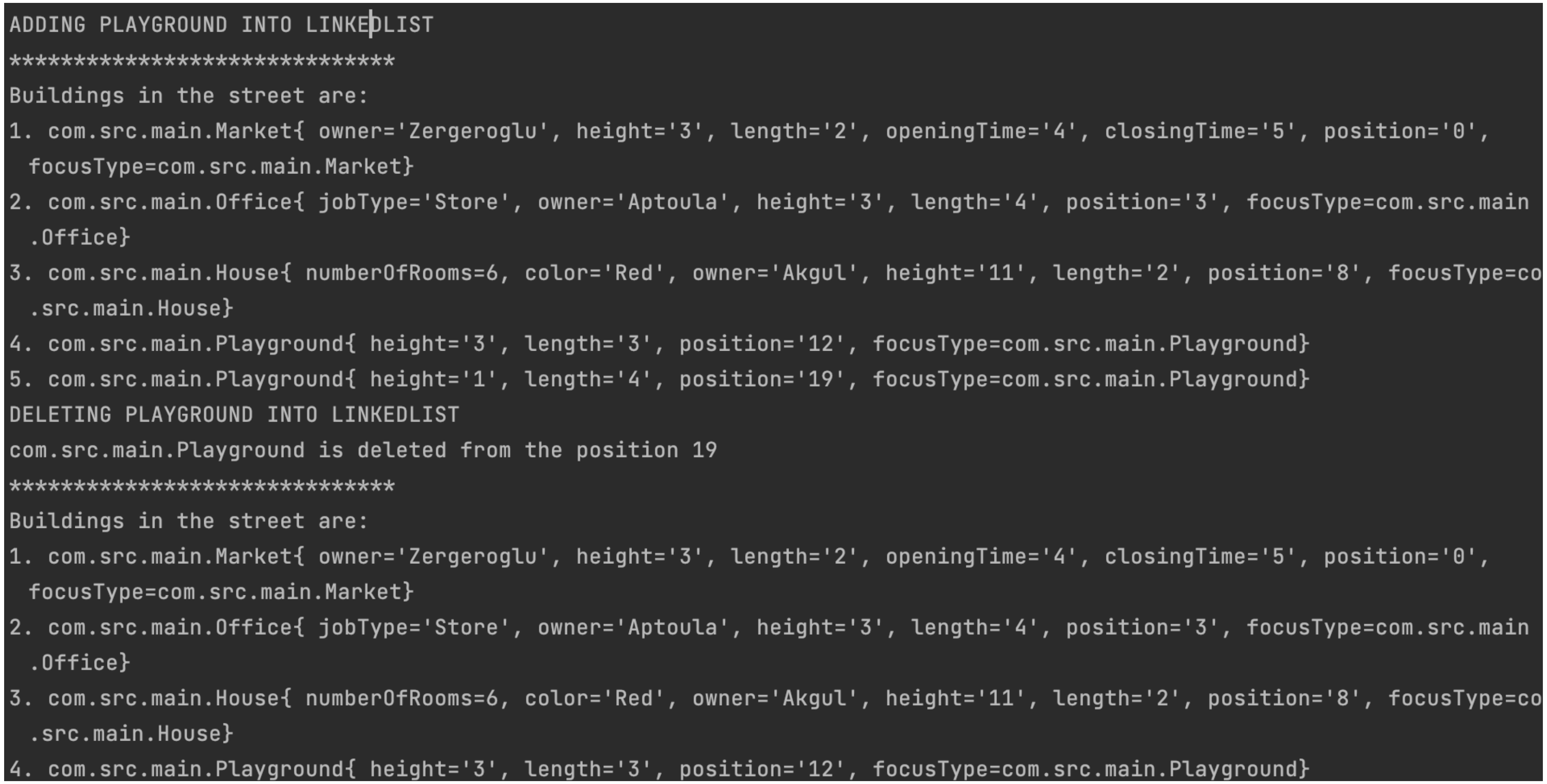
BURAK UGAR 200104004100

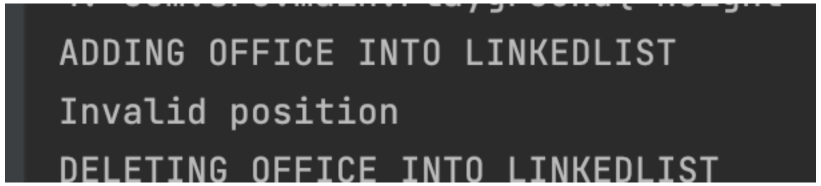
Note: Some of the inputs of the methods in linkedlist throws exception to show the exceptions are implemented true. The other ones implemented as usual.

For example:

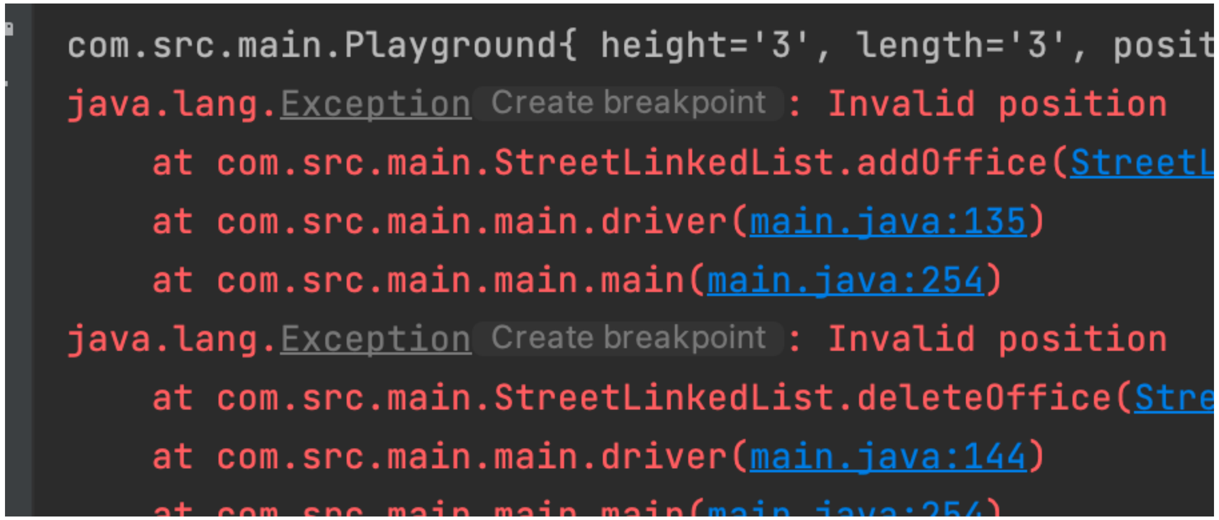


```
ADDING PLAYGROUND INTO LINKEDLIST
******************************
Buildings in the street are:
1. com.src.main.Market{ owner='Zergeroglu', height='3', length='2', openingTime='4', closingTime='5', position='0',
  focusType=com.src.main.Market}
2. com.src.main.Office{ jobType='Store', owner='Aptoula', height='3', length='4', position='3', focusType=com.src.main
  .Office}
3. com.src.main.House{ numberOfRooms=6, color='Red', owner='Akgul', height='11', length='2', position='8', focusType=co
  .src.main.House}
4. com.src.main.Playground{ height='3', length='3', position='12', focusType=com.src.main.Playground}
5. com.src.main.Playground{ height='1', length='4', position='19', focusType=com.src.main.Playground}
DELETING PLAYGROUND INTO LINKEDLIST
com.src.main.Playground is deleted from the position 19
******************************
Buildings in the street are:
1. com.src.main.Market{ owner='Zergeroglu', height='3', length='2', openingTime='4', closingTime='5', position='0',
  focusType=com.src.main.Market}
2. com.src.main.Office{ jobType='Store', owner='Aptoula', height='3', length='4', position='3', focusType=com.src.main
  .Office}
3. com.src.main.House{ numberOfRooms=6, color='Red', owner='Akgul', height='11', length='2', position='8', focusType=co
  .src.main.House}
4. com.src.main.Playground{ height='3', length='3', position='12', focusType=com.src.main.Playground}
```

Adding and deleting from linked list works well but adding and deleting Office throws exception since the given building is not on point.



```
ADDING OFFICE INTO LINKEDLIST
Invalid position
DELETING OFFICE INTO LINKEDLIST
```



```
com.src.main.Playground{ height='3', length='3', posit
java.lang.Exception Create breakpoint : Invalid position
    at com.src.main.StreetLinkedList.addOffice(StreetL
    at com.src.main.main.driver(main.java:135)
    at com.src.main.main.main(main.java:254)
java.lang.Exception Create breakpoint : Invalid position
    at com.src.main.StreetLinkedList.deleteOffice(Stre
    at com.src.main.main.driver(main.java:144)
    at com.src.main.main.main(main.java:254)
```

1. SYSTEM REQUIREMENTS

1.a

There will be a street, and the street will have two sides. One is on the left, and the other is on the right.

This is an example of a street specification:

• The length of the street can be set by the user.

• The street has two sides, both of which are designed.

• The street may contain three types of structures (houses, offices, and markets) as well as playgrounds. Please keep in mind that not all types of buildings must be available on the street.

• The user determines the buildings' position, length (the length on the street, not the depth of the buildings), and height. A building can only be built on a street lot if there is enough space for it in terms of length. There are four types of structures. Buildings include a market, an office, a playground, and a house. These structures have the following characteristics:

• The houses differ in terms of the number of rooms, color, and owner.

• The offices have properties for job type and owner; the markets have properties for owner and opening/closing times.

• The app allows you to focus on a certain building. When focused on a building, the user is presented with many forms of information:

• The house introduces its owner, • the office introduces its work sorts, • the playground introduces its length, and • the market introduces its closing time.

There are two modes of operation for the application for the user:

1-) In Editing mode, place a structure on a plot of land in the street.

• demolish a structure on a plot of land in the street.

2-) In viewing mode, show the complete length of lands on the roadway.

• show a list of the buildings on the street

• show the number

3. APPROACH TO PROBLEM SOLVING

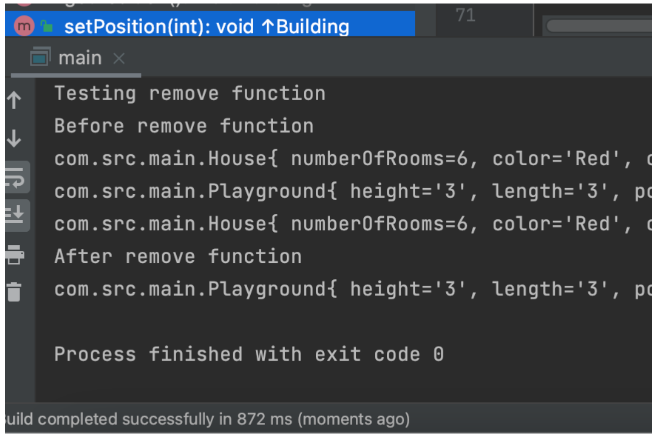
Creating and developing a city planning program to be utilized in the creation of a small one-street town.
Many classes were created to help with this process.
AbstractSide is a class that was built for left and right side specifications. It remembers the place, buildings, and building numbers. Basic operations are covered in this lesson.
Class Building This class was built to store building information and abilities such as height, length, position, and so on. This is the basic class for the Market, Office, and House classes.
Class House This class was built to store information and abilities about the house.
This class was intended to store office information and skills.
Market This class was intended to store market information and skills.
Edit Mode for Users
This class was built to keep edits.
In link list implementation:
I HAVE OVERRIDDEN THE ADD, REMOVE LIST METHODS SO THERE IS NO NEED TO CONTROL EXCEPTION BECAUSE THEY HAVE THEIR OWN EXCEPTION HANDLING METHODS.
For printing the skyline silhoutte, I use divide and concuer algorithm.
Given n rectangular structures in a two-dimensional metropolis, computes their skylines while ignoring hidden lines. The main goal is to look at buildings from different angles and delete any elements that are not visible.

Every building has a common bottom, and each building is represented by a triplet (left, ht, right)

'left': is the x coordinate of the left side (or wall).
'right' is the x coordinate of the right side.
'ht': the height of the building.
A skyline is made up of rectangular strips. A rectangular strip is represented by the pair (left, ht), where left is the x coordinate of the strip's left side and ht is the height of the strip.
A simple solution is to leave the skyline or result vacant at first, then gradually add buildings to the skyline. A building is added by first locating the overlapping strip (s). If there are no overlapping strips, the new structure adds a new strip (s). If an overlapping strip is discovered, the height of the present strip may grow. The time complexity of this solution is O. (n2)
How Do We Combine Two Skylines?

The concept is similar to merge sort in that it begins with the initial strips of two skylines and compares x coordinates. Select the strip with the smallest x coordinate and add it to the result. The height of the extra strip is calculated as the maximum of the present heights from skylines 1 and 2.

TEST CASE:
WITH INPUT SIZE 500:

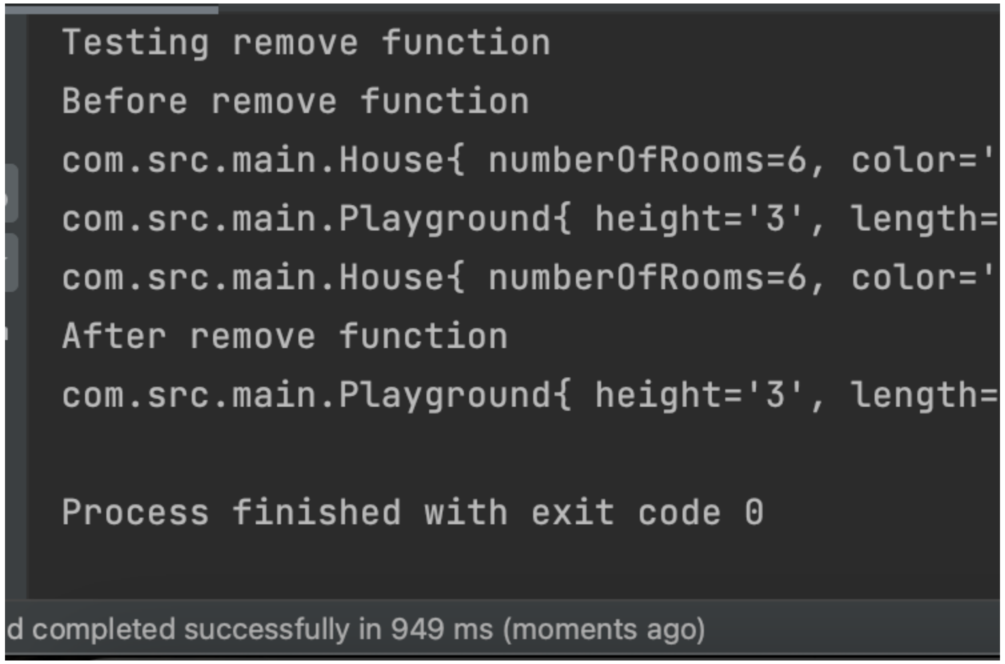


```
Testing remove function
Before remove function
com.src.main.House{ numberOfRooms=6, color='Red', c
com.src.main.Playground{ height='3', length='3', po
com.src.main.House{ numberOfRooms=6, color='Red', c
After remove function
com.src.main.Playground{ height='3', length='3', po

Process finished with exit code 0
```

uild completed successfully in 872 ms (moments ago)

WITH INPUT SIZE: 1000

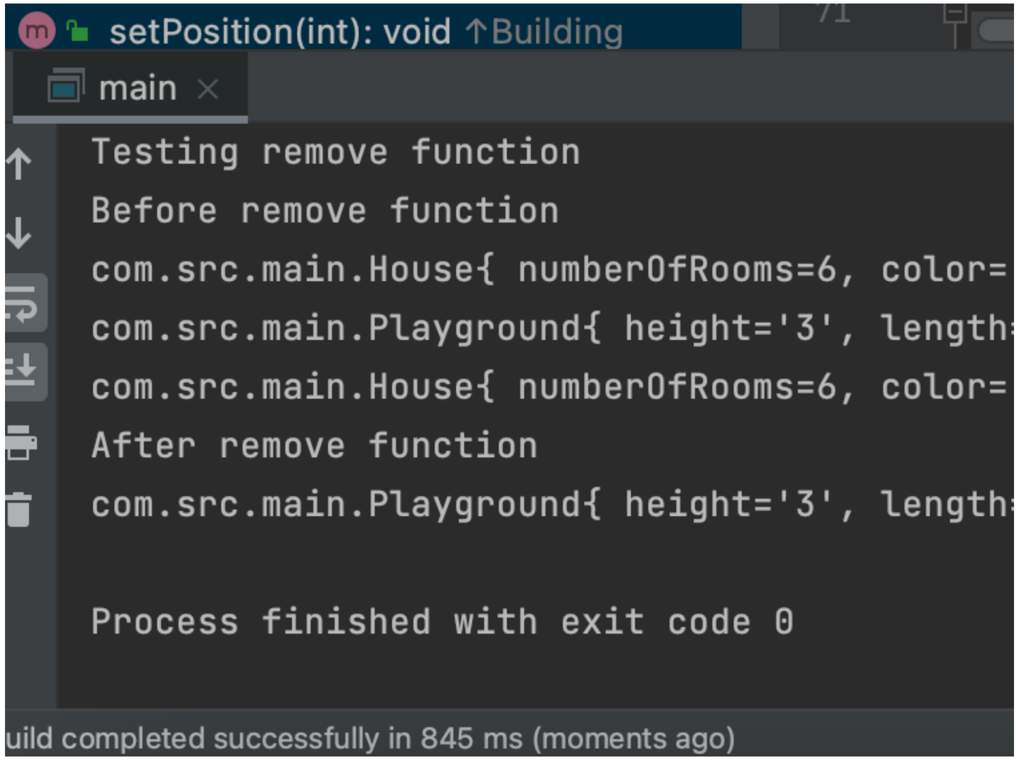


```
Testing remove function
Before remove function
com.src.main.House{ numberOfRooms=6, color='
com.src.main.Playground{ height='3', length=
com.src.main.House{ numberOfRooms=6, color='
After remove function
com.src.main.Playground{ height='3', length=

Process finished with exit code 0
```

d completed successfully in 949 ms (moments ago)

WITH INPUT SIZE: 200

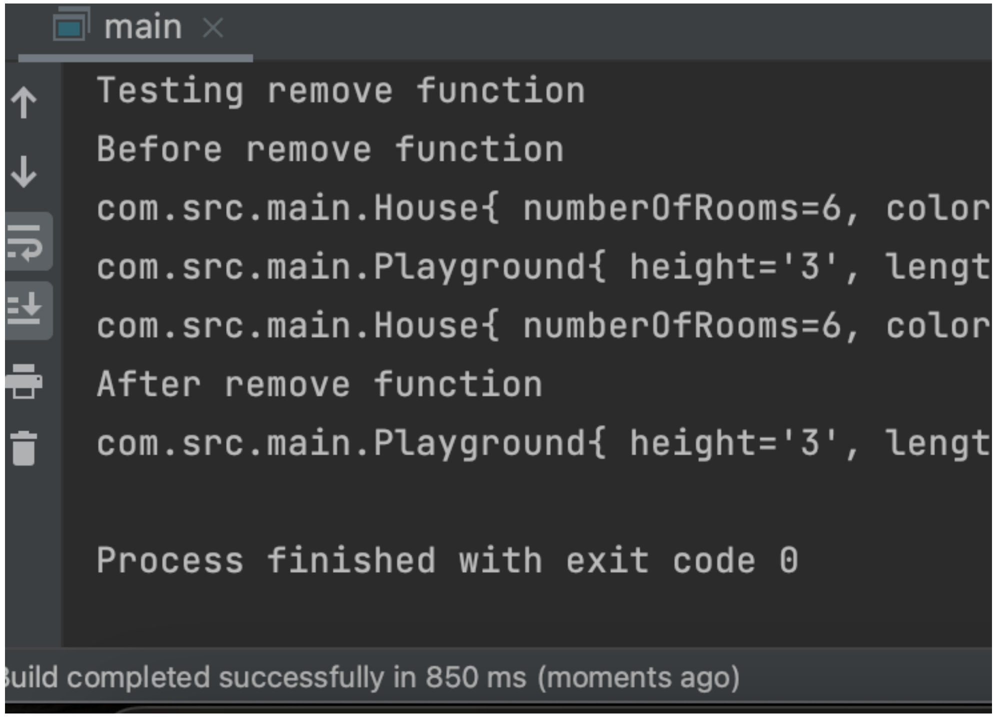


```
Testing remove function
Before remove function
com.src.main.House{ numberOfRooms=6, color=
com.src.main.Playground{ height='3', length
com.src.main.House{ numberOfRooms=6, color=
After remove function
com.src.main.Playground{ height='3', length

Process finished with exit code 0
```

uild completed successfully in 845 ms (moments ago)

WITH SIZE 250

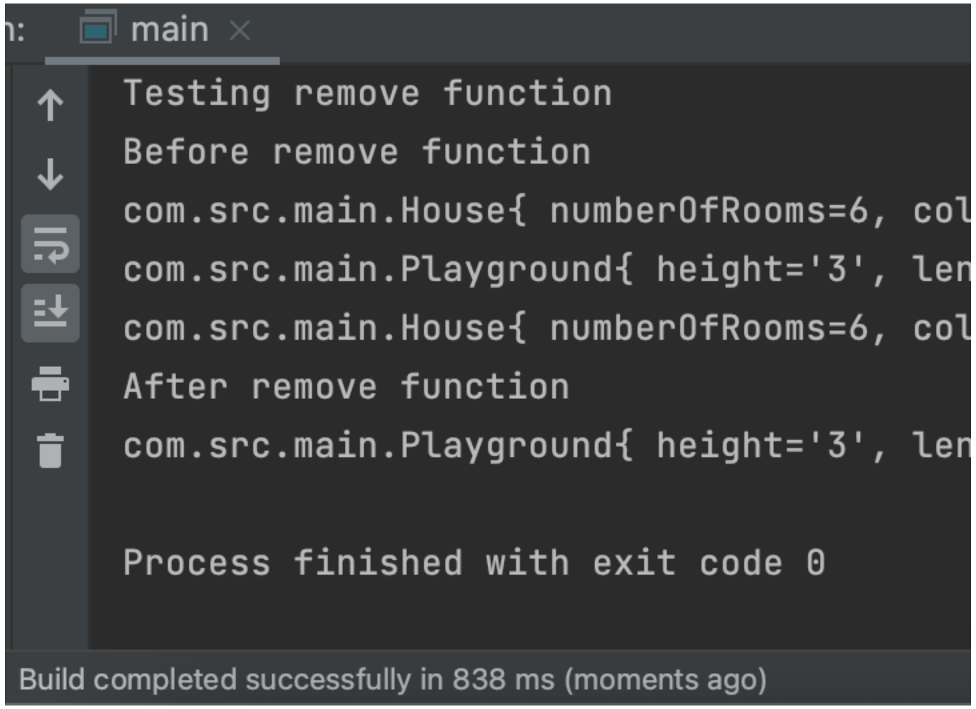


```
Testing remove function
Before remove function
com.src.main.House{ numberOfRooms=6, color
com.src.main.Playground{ height='3', lengt
com.src.main.House{ numberOfRooms=6, color
After remove function
com.src.main.Playground{ height='3', lengt

Process finished with exit code 0
```

```
uild completed successfully in 850 ms (moments ago)
```

WITH SIZE 150:

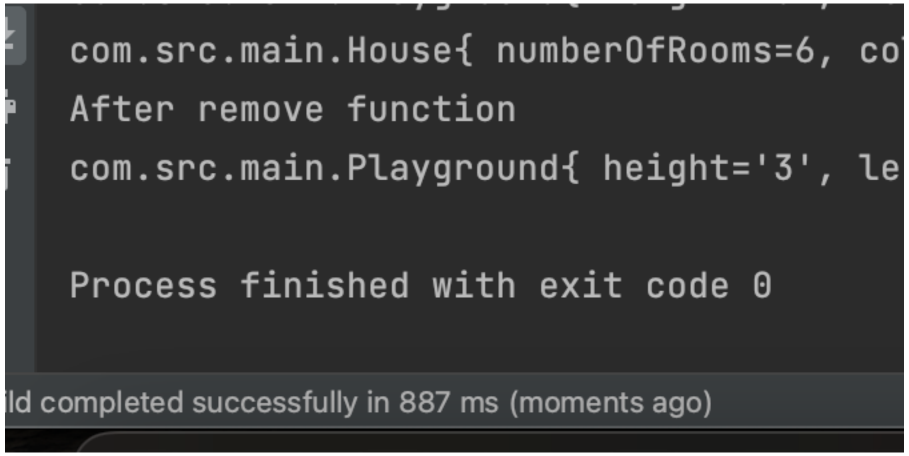


```
Testing remove function
Before remove function
com.src.main.House{ numberOfRooms=6, col
com.src.main.Playground{ height='3', len
com.src.main.House{ numberOfRooms=6, col
After remove function
com.src.main.Playground{ height='3', len

Process finished with exit code 0
```

Build completed successfully in 838 ms (moments ago)

WITH SIZE: 450



```
com.src.main.House{ numberOfRooms=6, co
After remove function
com.src.main.Playground{ height='3', le

Process finished with exit code 0
```

ild completed successfully in 887 ms (moments ago)

OVERALL TIME COMPLEXITY Q(n^2)

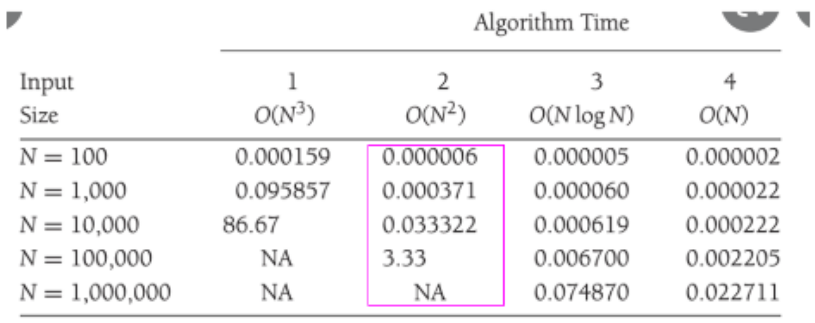
AS THE INPUT SIZE GROWS, COMPILE TIME GROWS USUALLY.

1. In practice, the size of the number has almost no effect on runtime. In practice, the array's numbers are all encoded with the same number of bits. If we use an int32, it will be 32 bits; if we use an int16, it will be 16 bits; and so on. The number 1 encoded in int32 uses the same number of bits (32) as the massive number 231 encoded in int32.

2. Theoretically, the size of the number absolutely affects runtime. For example, if the size of an element in the array was 10^2000.

3.Let's say the array has size n, and as n grows to infinity, time becomes quadratic; however, for small sizes, compile-time varies. As a result, we can say that for larger array sizes, time changes theoretically, but not for small sizes.

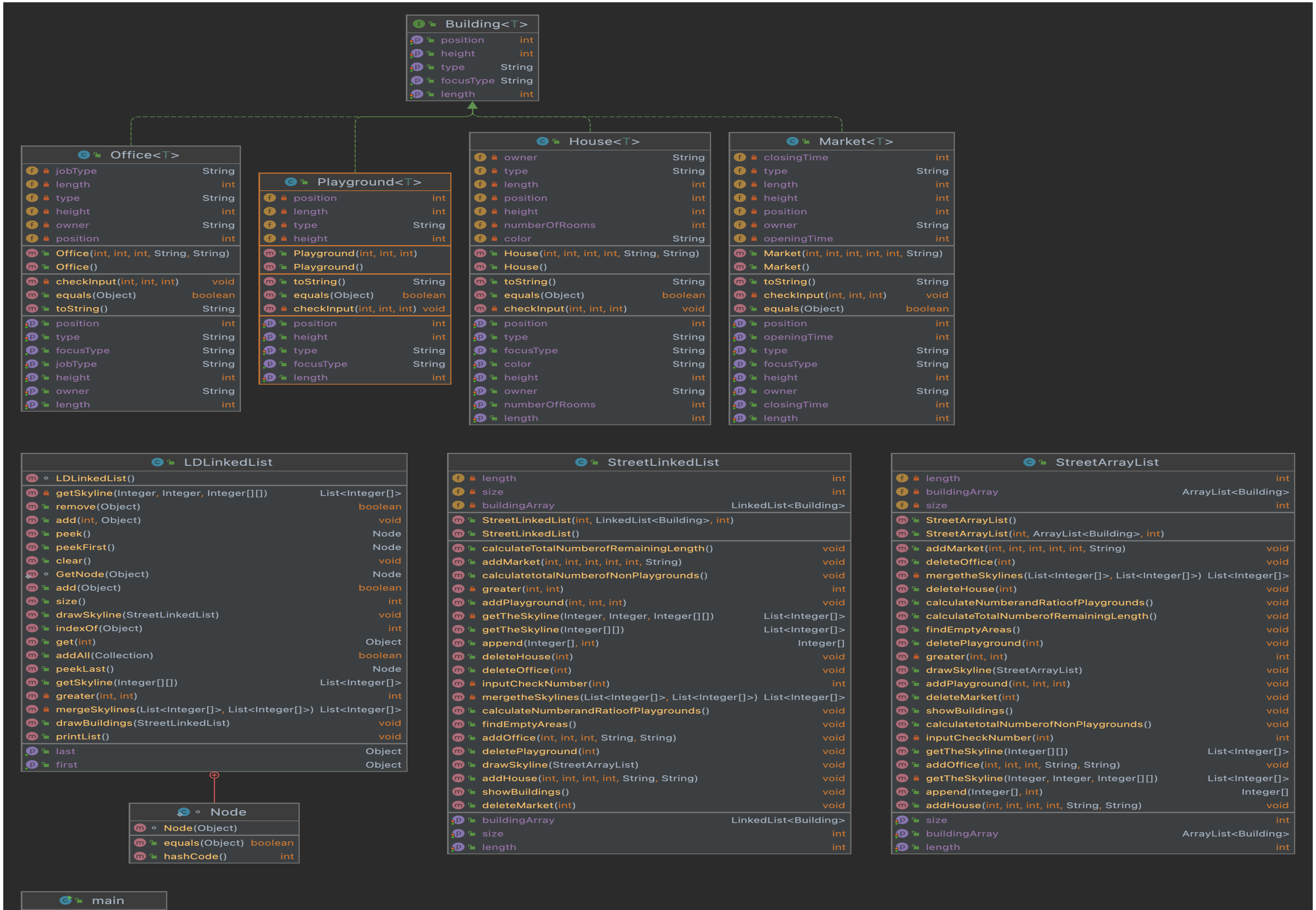
Theoretical results:



	Algorithm Time			
Input Size	1 $O(N^3)$	2 $O(N^2)$	3 $O(N \log N)$	4 $O(N)$
N = 100	0.000159	0.000006	0.000005	0.000002
N = 1,000	0.095857	0.000371	0.000060	0.000022
N = 10,000	86.67	0.033322	0.000619	0.000222
N = 100,000	NA	3.33	0.006700	0.002205
N = 1,000,000	NA	NA	0.074870	0.022711

7.

UML DIAGRAM


Building<T>
position int
height int
type String
focusType String
length int
Office<T>
jobType String
length int
type String
height int
owner String
position int
Office(int, int, int, String, String)
Office()
checkInput(int, int, int) void
equals(Object) boolean
toString() String
position int
type String
focusType String
jobType String
height int
owner String
length int
Playground<T>
position int
length int
type String
height int
Playground(int, int, int)
Playground()
toString() String
equals(Object) boolean
checkInput(int, int, int) void
position int
height int
type String
focusType String
length int
House<T>
owner String
type String
length int
position int
height int
numberOfRooms int
color String
House(int, int, int, int, String, String)
House()
toString() String
equals(Object) boolean
checkInput(int, int, int) void
position int
type String
focusType String
color String
height int
owner String
numberOfRooms int
length int
Market<T>
closingTime int
type String
length int
height int
position int
owner String
openingTime int
Market(int, int, int, int, int, String)
Market()
toString() String
checkInput(int, int, int) void
equals(Object) boolean
position int
openingTime int
type String
focusType String
height int
owner String
closingTime int
length int
LDLinkedList
LDLinkedList()
getSkyline(Integer, Integer, Integer[][]) List<Integer[]>
remove(Object) boolean
add(int, Object) void
peek() Node
peekFirst() Node
clear() void
GetNode(Object) Node
add(Object) boolean
size() int
drawSkyline(StreetLinkedList) void
indexOf(Object) int
get(int) Object
addAll(Collection) boolean
peekLast() Node
getSkyline(Integer[][]) List<Integer[]>
greater(int, int) int
mergeSkylines(List<Integer[]>, List<Integer[]>) List<Integer[]>
drawBuildings(StreetLinkedList) void
printList() void
last Object
first Object
Node
Node(Object)
equals(Object) boolean
hashCode() int
StreetLinkedList
length int
size int
buildingArray LinkedList<Building>
StreetLinkedList(int, LinkedList<Building>, int)
StreetLinkedList()
calculateTotalNumberofRemainingLength() void
addMarket(int, int, int, int, int, String) void
calculatetotalNumberofNonPlaygrounds() void
greater(int, int) int
addPlayground(int, int, int) void
getTheSkyline(Integer, Integer, Integer[][]) List<Integer[]>
getTheSkyline(Integer[][]) List<Integer[]>
append(Integer[], int) Integer[]
deleteHouse(int) void
deleteOffice(int) void
inputCheckNumber(int) int
mergetheSkylines(List<Integer[]>, List<Integer[]>) List<Integer[]>
calculateNumberandRatioofPlaygrounds() void
findEmptyAreas() void
addOffice(int, int, int, String, String) void
deletePlayground(int) void
drawSkyline(StreetArrayList) void
addHouse(int, int, int, int, String, String) void
showBuildings() void
deleteMarket(int) void
buildingArray LinkedList<Building>
size int
length int
StreetArrayList
length int
buildingArray ArrayList<Building>
size int
StreetArrayList()
StreetArrayList(int, ArrayList<Building>, int)
addMarket(int, int, int, int, int, String) void
deleteOffice(int) void
mergetheSkylines(List<Integer[]>, List<Integer[]>) List<Integer[]>
deleteHouse(int) void
calculateNumberandRatioofPlaygrounds() void
calculateTotalNumberofRemainingLength() void
findEmptyAreas() void
deletePlayground(int) void
greater(int, int) int
drawSkyline(StreetArrayList) void
addPlayground(int, int, int) void
deleteMarket(int) void
showBuildings() void
calculatetotalNumberofNonPlaygrounds() void
inputCheckNumber(int) int
getTheSkyline(Integer[][]) List<Integer[]>
addOffice(int, int, int, String, String) void
getTheSkyline(Integer, Integer, Integer[][]) List<Integer[]>
append(Integer[], int) Integer[]
addHouse(int, int, int, int, String, String) void
size int
buildingArray ArrayList<Building>
length int
main