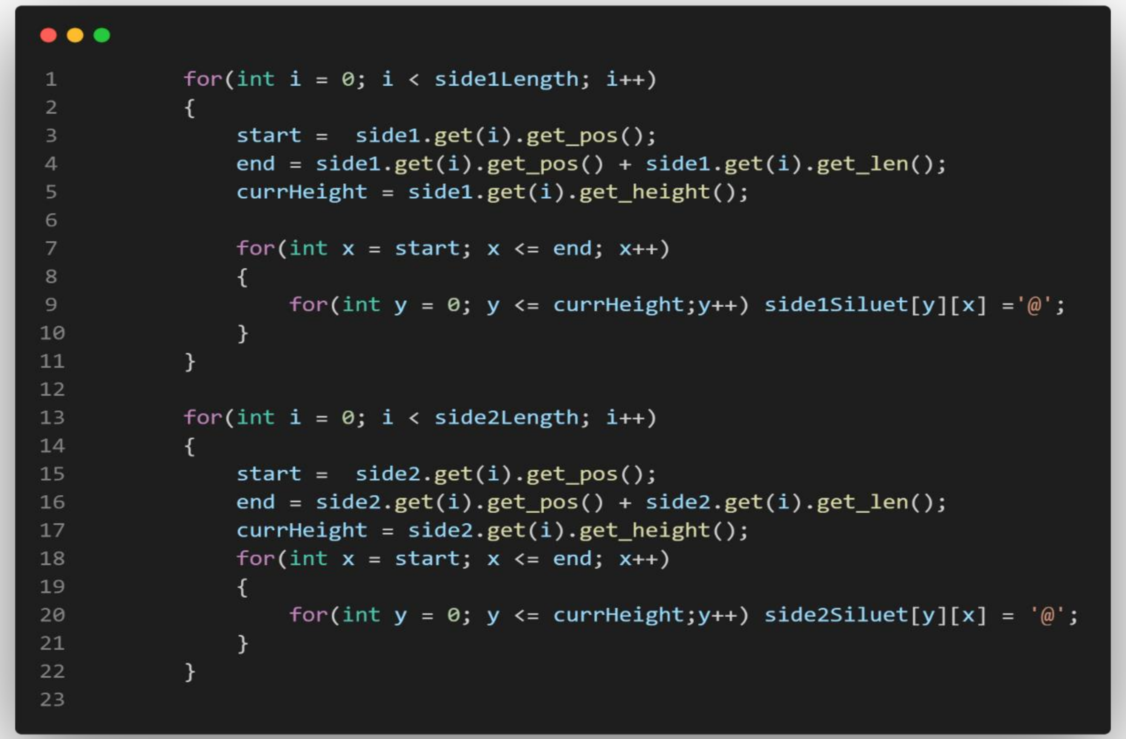
Example of Print Building Function with Different Data Structures

- Using Arrays



```
        for(int i = 0; i < side1Length; i++)
        {
            start =  side1[i].get_pos();
            end = side1[i].get_pos() + side1[i].get_len();
            currHeight = side1[i].get_height();
            for(int x = start; x <= end; x++)
            {
                for(int y = 0; y <= currHeight;y++) side1Siluet[y][x] ='@';
            }
        }

        for(int i = 0; i < side2Length; i++)
        {
            start =  side2[i].get_pos();
            end = side2[i].get_pos() + side2[i].get_len();
            currHeight = side2[i].get_height();
            for(int x = start; x <= end; x++)
            {
                for(int y = 0; y <= currHeight;y++) side2Siluet[y][x] = '@';
            }
        }

```

- Using Array list, Linked list, LDLinked list



```
        for(int i = 0; i < side1Length; i++)
        {
            start =  side1.get(i).get_pos();
            end = side1.get(i).get_pos() + side1.get(i).get_len();
            currHeight = side1.get(i).get_height();

            for(int x = start; x <= end; x++)
            {
                for(int y = 0; y <= currHeight;y++) side1Siluet[y][x] ='@';
            }
        }

        for(int i = 0; i < side2Length; i++)
        {
            start =  side2.get(i).get_pos();
            end = side2.get(i).get_pos() + side2.get(i).get_len();
            currHeight = side2.get(i).get_height();
            for(int x = start; x <= end; x++)
            {
                for(int y = 0; y <= currHeight;y++) side2Siluet[y][x] = '@';
            }
        }

```

Side1length and side2length represents the number of buildings on each side of the street.
Silhouette functions time complexities are related to the total number of buildings.

4 – Test Cases

a-) Testing the complexity and running of the street classes

1 – Create Street objects with each of the Data Structures
2 – Add Different Set of number of buildings to each street
3 – Compare Running Times
4 – Calculate Time complexity
5 – Testing the LDLinkedList Class

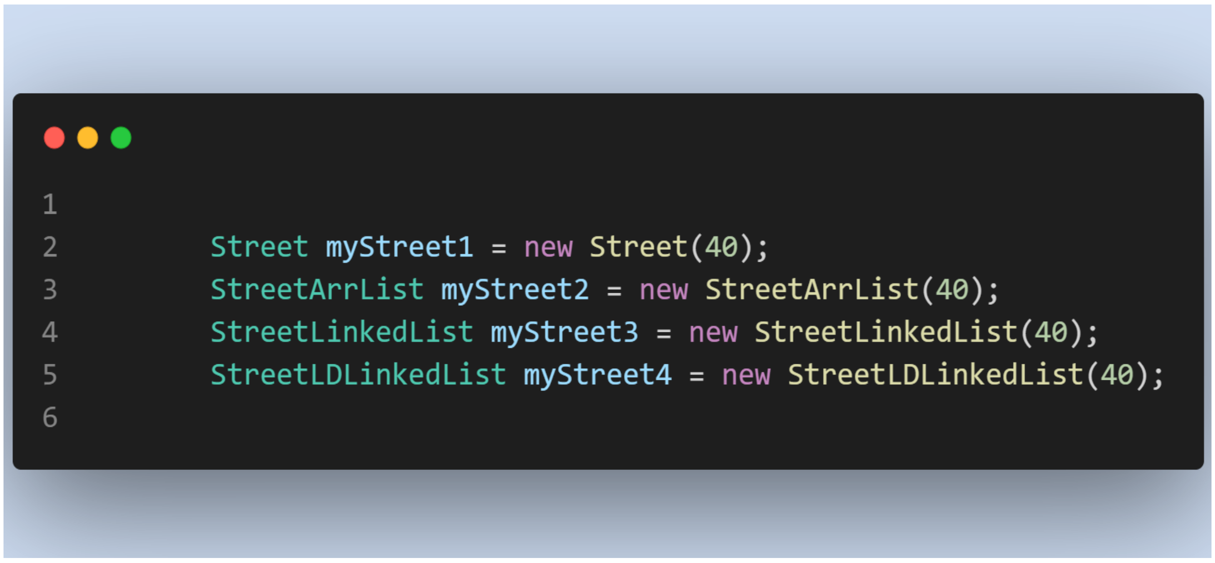
b-) Testing the functionality of the street classes

1- Create a Street
2- Create and Add Buildings
3- Display Silhouette
4- Display Details
5- Remove Buildings
6- Display Total Free Spaces
7- Display Total Used Spaces
8- Display Playground to Buildings Ratio
9- Print List of the Buildings

5 – Running Program and Results

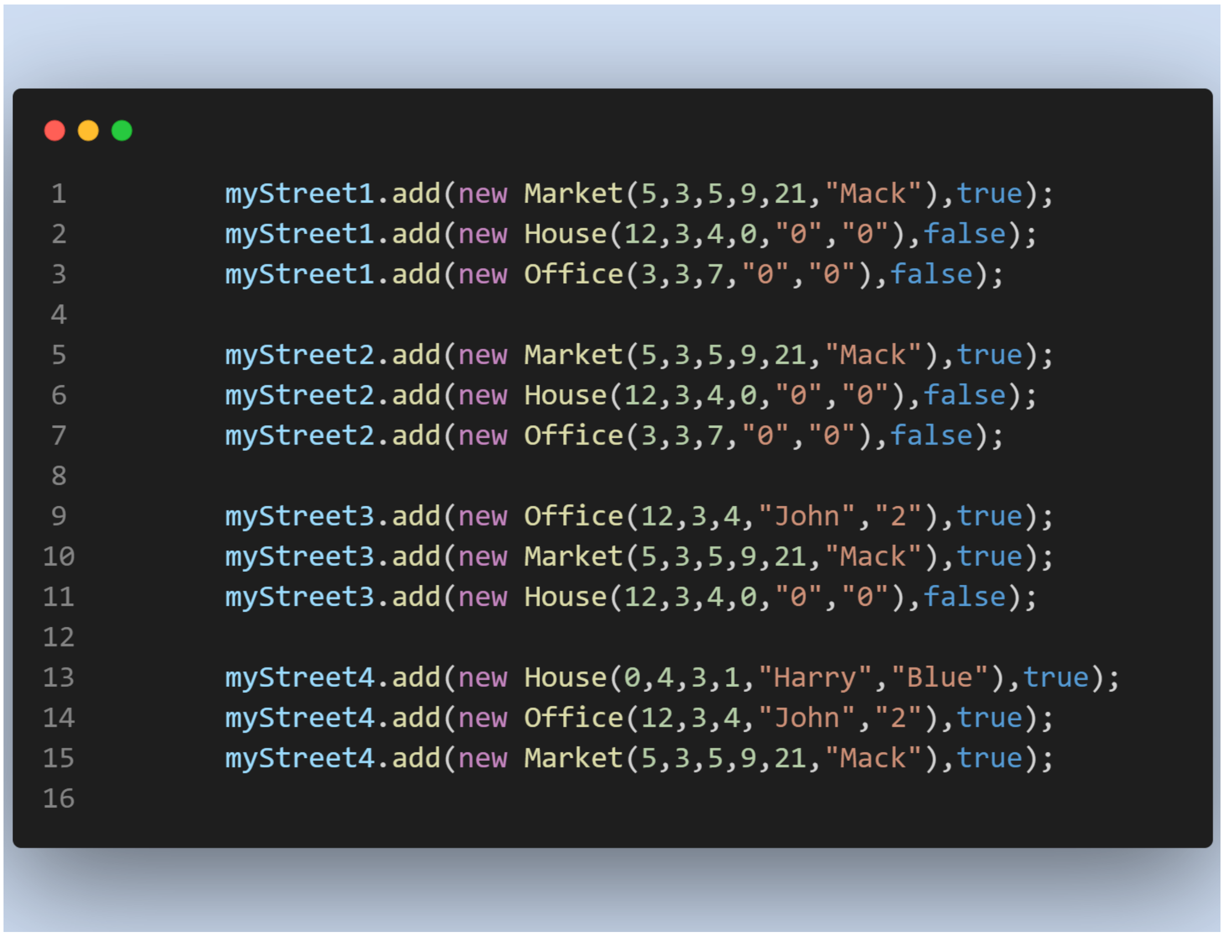
a-) Testing the complexity and running of the street classes

1 – Create Street objects with each of the Data Structures



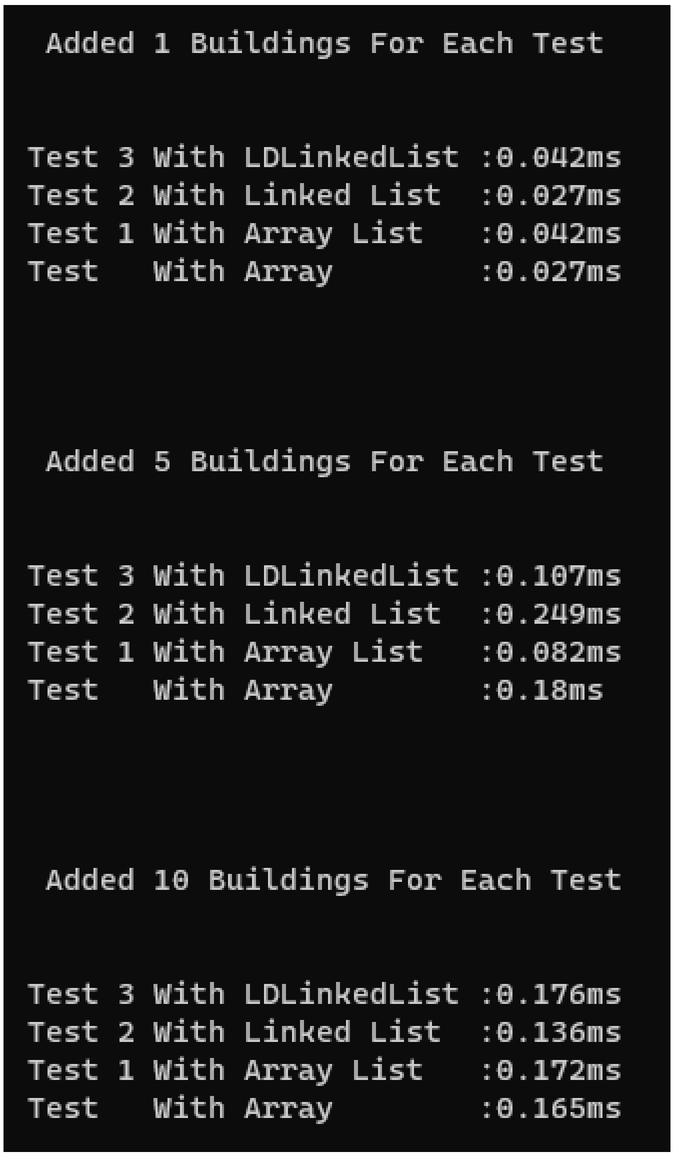
```java
        Street myStreet1 = new Street(40);
        StreetArrList myStreet2 = new StreetArrList(40);
        StreetLinkedList myStreet3 = new StreetLinkedList(40);
        StreetLDLinkedList myStreet4 = new StreetLDLinkedList(40);
```

2 – Add Different Set of number of buildings to each street

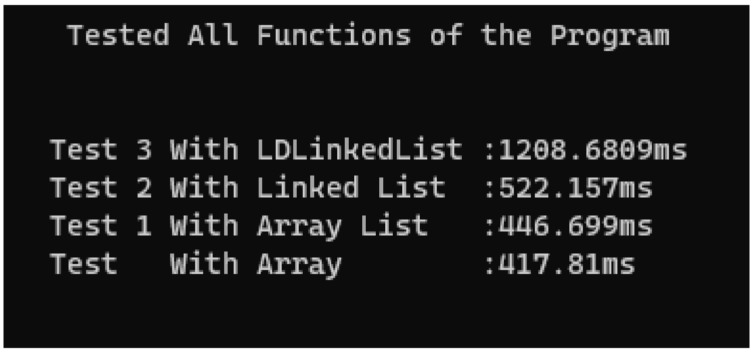


```java
        myStreet1.add(new Market(5,3,5,9,21,"Mack"),true);
        myStreet1.add(new House(12,3,4,0,"0","0"),false);
        myStreet1.add(new Office(3,3,7,"0","0"),false);

        myStreet2.add(new Market(5,3,5,9,21,"Mack"),true);
        myStreet2.add(new House(12,3,4,0,"0","0"),false);
        myStreet2.add(new Office(3,3,7,"0","0"),false);

        myStreet3.add(new Office(12,3,4,"John","2"),true);
        myStreet3.add(new Market(5,3,5,9,21,"Mack"),true);
        myStreet3.add(new House(12,3,4,0,"0","0"),false);

        myStreet4.add(new House(0,4,3,1,"Harry","Blue"),true);
        myStreet4.add(new Office(12,3,4,"John","2"),true);
        myStreet4.add(new Market(5,3,5,9,21,"Mack"),true);
```

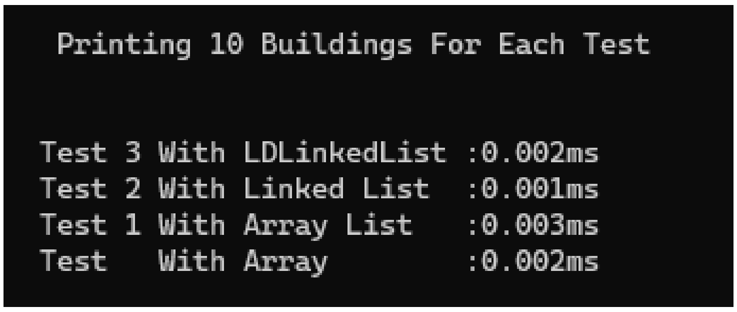
3 – Compare Running Times



```
 Added 1 Buildings For Each Test


Test 3 With LDLinkedList :0.042ms
Test 2 With Linked List  :0.027ms
Test 1 With Array List   :0.042ms
Test   With Array        :0.027ms



 Added 5 Buildings For Each Test


Test 3 With LDLinkedList :0.107ms
Test 2 With Linked List  :0.249ms
Test 1 With Array List   :0.082ms
Test   With Array        :0.18ms



 Added 10 Buildings For Each Test


Test 3 With LDLinkedList :0.176ms
Test 2 With Linked List  :0.136ms
Test 1 With Array List   :0.172ms
Test   With Array        :0.165ms
```



```
 Tested All Functions of the Program


Test 3 With LDLinkedList :1208.6809ms
Test 2 With Linked List  :522.157ms
Test 1 With Array List   :446.699ms
Test   With Array        :417.81ms
```



```
 Printing 10 Buildings For Each Test


Test 3 With LDLinkedList :0.002ms
Test 2 With Linked List  :0.001ms
Test 1 With Array List   :0.003ms
Test   With Array        :0.002ms
```

4 – Calculate Time complexity

-In all Classes Time Complexity is related to total number of Buildings (n),

Length of the street (x) and the maximum height of the buildings(y).

- The time Complexity for The Street Class that uses Array

  Tetha (n * x * y) = Tetha (N)

- The time Complexity for The Street Class that uses Array List

  Tetha (n * x * y) = Tetha(N)

- The time Complexity for The Street Class that uses Linked List

  Tetha (n*n * x * y) = Tetha(N*N)

- The time Complexity for The Street Class that uses LDLinked List

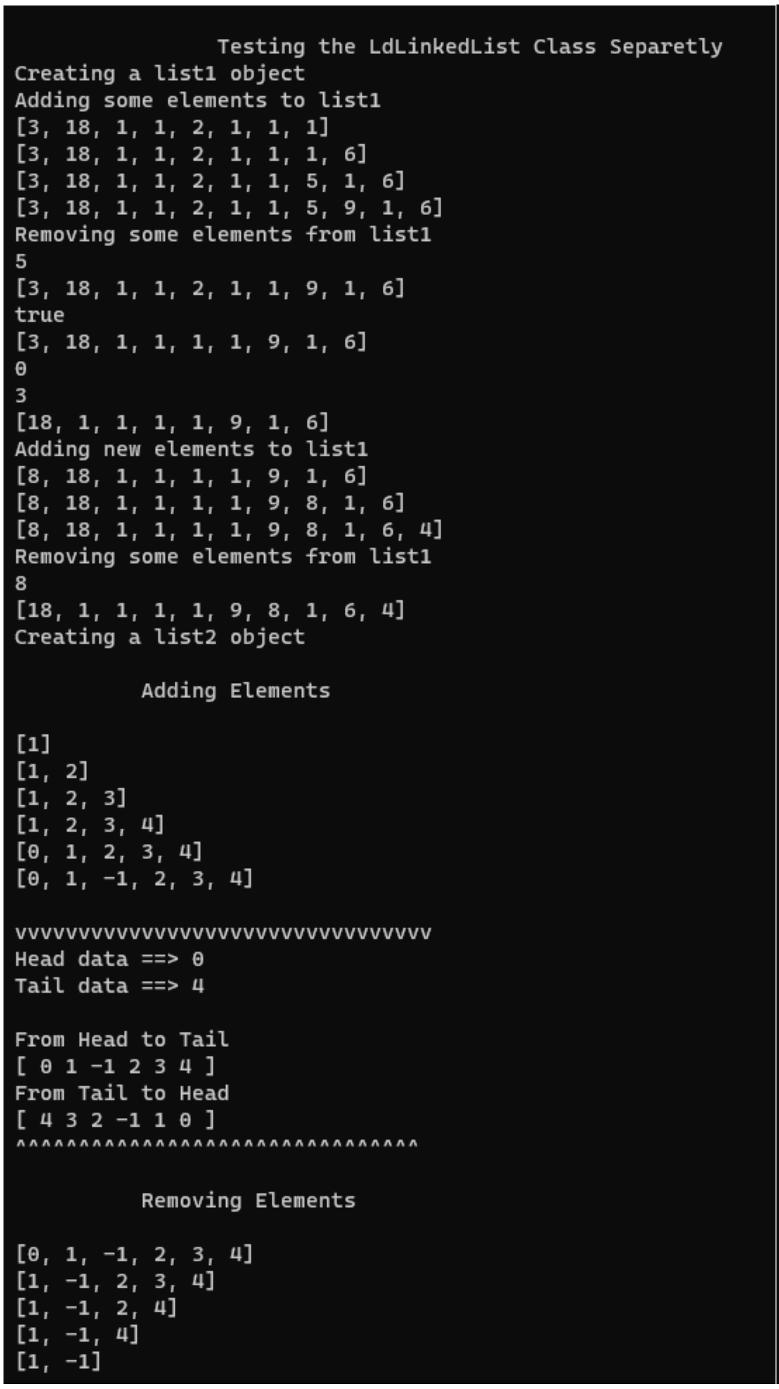
  Tetha (n*n * x * y) = Tetha(N*N)

For 1 Building Added	elements	Instructions
Array = 0.027ms	n = 1	n = 1
Array List= 0.042ms	n = 1	n = 1
Linked List = 0.027ms	n = 1	n*n = 1
LD Linked List = 0.042ms	n = 1	n *n = 1

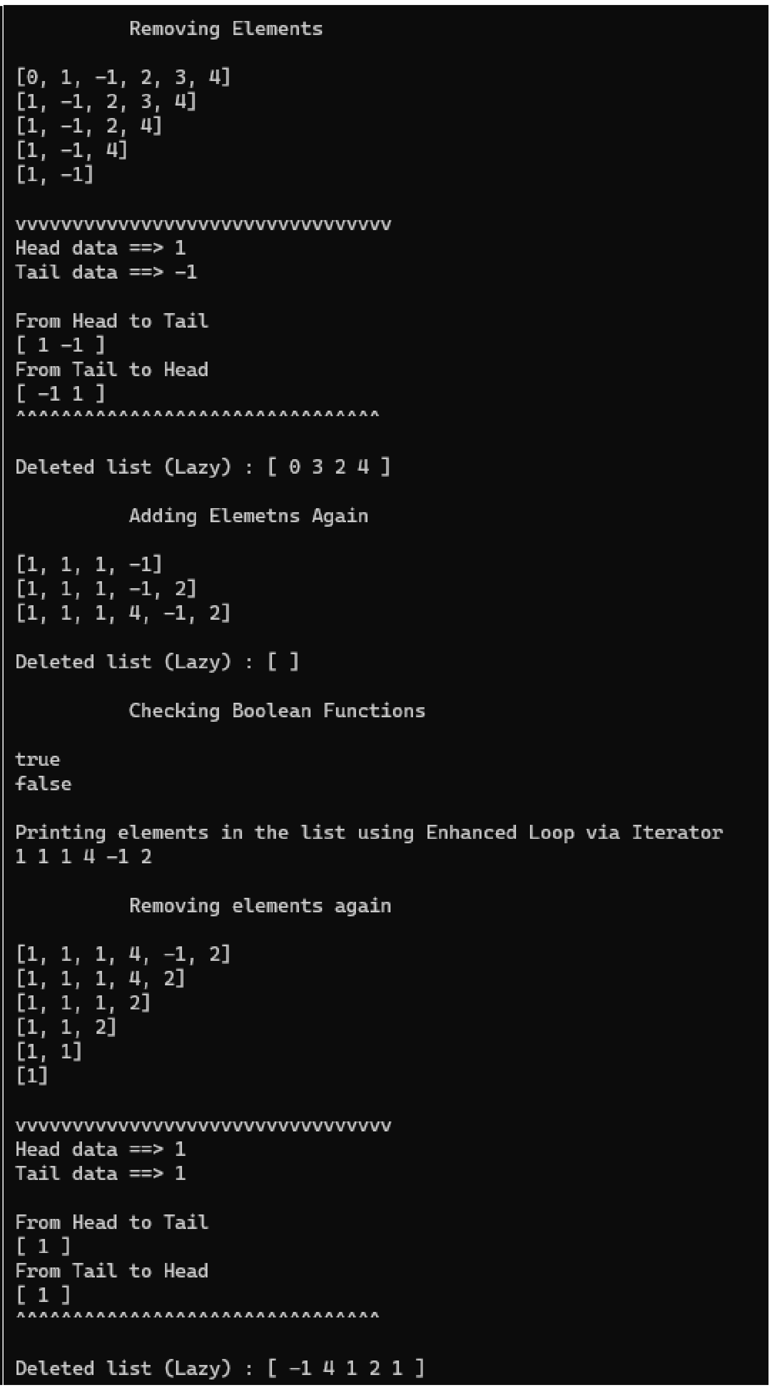
For 5 Building Added	elements	Instructions	Experimental Result
Array = 0.18ms	n = 5	n = 5	0.135
Array List= 0.082ms	n = 5	n = 5	0.21
Linked List = 0.249ms	n = 5	n*n = 25	0.135
LD Linked List = 0.107ms	n = 5	n *n = 25	0.21

For 10 Building Added	elements	Instructions	Experimental Result
Array = 0.176ms	n = 10	n = 10	0.27
Array List= 0.136ms	n = 10	n = 10	0.42
Linked List = 0.172ms	n = 10	n*n = 100	0.27
LD Linked List = 0.165ms	n = 10	n *n = 100	0.42

5 – Testing the LDLinkedList Class



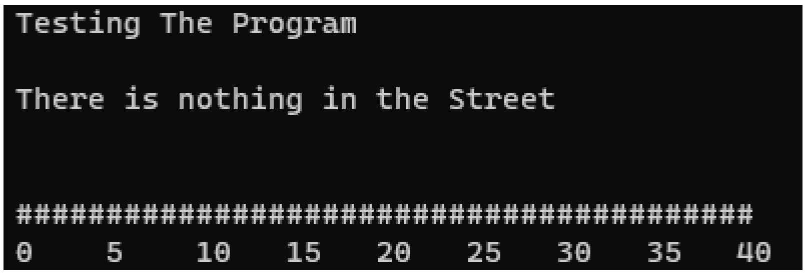
```
                 Testing the LdLinkedList Class Separetly
Creating a list1 object
Adding some elements to list1
[3, 18, 1, 1, 2, 1, 1, 1]
[3, 18, 1, 1, 2, 1, 1, 1, 6]
[3, 18, 1, 1, 2, 1, 1, 5, 1, 6]
[3, 18, 1, 1, 2, 1, 1, 5, 9, 1, 6]
Removing some elements from list1
5
[3, 18, 1, 1, 2, 1, 1, 9, 1, 6]
true
[3, 18, 1, 1, 1, 1, 9, 1, 6]
0
3
[18, 1, 1, 1, 1, 9, 1, 6]
Adding new elements to list1
[8, 18, 1, 1, 1, 1, 9, 1, 6]
[8, 18, 1, 1, 1, 1, 9, 8, 1, 6]
[8, 18, 1, 1, 1, 1, 9, 8, 1, 6, 4]
Removing some elements from list1
8
[18, 1, 1, 1, 1, 9, 8, 1, 6, 4]
Creating a list2 object

          Adding Elements

[1]
[1, 2]
[1, 2, 3]
[1, 2, 3, 4]
[0, 1, 2, 3, 4]
[0, 1, -1, 2, 3, 4]

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
Head data ==> 0
Tail data ==> 4

From Head to Tail
[ 0 1 -1 2 3 4 ]
From Tail to Head
[ 4 3 2 -1 1 0 ]
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

          Removing Elements

[0, 1, -1, 2, 3, 4]
[1, -1, 2, 3, 4]
[1, -1, 2, 4]
[1, -1, 4]
[1, -1]
```



```
          Removing Elements

[0, 1, -1, 2, 3, 4]
[1, -1, 2, 3, 4]
[1, -1, 2, 4]
[1, -1, 4]
[1, -1]

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
Head data ==> 1
Tail data ==> -1

From Head to Tail
[ 1 -1 ]
From Tail to Head
[ -1 1 ]
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Deleted list (Lazy) : [ 0 3 2 4 ]

          Adding Elemetns Again

[1, 1, 1, -1]
[1, 1, 1, -1, 2]
[1, 1, 1, 4, -1, 2]

Deleted list (Lazy) : [ ]

          Checking Boolean Functions

true
false

Printing elements in the list using Enhanced Loop via Iterator
1 1 1 4 -1 2

          Removing elements again

[1, 1, 1, 4, -1, 2]
[1, 1, 1, 4, 2]
[1, 1, 1, 2]
[1, 1, 2]
[1, 1]
[1]

vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv
Head data ==> 1
Tail data ==> 1

From Head to Tail
[ 1 ]
From Tail to Head
[ 1 ]
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Deleted list (Lazy) : [ -1 4 1 2 1 ]
```

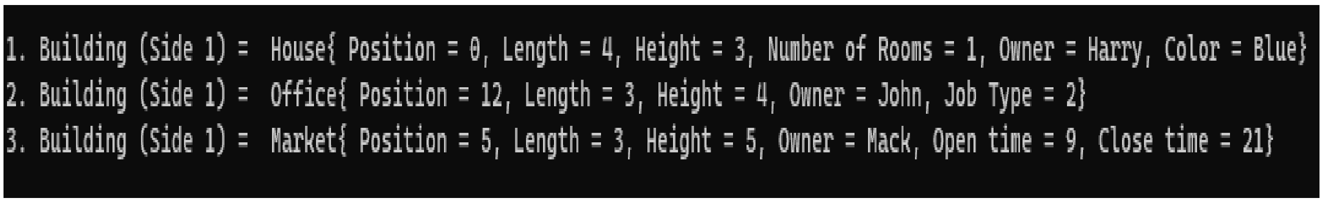
b-) Testing the functionality of the street classes

1 – Create a Street



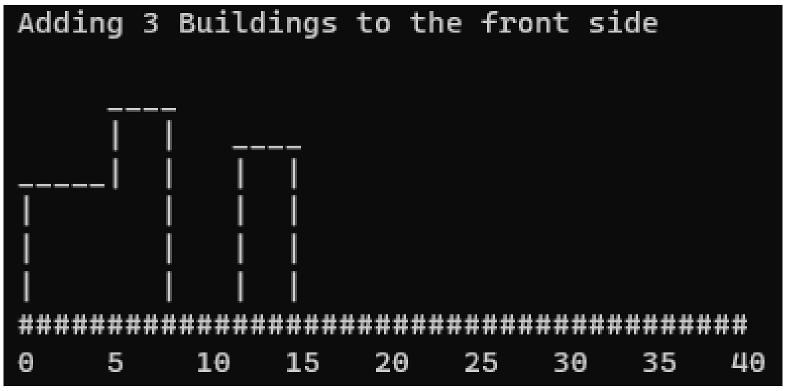
```
Testing The Program

There is nothing in the Street


#########################################
0    5    10   15   20   25   30   35   40
```

2 – Create And Add Buildings



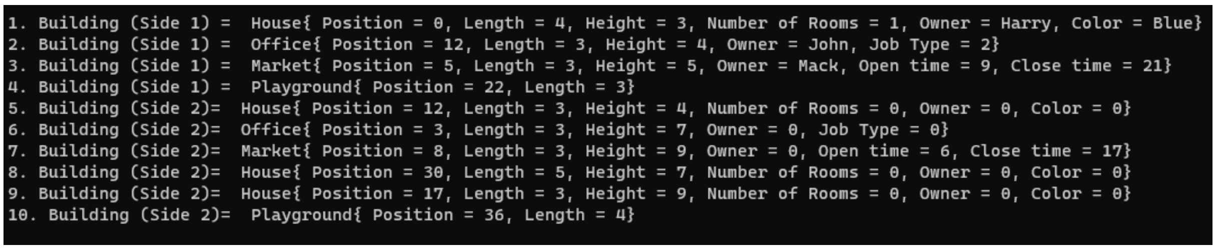
```
1. Building (Side 1) =  House{ Position = 0, Length = 4, Height = 3, Number of Rooms = 1, Owner = Harry, Color = Blue}
2. Building (Side 1) =  Office{ Position = 12, Length = 3, Height = 4, Owner = John, Job Type = 2}
3. Building (Side 1) =  Market{ Position = 5, Length = 3, Height = 5, Owner = Mack, Open time = 9, Close time = 21}
```

3 – Display Slihouette



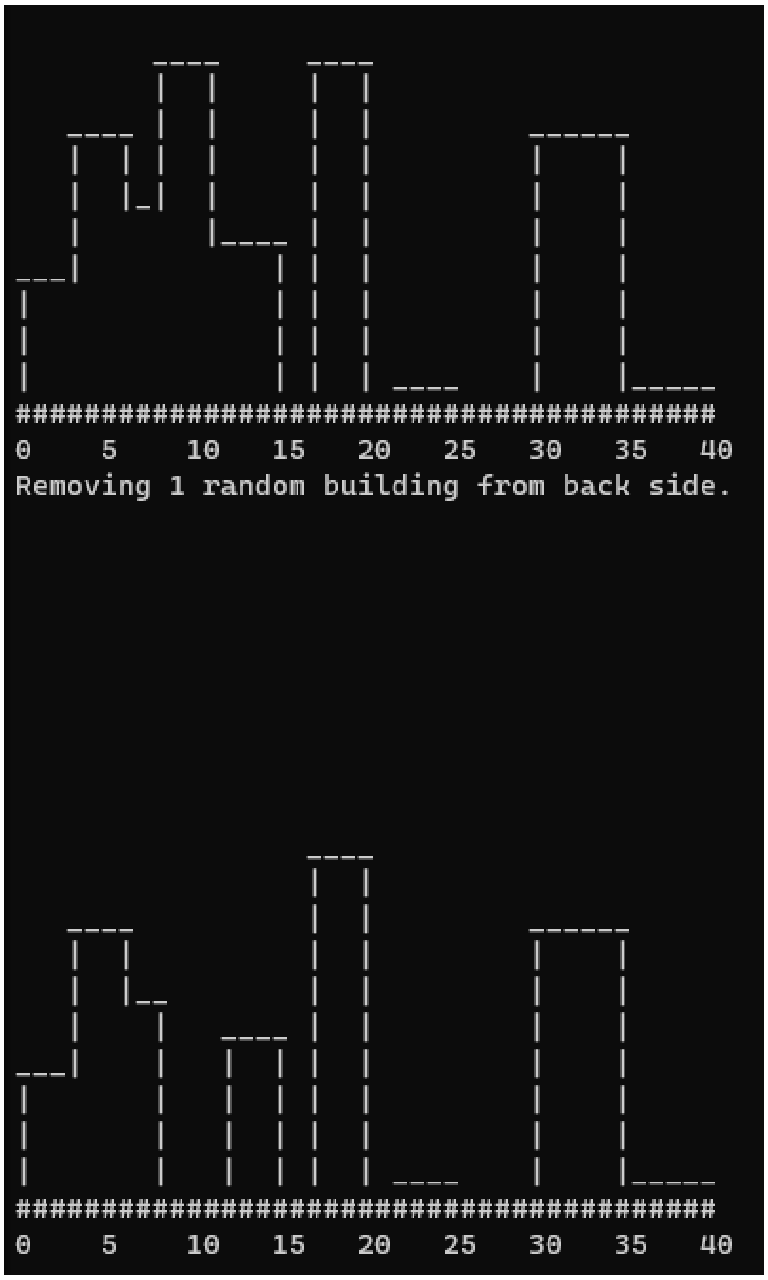
```
Adding 3 Buildings to the front side


     ----
     |  |   ----
-----|  |   |  |
|       |   |  |
|       |   |  |
|       |   |  |
#########################################
0    5    10   15   20   25   30   35   40
```

4- Display Details



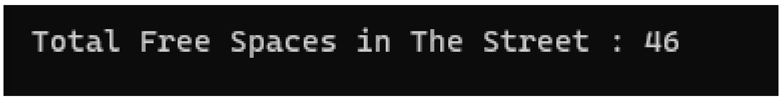
```
1. Building (Side 1) =  House{ Position = 0, Length = 4, Height = 3, Number of Rooms = 1, Owner = Harry, Color = Blue}
2. Building (Side 1) =  Office{ Position = 12, Length = 3, Height = 4, Owner = John, Job Type = 2}
3. Building (Side 1) =  Market{ Position = 5, Length = 3, Height = 5, Owner = Mack, Open time = 9, Close time = 21}
4. Building (Side 1) =  Playground{ Position = 22, Length = 3}
5. Building (Side 2)=  House{ Position = 12, Length = 3, Height = 4, Number of Rooms = 0, Owner = 0, Color = 0}
6. Building (Side 2)=  Office{ Position = 3, Length = 3, Height = 7, Owner = 0, Job Type = 0}
7. Building (Side 2)=  Market{ Position = 8, Length = 3, Height = 9, Owner = 0, Open time = 6, Close time = 17}
8. Building (Side 2)=  House{ Position = 30, Length = 5, Height = 7, Number of Rooms = 0, Owner = 0, Color = 0}
9. Building (Side 2)=  House{ Position = 17, Length = 3, Height = 9, Number of Rooms = 0, Owner = 0, Color = 0}
10. Building (Side 2)=  Playground{ Position = 36, Length = 4}
```

5 - Remove Buildings



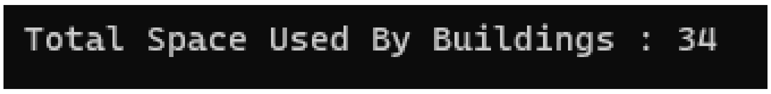
```
        ____     ____
        |  |     |  |
   ____ |  |     |  |         ______
   |  | |  |     |  |         |    |
   |  |_|  |     |  |         |    |
   |       |____ |  |         |    |
___|           | |  |         |    |
|              | |  |         |    |
|              | |  |         |    |
|              | |  | ____    |    |_____
#########################################
0    5    10   15   20   25   30   35   40
Removing 1 random building from back side.




                 ____
                 |  |
   ____          |  |         ______
   |  |          |  |         |    |
   |  |__        |  |         |    |
   |    |   ____ |  |         |    |
___|    |   |  | |  |         |    |
|       |   |  | |  |         |    |
|       |   |  | |  |         |    |
|       |   |  | |  | ____    |    |_____
#########################################
0    5    10   15   20   25   30   35   40
```

6 – Display Total Free Spaces



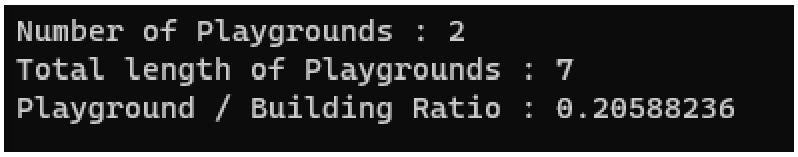
```
 Total Free Spaces in The Street : 46
```

7 – Display Total Used Spaces



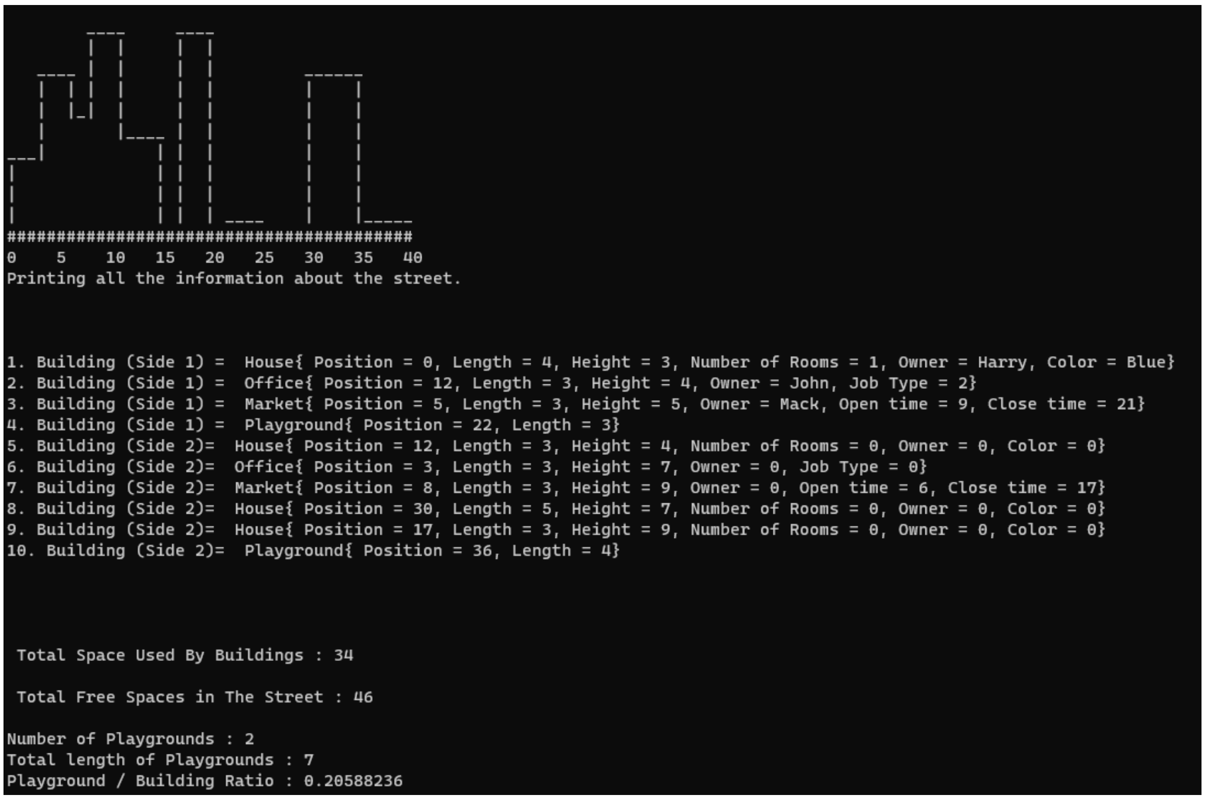
```
 Total Space Used By Buildings : 34
```

8 – Display Playground Ratio



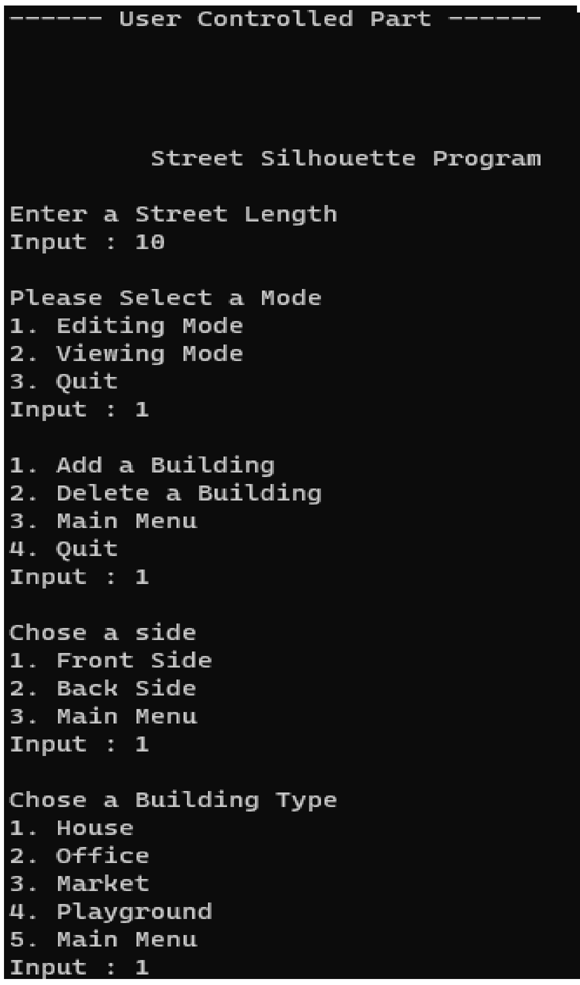
```
Number of Playgrounds : 2
Total length of Playgrounds : 7
Playground / Building Ratio : 0.20588236
```

9 – Display List of Buildings

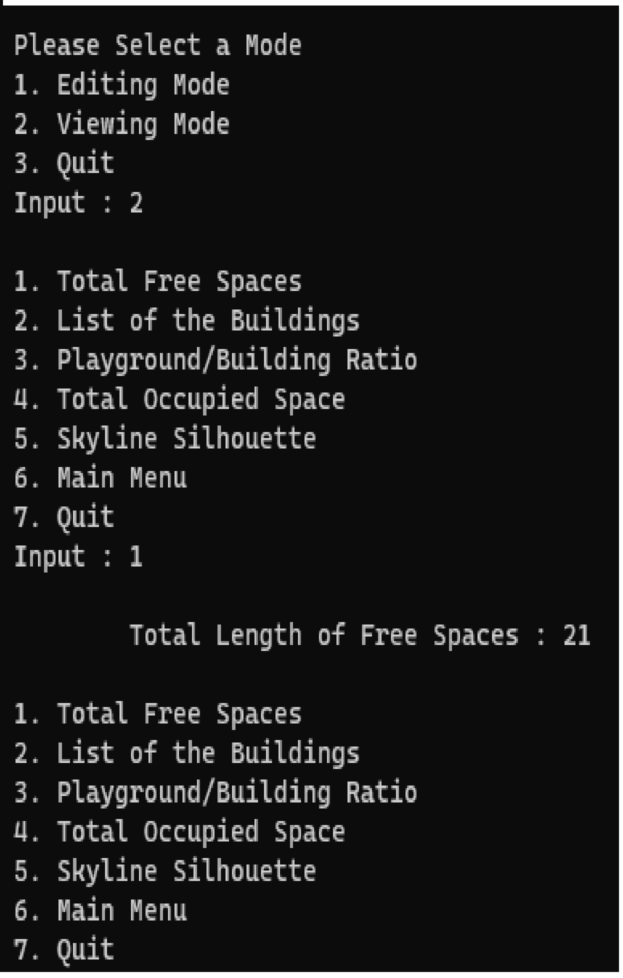


```
        ____     ____
        |  |     |  |
   ____ |  |     |  |         ______
   |  | |  |     |  |         |    |
   |  |_|  |     |  |         |    |
   |       |____ |  |         |    |
___|           | |  |         |    |
|              | |  |         |    |
|              | |  |         |    |
|              | |  | ____    |    |_____
#########################################
0    5    10   15   20   25   30   35   40
Printing all the information about the street.



1. Building (Side 1) =  House{ Position = 0, Length = 4, Height = 3, Number of Rooms = 1, Owner = Harry, Color = Blue}
2. Building (Side 1) =  Office{ Position = 12, Length = 3, Height = 4, Owner = John, Job Type = 2}
3. Building (Side 1) =  Market{ Position = 5, Length = 3, Height = 5, Owner = Mack, Open time = 9, Close time = 21}
4. Building (Side 1) =  Playground{ Position = 22, Length = 3}
5. Building (Side 2)=  House{ Position = 12, Length = 3, Height = 4, Number of Rooms = 0, Owner = 0, Color = 0}
6. Building (Side 2)=  Office{ Position = 3, Length = 3, Height = 7, Owner = 0, Job Type = 0}
7. Building (Side 2)=  Market{ Position = 8, Length = 3, Height = 9, Owner = 0, Open time = 6, Close time = 17}
8. Building (Side 2)=  House{ Position = 30, Length = 5, Height = 7, Number of Rooms = 0, Owner = 0, Color = 0}
9. Building (Side 2)=  House{ Position = 17, Length = 3, Height = 9, Number of Rooms = 0, Owner = 0, Color = 0}
10. Building (Side 2)=  Playground{ Position = 36, Length = 4}



 Total Space Used By Buildings : 34

 Total Free Spaces in The Street : 46

Number of Playgrounds : 2
Total length of Playgrounds : 7
Playground / Building Ratio : 0.20588236
```

Lastly User Interface Menus



```
------ User Controlled Part ------



         Street Silhouette Program

Enter a Street Length
Input : 10

Please Select a Mode
1. Editing Mode
2. Viewing Mode
3. Quit
Input : 1

1. Add a Building
2. Delete a Building
3. Main Menu
4. Quit
Input : 1

Chose a side
1. Front Side
2. Back Side
3. Main Menu
Input : 1

Chose a Building Type
1. House
2. Office
3. Market
4. Playground
5. Main Menu
Input : 1
```



```
Please Select a Mode
1. Editing Mode
2. Viewing Mode
3. Quit
Input : 2

1. Total Free Spaces
2. List of the Buildings
3. Playground/Building Ratio
4. Total Occupied Space
5. Skyline Silhouette
6. Main Menu
7. Quit
Input : 1

        Total Length of Free Spaces : 21

1. Total Free Spaces
2. List of the Buildings
3. Playground/Building Ratio
4. Total Occupied Space
5. Skyline Silhouette
6. Main Menu
7. Quit
```