System Requirements

In this homework, we are asked to make 3 different Street class that using Arraylist, LinkedList and LDLinkedList that we implemented. All of these classes process adding and deleting Buildings(House, Office, Market, Playground) with different data structure. There are some advantegous over each other classes like adding and deleting on LinkedList is fast. After implement all classes, we made time complexity calculations and running time tests. We have 4 building type which has some common and not common properties. For common properties such as position, length, height fields and functions like focus(), we agreed that these 4 building have to be subclass of a superclass so execute methods polymorphically.

Use Case and Class Diagrams

Problem Solution Approach

I created 3 different version of Street class from HW1, AL_Street for arraylist, LL_Street for linkedlist and LD_Street for LDLinkedList which is asked to implement ourselves. To do this, we need to implement LDListIterator and Node class. LDLinkedList class has two Node object, one is “head” and the other one is “tail”. It also keeps int for size. There are overrided Abstractlist and Iterable methods in LDLinkedList and LDListIterator. Iterator() and listIterator() methods returns LDListIterator object. So we can iterate list through that iterator. Building objects are stored in the “data” fields in Nodes. By using head and tail nodes, methods like adding at first and last element are process fast. In LD_Street and LL_Street classes printing silhouette made by basic arrays. AL_Street using Arraylist for printing. In adding building function, user enters type of the building. After creating building, user gives property inputs(position, length and uncommon variables) for that building. Deleting function takes 2 two parameter (position and side). Position doesn’t have to be beginning position of building. It can be any point of a building. I assumed that when we are looking silhouette, near side is right side, further side is left side. Classes has 2 Building lists, one for ‘left’ and other one for ‘right’. It stores Building objects that exist at that time. LD_LinkedList has lazy_deletion strategy. After removing an element, that element stored in field named “lazy_linked” LinkedList. Then at each adding method, lazy_linked is controlled. If new element is found in lazy_linked, it will be moved from lazy to LDLinkedList. Building classes have overridden toString, equals, clone and hashcode methods. They have some polymorphic methods like focus() that prints particular informations of all buildings in street.

```
//methods overridden from AbstractList interface
boolean add(E e)
E get(int index)
int size()
Iterator<E> iterator()
ListIterator<E> listIterator(int index)
ListIterator<E> listIterator()
E remove(int index)

//methods overridden from ListIterator interface
boolean hasNext();
E next();
hasPrevious();
previous();
nextIndex();
previousIndex();
remove();
set(E arg0);
add(E arg0);
```

Test Cases

Test cases are same for every Data Structure and same as HW1 test cases.

```
//Length of street is set as 100

AL_Street AL_street = new AL_Street(100);
LL_Street LL_street = new LL_Street(100);
LD_Street LD_street = new LD_Street(100);
Street street = new Street(100);
```

```
// parameters are (position,Length,height,owner,color,room_number)
Building LD_house1 = new House(0,8,4,"ersel","red",30);
LD_street.add_building(LD_house1,'l');
LD_street.view_mode(5);

//this playground will not be added into street
Building playground2 = new Playground(2,5);
LD_street.add_building(playground2,'l');

//house5 wont be added, because 130 is out of bound. Street length is 100
Building house5 = new House(130,10,5,"henry","yellow",60);
LD_street.add_building(house5, 'l');
LD_street.view_mode(5);

// -1 is not an valid position
Building house6 = new House(-1,10,5,"henry","yellow",60);
LD_street.add_building(house6, 'l');
LD_street.view_mode(5);
```

```
//valid building
Building LD_market1 = new Market(4,16,9,"celal","08.00","21.00");
LD_street.add_building(LD_market1,'r');
LD_street.view_mode(5);

//valid building
Building LD_office1 = new Office(10,10,12);
LD_street.add_building(LD_office1,'l');
LD_street.view_mode(5);

//valid building
Building LD_house2 = new House(26,13,7,"eren","blue",17);
LD_street.add_building(LD_house2,'r');
LD_street.view_mode(5);

//valid building
Building LD_playground1 = new Playground(35,30);
LD_street.add_building(LD_playground1,'l');
LD_street.view_mode(5);

//valid building
Building LD_market2 = new Market(55,20,10,"mike","09.00","22.00");
LD_street.add_building(LD_market2,'r');
LD_street.view_mode(5);
```

Deleting Test

```
// First parameter is position, second is side. 3 is left, 4 is right
LD_street.delete_building(5,3);
LD_street.delete_building(5,4);
LD_street.view_mode(5);

//adding new building after delete
Building LD_house3 = new House(2,9,5,"john","green",12);
LD_street.add_building(LD_house3,'l');
LD_street.view_mode(5);
```

Overriden clone test

```
//method tests
System.out.printf("\n>>>>CLONE TEST<<<<\n");
House LD_h3 = (House)LD_house3;
House LD_h4 = (House)LD_h3.clone();
System.out.printf("\n--->Before changing fields<---");
System.out.printf("\nHouse3|Owner -> %s , Room Number -> %d",LD_h3.get_owner(),LD_h3.get_room_number());
System.out.printf("\nHouse4|Owner -> %s , Room Number -> %d",LD_h4.get_owner(),LD_h4.get_room_number());
LD_h4.set_room_number(33);
System.out.printf("\n\n--->After changing room-number field<---");
System.out.printf("\nHouse3|Owner -> %s , Room Number -> %d",LD_h3.get_owner(),LD_h3.get_room_number());
System.out.printf("\nHouse4|Owner -> %s , Room Number -> %d",LD_h4.get_owner(),LD_h4.get_room_number());
```

Overriden equals test

```
System.out.printf("\n\n>>>>EQUALS TEST<<<<\n");
if(LD_h3.equals(LD_h4)==true){
    System.out.printf("H3 and H4 are equal");
}
else{
    System.out.printf("H3 and H4 is not equal");
}
System.out.printf("\nH3 hashcode : %d",LD_h3.hashCode());
System.out.printf("\nH4 hashcode : %d",LD_h4.hashCode());
```

Overriden toString test

```
System.out.printf("\n\n>>>> toString TEST <<<<");
System.out.printf("\n%s",LD_h3);
System.out.printf("\n%s",LD_market1);
System.out.printf("\n%s",LD_office1);
System.out.printf("\n%s",LD_playground1);
```

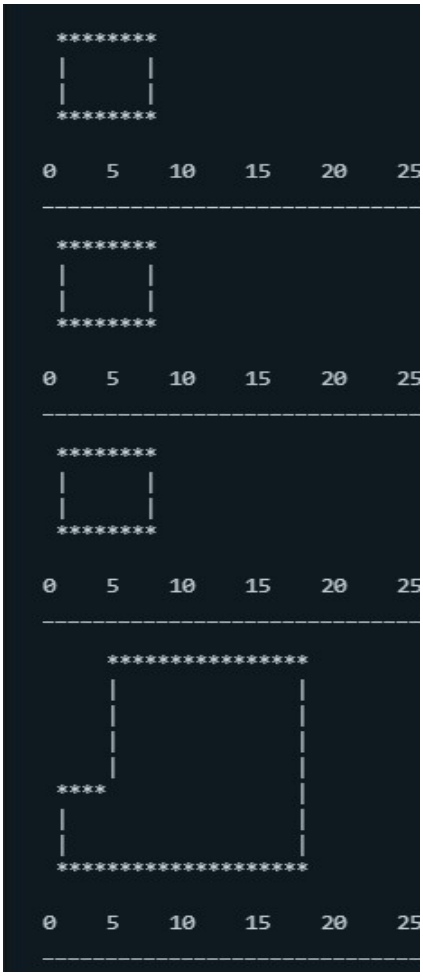
View modes

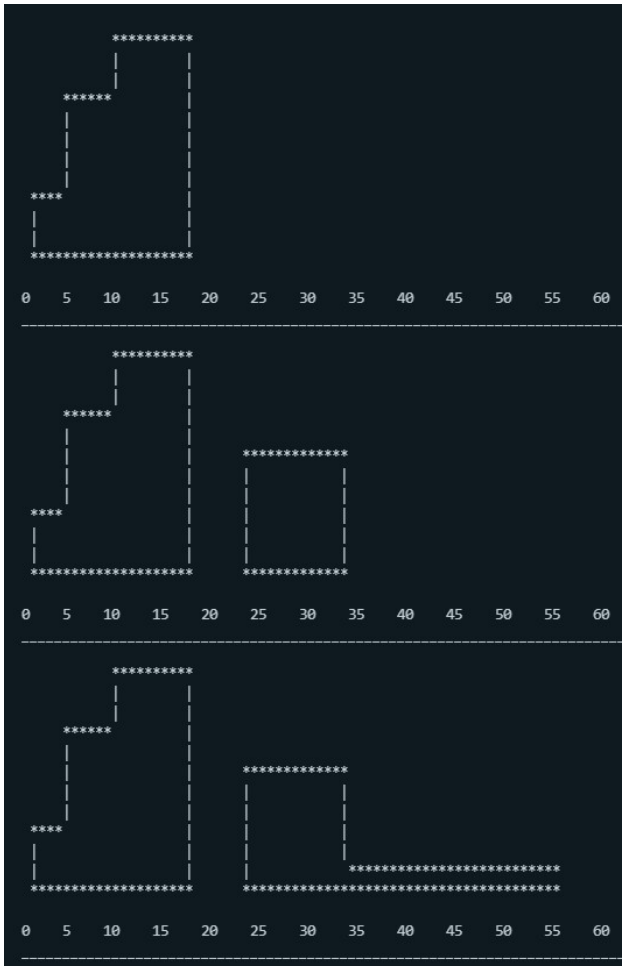
```
System.out.printf("\n");
LD_street.view_mode(1);
System.out.printf("\n");
LD_street.view_mode(2);
System.out.printf("\n");
LD_street.view_mode(3);
System.out.printf("\n");
LD_street.view_mode(4);
System.out.printf("\n");
LD_street.view_mode(6);
System.out.printf("\n");
System.out.printf("\n====-
```

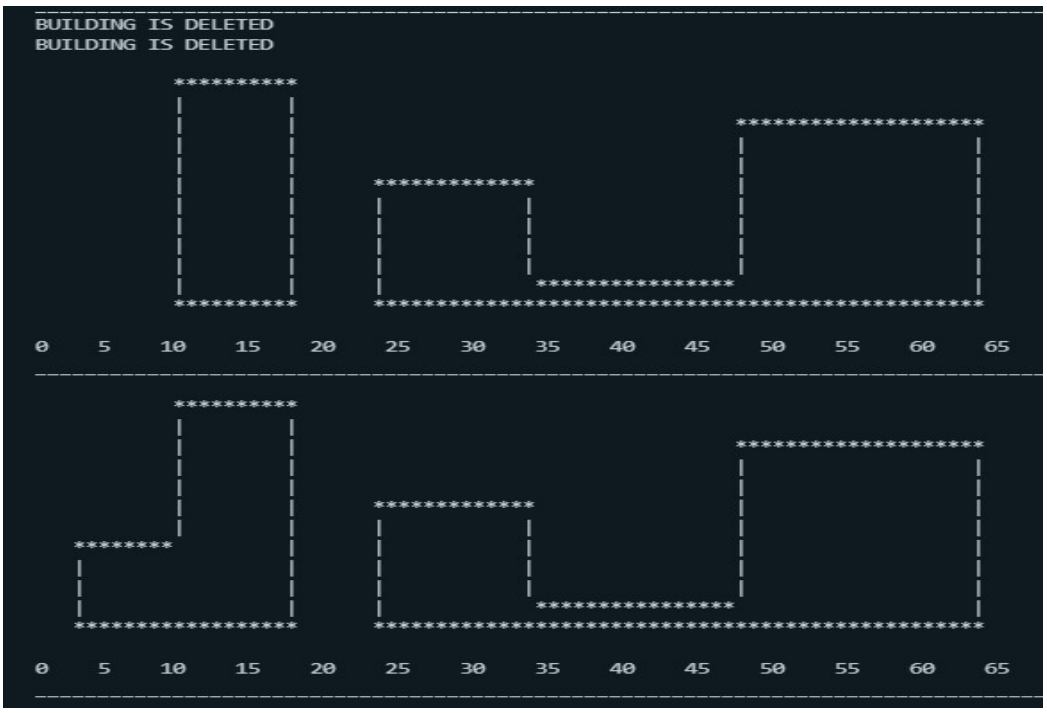
LDLinkedList Test, adding same elements to LinkedList as nodes

```
LDLinkedList<Building> LD_building_list = new LDLinkedList<>();
ListIterator<Building> iterator;

//creating nodes for list.
//first three parameters are common(position, length, height)
Building LD_house1 = new House(0,8,4,"ersel","red",30);
Building LD_market1 = new Market(4,16,9,"celal","08.00","21.00");
Building LD_office1 = new Office(10,10,12);
Building AL_house2 = new House(26,13,7,"eren","blue",17);
Building AL_playground1 = new Playground(35,30);
Building AL_house3 = new House(2,9,5,"john","green",12);

//adding Building nodes to LD_LinkedList
//testing add(), addFirst() and addLast()
LD_building_list.add(LD_house1);
LD_building_list.add(LD_market1);
LD_building_list.addFirst(LD_office1);
LD_building_list.addFirst(AL_house2);
LD_building_list.addLast(AL_playground1);
LD_building_list.addLast(AL_house3);
```

RUNNING AND RESULTS







```
>>>>CLONE TEST<<<<

--->Before changing fields<---
House3|Owner -> john , Room Number -> 12
House4|Owner -> john , Room Number -> 12

--->After changing room-number field<---
House3|Owner -> john , Room Number -> 12
House4|Owner -> john , Room Number -> 33

>>>>EQUALS TEST<<<<
H3 and H4 is not equal
H3 hashcode : 121500270
H4 hashcode : 121501299
--------------------------------
Market1 and Market2 is not equal
Market1 hashcode : 719202950
Market2 hashcode : 545979063
--------------------------------
Office1 and Office2 are equal
Market1 hashcode : 2016246578
Market2 hashcode : 2016246578
--------------------------------
```

```
>>>> toString TEST <<<<
|House| Position-> 2| Length-> 9| Height-> 5| Owner : john, Color : green, Number of rooms : 12
|Market| Position-> 4| Length-> 16| Height-> 9| Owner : celal, Opening Time : 08.00, Closing Time : 21.00
|Office| Position-> 10| Length-> 10| Height-> 12| Owner : unknown, Job-type : unknown
|Playground| Position-> 35| Length-> 30| Height-> 2
-------------------------------------
-------------------------------------
-------------------------------------

Remaining lands on the left side : 51
Remaining lands on the right side : 67
Total remaining lands on the street : 118

There are 2 House in street.
There are 1 Office in street.
There are 1 Market in street.
There are 1 Playground in street.

Total number of playground buildings : 1
Ratio of length of playgrounds : 15.000000

Total length of street occupied by Houses : 19
Total length of street occupied by Market : 30
Total length of street occupied by Office : 10

Job-type of this office is unknown
Length of this playground is 30
Owner of this house is john
Owner of this house is eren
Closing time of this market is 22.00

===-----------------------------------------END OF ARRAYLIST STREET-----------------------------------------===
```

LDLinkedList test results

```
======---Printing from head to tail---===
Position : 26
Position : 10
Position : 0
Position : 4
Position : 35
Position : 2

======---Printing from tail to head---===
Position : 2
Position : 35
Position : 4
Position : 0
Position : 10
Position : 26

======---Printing from head to tail after first element is deleted---===
Position : 10
Position : 0
Position : 4
Position : 35
Position : 2

_____Lazy Deleting____
Position : 10
Position : 26
Position : 0
Position : 4
Position : 35
Position : 2
----------------------------------------
```

```
======---Printing from head to tail after set() method---===
Position : 10
Position : 26
Position : 0
Position : 4
Position : 35
Position : 49

======---Printing from index 0 to size by using get() method---===
Position : 10
Position : 26
Position : 0
Position : 4
Position : 35
Position : 49

First item : 10
Last item : 49

======---Printing from index 0 to size after -> index=1 is removed---===
Position : 10
Position : 0
Position : 4
Position : 35
Position : 49
```

===-------TIME COMPLEXITY------===

HW1 STREET TIME COMPLEXITY

remaining_lands() → θ(n)
list_of_buildings() → θ(n)
playground_ratio() → O(n)
occupied_length() → O(n)
sillhouette() → O(n*m*k) n=number of building, m=length, k=height
mark_sillhouette() → θ(n*m) n=length of street , m=height of street
print_sillhouette() → θ(n*m) n=length of street , m=height of street
edit_sillhouette() → θ(n*m) n=length of street , m=height of street
fill_sillhoutte() → O(n*m*k) n=number of building, m=length, k=height
find_max_height() → O(n) both left and right might be null
add_building() → θ(n) because find_space() is θ(n)
add_building(Building obj,char side) → θ(n) because find_space() is θ(n)
add_process(Building newBuilding, Building[] street) → θ(1) amortized
find_space(Building[] street, Building build) → θ(n)
set_length_of_street(int l) → θ(1)
get_delete_input() → θ(n) because delete_building() is θ(n)
delete_building(int chosen_position, int choice) → θ(n)
delete_process(Building[] street) → O(n)
street_focus() → θ(n)

AL_STREET TIME COMPLEXITY

remaining_lands() → θ(n)
list_of_buildings() → θ(n)
playground_ratio() → O(n)
occupied_length() → O(n)
sillhouette() → O(n*m*k) n=number of building, m=length, k=height
mark_sillhouette() → θ(n*m) n=length of street , m=height of street
print_sillhouette() → θ(n*m) n=length of street , m=height of street
edit_sillhouette() → θ(n*m) n=length of street , m=height of street
fill_sillhoutte() → O(n*m*k) n=number of building, m=length, k=height
find_max_height() → O(n) both left and right might be null
add_building() → θ(n) because find_space() is θ(n)
add_building(Building obj,char side) → θ(n) because find_space() is θ(n)
add_process(Building newBuilding, ArrayList<Building> street) → θ(1) amortized
find_space(ArrayList<Building> street, Building build) → θ(n)
set_length_of_street(int l) → θ(1)
get_delete_input() → θ(n) because delete_building() is θ(n)
delete_building(int chosen_position, int choice) → θ(n)
delete_process(ArrayList<Building> street) → O(n)
street_focus() → θ(n)

LL_STREET TIME COMPLEXITY

remaining_lands() → θ(n)
list_of_buildings() → θ(n)
playground_ratio() → O(n)
occupied_length() → O(n)
sillhouette() → O(n*m*k) n=number of building, m=length, k=height
mark_sillhouette() → θ(n*m) n=length of street , m=height of street
print_sillhouette() → θ(n*m) n=length of street , m=height of street
edit_sillhouette() → θ(n*m) n=length of street , m=height of street
fill_sillhoutte() → O(n*m*k) n=number of building, m=length, k=height
find_max_height() → O(n) both left and right might be null
add_building() → θ(n) because find_space() is θ(n)
add_building(Building obj,char side) → θ(n) because find_space() is θ(n)
add_process(Building newBuilding, LinkedList<Building> street) → θ(1)
find_space(LinkedList<Building> street, Building build) → θ(n)
set_length_of_street(int l) → θ(1)
get_delete_input() → θ(n) because delete_building() is θ(n)
delete_building(int chosen_position, int choice) → θ(n)
delete_process(LinkedList<Building> street) → O(n)
street_focus() → θ(1)

LD_STREET TIME COMPLEXITY

remaining_lands() → θ(n)
list_of_buildings() → θ(n)
playground_ratio() → O(n)
occupied_length() → O(n)
sillhouette() → O(n*m*k) n=number of building, m=length, k=height
mark_sillhouette() → θ(n*m) n=length of street , m=height of street
print_sillhouette() → θ(n*m) n=length of street , m=height of street
edit_sillhouette() → θ(n*m) n=length of street , m=height of street
fill_sillhoutte() → O(n*m*k) n=number of building, m=length, k=height
find_max_height() → O(n) both left and right might be null
add_building() → θ(n) because find_space() is θ(n)
add_building(Building obj,char side) → θ(n) because find_space() is θ(n)
add_process(Building newBuilding, LDLinkedList<Building> street) → θ(1)
find_space(LDLinkedList<Building> street, Building build) → θ(n)
set_length_of_street(int l) → θ(1)
get_delete_input() → θ(n) because delete_building() is θ(n)
delete_building(int chosen_position, int choice) → θ(n)
delete_process(LDLinkedList<Building> street) → O(n)
street_focus() → θ(1)

LDLINKEDLIST TIME COMPLEXITY

```
check_lazy(E e)
add(int index, E obj)
boolean add(E e)
addFirst(E obj) → θ(1)
addLast(E obj) → θ(1)
getFirst() → θ(1)
getLast() → θ(1)
E seperateNode(Node<E> n)
boolean remove(Object o) → O(n)
E get(int index) → θ(1)
int size() → θ(1)
E remove(int index) → O(1)
```

LDLISTITERATOR TIME COMPLEXITY

```
LDListIterator(int i)
void add(E e) → O(n) because searching in lazy_linked
boolean hasNext() → θ(1)
boolean hasPrevious() → θ(1)
E next() → θ(1)
int nextIndex() → θ(1)
E previous() → θ(1)
int previousIndex() → θ(1)
void remove() → θ(1)
void set(E e) → θ(1)
```

Running time testing of adding buildings

```
//starting timer
long start = System.nanoTime();
for(int i=0;i< number_of_buildings ;i++){
    office = new Office(p,10,15);
    AL_street.add_building(office,'l');
    p = p+11;
}
p=0;
for(int i=0;i< number_of_buildings ;i++){
    office = new Office(p+2,10,p+3);
    AL_street.add_building(office,'r');
    p = p+10;
}
//stopping timer
long end = System.nanoTime();
long elapsedTime = end - start;
System.out.printf("\nN : %d  |  Time : %d\n\n",number_of_buildings,elapsedTime/10000);
```

```
N : 100  |  Time : 1192
N : 100  |  Time : 1243
N : 100  |  Time : 1189
N : 100  |  Time : 1176
N : 100  |  Time : 1234
N : 100  |  Time : 1281
```

```
N : 200  |  Time : 2218
N : 200  |  Time : 2112
N : 200  |  Time : 2141
N : 200  |  Time : 2090
N : 200  |  Time : 2122
```

```
N : 300  |  Time : 2892
N : 300  |  Time : 2654
N : 300  |  Time : 2842
N : 300  |  Time : 2778
N : 300  |  Time : 2760
N : 300  |  Time : 2884
```

```
N : 400  |  Time : 3224
N : 400  |  Time : 3585
N : 400  |  Time : 3365
N : 400  |  Time : 3524
N : 400  |  Time : 3236
N : 400  |  Time : 3427
```

```java
//starting timer
long start = System.nanoTime();
AL_street.view_mode(5);
//stopping timer
long end = System.nanoTime();
long elapsedTime = end - start;
System.out.printf("N : %d  |  Time : %d ",
        number_of_buildings,elapsedTime/10000);
```

```
N : 100  |  Time : 8934
N : 100  |  Time : 8892
N : 200  |  Time : 24970
N : 200  |  Time : 27899
N : 300  |  Time : 42972
N : 300  |  Time : 43674
```