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

Sefa Çiçek
1801042657

1. **System Requirements**

- User has to choose mode on menu (user or driver mode). If user choose ‘user mode’, street length must be determined by the user to execute other parts.

- On the second section, user has to again choose mode such as editing mode, viewing mode or focus a building. On editing mode, user must select ‘add a building’ or ‘delete a building’ part. In these two selections, building information is requested. If user wants to see actual street display and some calculations about the street, he/she has to add some buildings to the street, otherwise he/she see empty street info.

- There must be empty spaces for the 2 sides of the street. frontStreet and backStreet array’s size are equal to the street length.

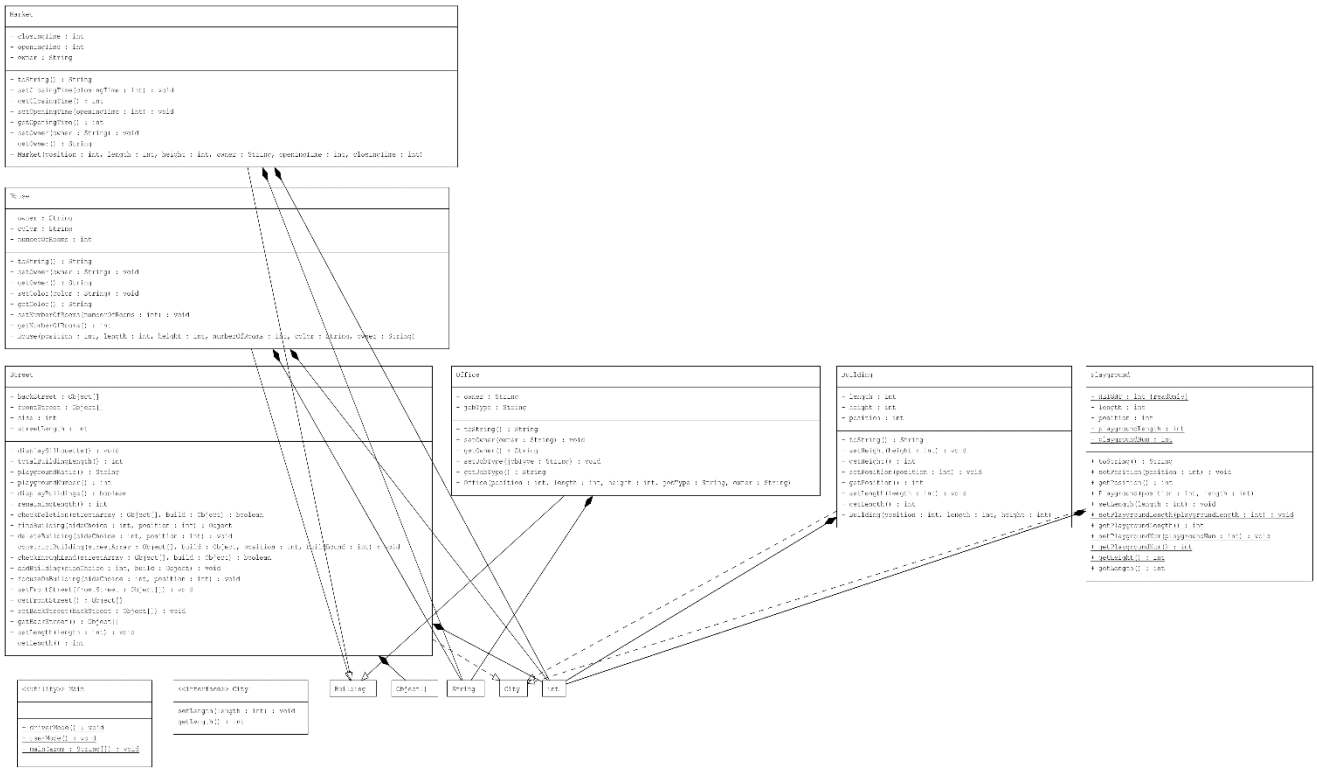
- jdk and jre are requested from operating system to execute this java program.

-User has to run makefiles for 3 separate folder.

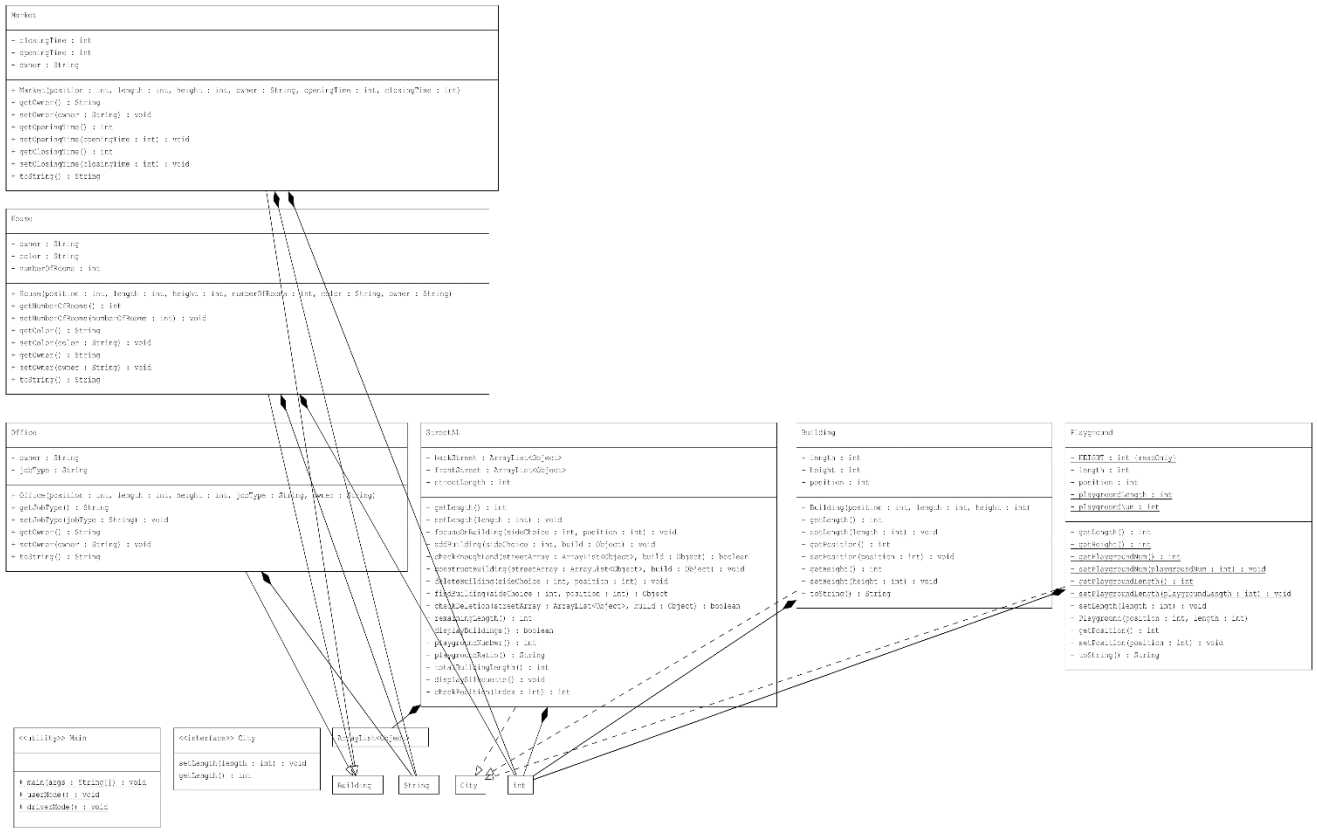
2. Class Diagrams

City is an interface. I didn’t need to implement City as a class.
Required operations (add/delete building) are generally stored in Street class.
House, Office and Market extend Building class.
Playground is different from building but they have similarity.

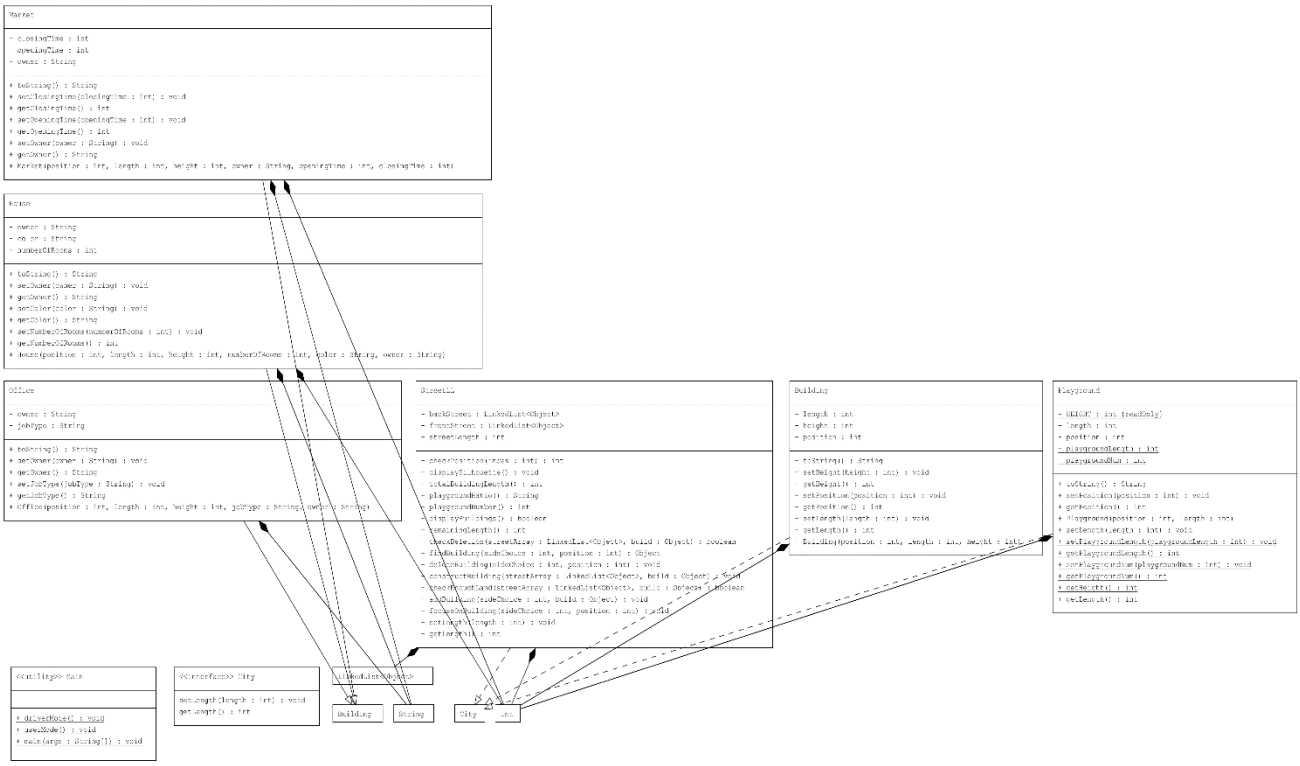
Basic Array



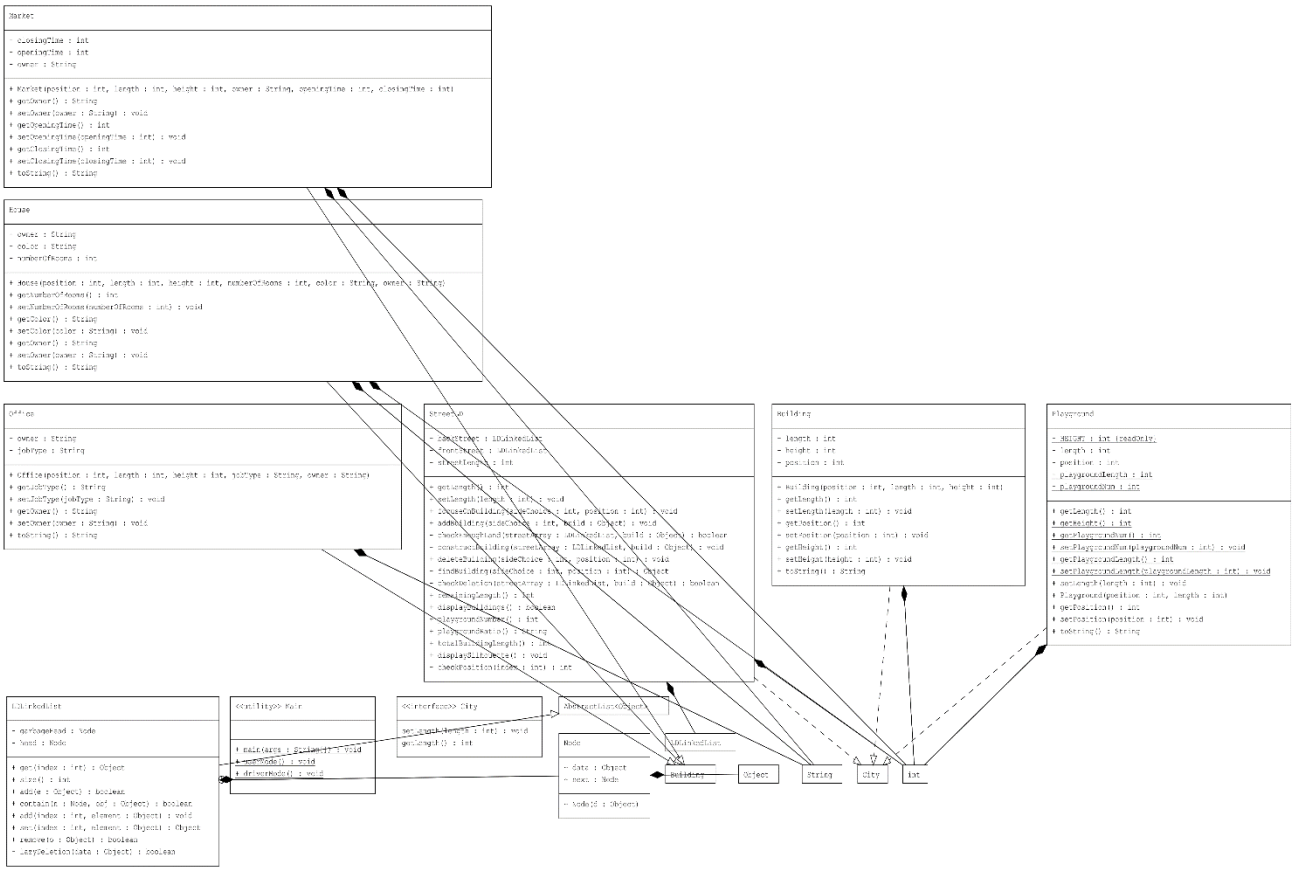
Array List



Linked List



LDLinked List



3. Problem Solution Approach

Firstly, I wrote StreetAL class(for Array List) which has 2 arrays to hold the sides of the street and then I identified the differences between Street class(for Basic Array) and StreetAL. The main idea is same but code was changed a little bit for translation.
Secondly, I wrote StreetLL class(for Linked List) and LD LinkedList class with some basic modifications.
In LD LinkedList class, I overrode some AbstractList methods such as add and remove. Additionally, I wrote 2 extra function –'contain' and 'lazyDeletion'-. These are used for lazy deletion part. In 'lazyDeletion' function, I add deleted item to the second list for later use. Also, I checked existence of node on the second list in 'contain' function.

4. Test Cases

- Setting street length with negative and correct input
- Empty street actions:
    - Delete building from empty street
    - Display total remaining length of empty street
    - Display buildings on empty street
    - Show number and ratio of playgrounds
    - Show total length of street occupied by the markets, houses or offices
    - Focusing on a building and playground on empty street
    - Display the skyline silhouette of empty street

- Adding buildings and playground to the street
- Adding building or playground to the filled area of street
- Adding building or playground to the outside of street
- Deleting building from empty position on the street
- Deleting building from outside of the street
- Deleting building from the street (correct input)
- Displaying the total remaining length of lands on the street
- Displaying the list of buildings on the street
- Display the number and ratio of length of playgrounds
- Display the total length of street occupied by the markets, houses or offices
- Focusing on a building or playground
- Displaying the skyline silhouette of the street

5. Running Command and Results

```
sefa@DESKTOP-O4JURCM:/mnt/c/Users/seffa/Desktop/VsJava$ javac Main.java
sefa@DESKTOP-O4JURCM:/mnt/c/Users/seffa/Desktop/VsJava$ java Main
Welcome!
1- User Mode
2- Driver Mode
0- Exit

Choose the mode:
2
Setting street length -> -20
java.lang.NegativeArraySizeException: -20
        at Street.setLength(Street.java:24)
        at Main.driverMode(Main.java:272)
        at Main.main(Main.java:34)
----------------------------
----------------------------
Setting street length -> 100
----------------------------
----------------------------


----------------------------
EMPTY STREET
----------------------------
Deleting from empty street
Side choice = 1 (front)
position = 5
java.lang.NullPointerException
        at Street.deleteBuilding(Street.java:212)
        at Main.driverMode(Main.java:294)
        at Main.main(Main.java:34)
----------------------------
----------------------------
Displaying the total remaining length of lands on the street

--Total remaining length: 200

----------------------------
----------------------------
Displaying the list of buildings on the street

Front of street:
--This side is empty.

Back of street:
--This side is empty.

--There is no building for display!

----------------------------
----------------------------
Display the number and ratio of length of playgrounds

--Number: 0

--Ratio: 0/200

----------------------------
----------------------------
Display the total length of street occupied by the markets, houses or offices

--Total length of buildings: 0

****************************
----------------------------
Focusing on a building

Front of street:
--This side is empty.

Back of street:
--This side is empty.

--There is no building for focusing!

----------------------------
----------------------------
Focusing on a playground
```

```
Front of street:
--This side is empty.

Back of street:
--This side is empty.

--There is no playground for focusing!

----------------------------
----------------------------
Displaying the skyline silhouette of the street

 ____________________________________________________________________________________________________
|____________________________________________________________________________________________________|

----------------------------
END OF THE EMPTY STREET PART
----------------------------


Adding buildings to the front of street

House [position=0, length=7, height=14, numberOfRooms=3, color=ocean blue, owner=sefa]
--Building has added!

Front of street:
House [position=0, length=7, height=14, numberOfRooms=3, color=ocean blue, owner=sefa]

Back of street:
--This side is empty.


Office [position=10, length=12, height=14, openingTime=8, closingTime=22, owner=bim]
--Building has added!

Front of street:
House [position=0, length=7, height=14, numberOfRooms=3, color=ocean blue, owner=sefa]
Office [position=10, length=12, height=14, openingTime=8, closingTime=22, owner=bim]

Back of street:
--This side is empty.

Office [position=48, length=8, height=9, jobType=detective, owner=faruk]
--Building has added!

Front of street:
House [position=0, length=7, height=14, numberOfRooms=3, color=ocean blue, owner=sefa]
Office [position=10, length=12, height=14, openingTime=8, closingTime=22, owner=bim]
Office [position=48, length=8, height=9, jobType=detective, owner=faruk]

Back of street:
--This side is empty.


Office [position=58, length=8, height=9, openingTime=9, closingTime=21, owner=a101]
--Building has added!

Front of street:
House [position=0, length=7, height=14, numberOfRooms=3, color=ocean blue, owner=sefa]
Office [position=10, length=12, height=14, openingTime=8, closingTime=22, owner=bim]
Office [position=48, length=8, height=9, jobType=detective, owner=faruk]
Office [position=58, length=8, height=9, openingTime=9, closingTime=21, owner=a101]

Back of street:
--This side is empty.

Adding playground to the front of street

Playground [position=0, length=4, height=2]
--Building cannot be added to this position.

Front of street:
House [position=0, length=7, height=14, numberOfRooms=3, color=ocean blue, owner=sefa]
Office [position=10, length=12, height=14, openingTime=8, closingTime=22, owner=bim]
Office [position=48, length=8, height=9, jobType=detective, owner=faruk]
Office [position=58, length=8, height=9, openingTime=9, closingTime=21, owner=a101]

Back of street:
--This side is empty.
```

```
Adding buildings to the back of street

Office [position=5, length=7, height=4, jobType=lawyer, owner=robby]
--Building has added!

Front of street:
House [position=0, length=7, height=14, numberOfRooms=3, color=ocean blue, owner=sefa]
Office [position=10, length=12, height=14, openingTime=8, closingTime=22, owner=bim]
Office [position=48, length=8, height=9, jobType=detective, owner=faruk]
Office [position=58, length=8, height=9, openingTime=9, closingTime=21, owner=a101]

Back of street:
Office [position=5, length=7, height=4, jobType=lawyer, owner=robby]


House [position=19, length=11, height=25, numberOfRooms=4, color=red, owner=cicek]
--Building has added!

Front of street:
House [position=0, length=7, height=14, numberOfRooms=3, color=ocean blue, owner=sefa]
Office [position=10, length=12, height=14, openingTime=8, closingTime=22, owner=bim]
Office [position=48, length=8, height=9, jobType=detective, owner=faruk]
Office [position=58, length=8, height=9, openingTime=9, closingTime=21, owner=a101]

Back of street:
Office [position=5, length=7, height=4, jobType=lawyer, owner=robby]
House [position=19, length=11, height=25, numberOfRooms=4, color=red, owner=cicek]


House [position=53, length=10, height=14, numberOfRooms=5, color=orange, owner=muzaffer]
--Building has added!

Front of street:
House [position=0, length=7, height=14, numberOfRooms=3, color=ocean blue, owner=sefa]
Office [position=10, length=12, height=14, openingTime=8, closingTime=22, owner=bim]
Office [position=48, length=8, height=9, jobType=detective, owner=faruk]
Office [position=58, length=8, height=9, openingTime=9, closingTime=21, owner=a101]

Back of street:
Office [position=5, length=7, height=4, jobType=lawyer, owner=robby]
House [position=19, length=11, height=25, numberOfRooms=4, color=red, owner=cicek]
House [position=53, length=10, height=14, numberOfRooms=5, color=orange, owner=muzaffer]



House [position=82, length=14, height=10, numberOfRooms=3, color=lime, owner=robo]
--Building has added!

Front of street:
House [position=0, length=7, height=14, numberOfRooms=3, color=ocean blue, owner=sefa]
Office [position=10, length=12, height=14, openingTime=8, closingTime=22, owner=bim]
Office [position=48, length=8, height=9, jobType=detective, owner=faruk]
Office [position=58, length=8, height=9, openingTime=9, closingTime=21, owner=a101]

Back of street:
Office [position=5, length=7, height=4, jobType=lawyer, owner=robby]
House [position=19, length=11, height=25, numberOfRooms=4, color=red, owner=cicek]
House [position=53, length=10, height=14, numberOfRooms=5, color=orange, owner=muzaffer]
House [position=82, length=14, height=10, numberOfRooms=3, color=lime, owner=robo]

Adding playground to the back of street

Playground [position=35, length=4, height=2]
--Building has added!

Front of street:
House [position=0, length=7, height=14, numberOfRooms=3, color=ocean blue, owner=sefa]
Office [position=10, length=12, height=14, openingTime=8, closingTime=22, owner=bim]
Office [position=48, length=8, height=9, jobType=detective, owner=faruk]
Office [position=58, length=8, height=9, openingTime=9, closingTime=21, owner=a101]

Back of street:
Office [position=5, length=7, height=4, jobType=lawyer, owner=robby]
House [position=19, length=11, height=25, numberOfRooms=4, color=red, owner=cicek]
Playground [position=35, length=4, height=2]
House [position=53, length=10, height=14, numberOfRooms=5, color=orange, owner=muzaffer]
House [position=82, length=14, height=10, numberOfRooms=3, color=lime, owner=robo]

----------------------------
----------------------------
Adding building or playground to the filled area of street

Office [position=3, length=12, height=18, jobType=dentist, owner=necmi]
--Building cannot be added to this position.
```

```
Front of street:
House [position=0, length=7, height=14, numberOfRooms=3, color=ocean blue, owner=sefa]
Office [position=10, length=12, height=14, openingTime=8, closingTime=22, owner=bim]
Office [position=48, length=8, height=9, jobType=detective, owner=faruk]
Office [position=58, length=8, height=9, openingTime=9, closingTime=21, owner=a101]

Back of street:
Office [position=5, length=7, height=4, jobType=lawyer, owner=robby]
House [position=19, length=11, height=25, numberOfRooms=4, color=red, owner=cicek]
Playground [position=35, length=4, height=2]
House [position=53, length=10, height=14, numberOfRooms=5, color=orange, owner=muzaffer]
House [position=82, length=14, height=10, numberOfRooms=3, color=lime, owner=robo]


Playground [position=55, length=9, height=2]
--Building cannot be added to this position.

Front of street:
House [position=0, length=7, height=14, numberOfRooms=3, color=ocean blue, owner=sefa]
Office [position=10, length=12, height=14, openingTime=8, closingTime=22, owner=bim]
Office [position=48, length=8, height=9, jobType=detective, owner=faruk]
Office [position=58, length=8, height=9, openingTime=9, closingTime=21, owner=a101]

Back of street:
Office [position=5, length=7, height=4, jobType=lawyer, owner=robby]
House [position=19, length=11, height=25, numberOfRooms=4, color=red, owner=cicek]
Playground [position=35, length=4, height=2]
House [position=53, length=10, height=14, numberOfRooms=5, color=orange, owner=muzaffer]
House [position=82, length=14, height=10, numberOfRooms=3, color=lime, owner=robo]

----------------------------
----------------------------
Adding building or playground to the outside of street
Office [position=-10, length=5, height=14, openingTime=7, closingTime=22, owner=bakkal mito]
Playground [position=135, length=20, height=2]
--Invalid build property!
java.lang.ArrayIndexOutOfBoundsException
        at Street.checkEnoughLand(Street.java:148)
        at Street.addBuilding(Street.java:100)
        at Main.driverMode(Main.java:396)
        at Main.main(Main.java:34)
----------------------------
----------------------------
Deleting from empty position of the street
position = 25
java.lang.NullPointerException
        at Street.deleteBuilding(Street.java:212)
        at Main.driverMode(Main.java:410)
        at Main.main(Main.java:34)
----------------------------
----------------------------
Deleting from outside of the street
position = 150 (street length = 100)
java.lang.ArrayIndexOutOfBoundsException: Index 150 out of bounds for length 100
        at Street.findBuilding(Street.java:260)
        at Street.deleteBuilding(Street.java:205)
        at Main.driverMode(Main.java:421)
        at Main.main(Main.java:34)
----------------------------
----------------------------
Deleting from the street (correct input - position = 82)

House [position=82, length=14, height=10, numberOfRooms=3, color=lime, owner=robo]
--Building has deleted.

Front of street:
House [position=0, length=7, height=14, numberOfRooms=3, color=ocean blue, owner=sefa]
Office [position=10, length=12, height=14, openingTime=8, closingTime=22, owner=bim]
Office [position=48, length=8, height=9, jobType=detective, owner=faruk]
Office [position=58, length=8, height=9, openingTime=9, closingTime=21, owner=a101]

Back of street:
Office [position=5, length=7, height=4, jobType=lawyer, owner=robby]
House [position=19, length=11, height=25, numberOfRooms=4, color=red, owner=cicek]
Playground [position=35, length=4, height=2]
House [position=53, length=10, height=14, numberOfRooms=5, color=orange, owner=muzaffer]

----------------------------
----------------------------
Displaying the total remaining length of lands on the street

--Total remaining length: 133

----------------------------
----------------------------
```

```
Displaying the list of buildings on the street

Front of street:
House [position=0, length=7, height=14, numberOfRooms=3, color=ocean blue, owner=sefa]
Office [position=10, length=12, height=14, openingTime=8, closingTime=22, owner=bim]
Office [position=48, length=8, height=9, jobType=detective, owner=faruk]
Office [position=58, length=8, height=9, openingTime=9, closingTime=21, owner=a101]

Back of street:
Office [position=5, length=7, height=4, jobType=lawyer, owner=robby]
House [position=19, length=11, height=25, numberOfRooms=4, color=red, owner=cicek]
Playground [position=35, length=4, height=2]
House [position=53, length=10, height=14, numberOfRooms=5, color=orange, owner=muzaffer]

----------------------------
----------------------------
Display the number and ratio of length of playgrounds

--Number: 1

--Ratio: 4/200

----------------------------
----------------------------
Display the total length of street occupied by the markets, houses or offices

--Total length of buildings: 63

----------------------------
----------------------------
Focusing on a building

Front of street:
House [position=0, length=7, height=14, numberOfRooms=3, color=ocean blue, owner=sefa]
Office [position=10, length=12, height=14, openingTime=8, closingTime=22, owner=bim]
Office [position=48, length=8, height=9, jobType=detective, owner=faruk]
Office [position=58, length=8, height=9, openingTime=9, closingTime=21, owner=a101]
```

```
Back of street:
Office [position=5, length=7, height=4, jobType=lawyer, owner=robby]
House [position=19, length=11, height=25, numberOfRooms=4, color=red, owner=cicek]
Playground [position=35, length=4, height=2]
House [position=53, length=10, height=14, numberOfRooms=5, color=orange, owner=muzaffer]

Side of the street (1 -- front)
Position of building to focus (3)
--House owner: sefa

----------------------------
----------------------------
Focusing on a playground

Front of street:
House [position=0, length=7, height=14, numberOfRooms=3, color=ocean blue, owner=sefa]
Office [position=10, length=12, height=14, openingTime=8, closingTime=22, owner=bim]
Office [position=48, length=8, height=9, jobType=detective, owner=faruk]
Office [position=58, length=8, height=9, openingTime=9, closingTime=21, owner=a101]

Back of street:
Office [position=5, length=7, height=4, jobType=lawyer, owner=robby]
House [position=19, length=11, height=25, numberOfRooms=4, color=red, owner=cicek]
Playground [position=35, length=4, height=2]
House [position=53, length=10, height=14, numberOfRooms=5, color=orange, owner=muzaffer]

Side of the street (2 -- back)
Position of playground to focus (35)
--Playground length: 4

----------------------------
----------------------------
```


Displaying the skyline silhouette of the street

Time Complexity of Four Versions

```java
@Override
public int getLength() {
    return streetLength;
}
```

Basic Array : Θ(1)
Array List : Θ(1)
Linked List : Θ(1)
LD Linked List : Θ(1)

```java
@Override
public void setLength(int length) {
    if (length <= 0)
        throw new NegativeArraySizeException();
    this.streetLength = length;
}
```

Basic Array : Θ(1)
Array List : Θ(1)
Linked List : Θ(1)
LD Linked List : Θ(1)

```java
public void focuseOnBuilding(int sideChoice, int position) {

    int buildPos = 0;
    int buildLength = 0;
    int buildBound = 0;

    Object build = new Object();

    if (sideChoice == 1) { // frontStreet
        for (int i = 0; i < frontStreet.size(); i++) {
            buildPos = (frontStreet.get(i) instanceof Playground)
                    ? ((Playground) frontStreet.get(i)).getPosition()
                    : ((Building) frontStreet.get(i)).getPosition();
            buildLength = (frontStreet.get(i) instanceof Playground)
                    ? ((Playground) frontStreet.get(i)).getLength()
                    : ((Building) frontStreet.get(i)).getLength();

            buildBound = buildPos + buildLength;
            if (position < buildBound && position >= buildPos) {
                build = frontStreet.get(i);
                break;
            }
        }
    } else if (sideChoice == 2) { // backStreet
          for (int i = 0; i < backStreet.size(); i++) {
            buildPos = (backStreet.get(i) instanceof Playground)
                    ? ((Playground) backStreet.get(i)).getPosition()
                    : ((Building) backStreet.get(i)).getPosition();
            buildLength = (backStreet.get(i) instanceof Playground)
                    ? ((Playground) backStreet.get(i)).getLength()
                    : ((Building) backStreet.get(i)).getLength();

            buildBound = buildPos + buildLength;
            if (position < buildBound && position >= buildPos) {
                build = backStreet.get(i);
                break;
            }
          }
    }

    System.out.println();

    if (build instanceof House)
        System.out.println("--House owner: " + ((House) build).getOwner());

    else if (build instanceof Office)
        System.out.println("--Job type: " + ((Office) build).getJobType());

    else if (build instanceof Playground)
        System.out.println("--Playground length: " + ((Playground) build).getLength());

    else if (build instanceof Market)
        System.out.println("--Market closing time: " + ((Market) build).getClosingTime());

    else
        System.out.println("--No building at this position.");
    System.out.println();

}
```

Basic Array : Θ(1) – It has reference (frontStreet[position])
Array List : O(n) - get(i) time comp.(O(1)) * for loop = O(n)
Linked List : O(n^2) – get(i) time comp.(O(n)) * for loop = O(n^2)
LD Linked List : O(n^2) - get(i) time comp.(O(n)) * for loop = O(n^2)

```java
public void addBuilding(int sideChoice, Object build) {

    switch (sideChoice) {
        case 1:
            if (checkEnoughLand(frontStreet, build))
                System.out.println("--Building has added!");
            else
                System.out.println("--Building cannot be added to this position.");
            break;

        case 2:
            if (checkEnoughLand(backStreet, build))
                System.out.println("--Building has added!");
            else
                System.out.println("--Building cannot be added to this position.");
            break;

        default:
            System.out.println("--Invalid choice!");
            break;
    }

    displayBuildings();

}
```

Basic Array : O(n) – displayBuildings(O(n)) + checkEnoughLand(O(n)) = O(n)
Array List : O(n) – displayBuildings(O(n)) + checkEnoughLand(O(n)) = O(n)
Linked List : O(n^2) - displayBuildings(O(n^2)) + checkEnoughLand(O(n^2)) = O(n^2)
LD Linked List : O(n^2) - displayBuildings(O(n^2)) + checkEnoughLand(O(n^2)) = O(n^2)

```java
private boolean checkEnoughLand(LinkedList<Object> streetArray, Object build) {

    int buildPos = 0;
    int buildLength = 0;
    int buildBound = 0;

    buildPos = (build instanceof Playground)
            ? ((Playground) build).getPosition()
            : ((Building) build).getPosition();

    buildLength = (build instanceof Playground)
            ? ((Playground) build).getLength()
            : ((Building) build).getLength();

    if (buildPos >= streetLength || buildLength > streetLength || buildPos < 0 || buildLength <= 0) {
        System.out.println("--Invalid build property!");
        throw new IndexOutOfBoundsException();
    }

    if (streetArray.isEmpty()) {
        constructBuilding(streetArray, build);
        System.out.println("\n" + build);
        return true;
    }

    for (int i = 0; i < streetArray.size(); i++) {
        buildPos = (streetArray.get(i) instanceof Playground)
                ? ((Playground) streetArray.get(i)).getPosition()
                : ((Building) streetArray.get(i)).getPosition();
        buildLength = (streetArray.get(i) instanceof Playground)
                ? ((Playground) streetArray.get(i)).getLength()
                : ((Building) streetArray.get(i)).getLength();

        buildBound = buildPos + buildLength;
        if (build instanceof Building) {
            if (((Building) build).getPosition() < buildBound &&
                    ((Building) build).getPosition() >= buildPos) {
                System.out.println("\n" + build);
                return false;
            }
        } else if (build instanceof Playground) {
            if (((Playground) build).getPosition() < buildBound &&
                    ((Playground) build).getPosition() >= buildPos) {
                System.out.println("\n" + build);
                return false;
            }
        }
    }
    // Position is empty
    constructBuilding(streetArray, build);
    System.out.println("\n" + build);
    return true;

}
```

Basic Array : O(n) – for loop(O(n)) + constructBuilding(Θ(1)) = O(n)
Array List : O(n) – get(i) time comp.(O(1)) * for loop = O(n)
Linked List : O(n^2) - get(i) time comp.(O(n)) * for loop = O(n^2)
LD Linked List : O(n^2) - get(i) time comp.(O(n)) * for loop = O(n^2)

```java
private void constructBuilding(ArrayList<Object> streetArray, Object build) {

    streetArray.add(build); // Adding building to the street

    /* Changing total playground length and number */
    if (build instanceof Playground) {
        int oldLength = Playground.getPlaygroundLength();
        int buildLength = ((Playground) build).getLength();
        int oldNum = Playground.getPlaygroundNum();
        Playground.setPlaygroundLength(oldLength + buildLength);
        Playground.setPlaygroundNum(oldNum + 1);
    }

}
```

Basic Array : Θ(1) – It has reference (streetArray[position])
Array List : amortized O(1) - add
Linked List : O(1) - add
LD Linked List : O(1) - add

```java
public void deleteBuilding(int sideChoice, int position) {

    Object build = findBuilding(sideChoice, position);

    int buildPos = 0;
    int buildLength = 0;

    if (!(build instanceof Playground) && !(build instanceof Building)) {
        System.out.println("--There is no building.");
        return;
    }

    buildPos = (build instanceof Playground)
            ? ((Playground) build).getPosition()
            : ((Building) build).getPosition();

    buildLength = (build instanceof Playground)
            ? ((Playground) build).getLength()
            : ((Building) build).getLength();

    if (buildPos < 0 || buildPos >= streetLength || buildLength <= 0 || buildLength > streetLength) {
        System.out.println("--Invalid build property!");
        throw new IndexOutOfBoundsException();
    }

    switch (sideChoice) {
        case 1:
            if (checkDeletion(frontStreet, build)) {
                System.out.println("\n" + build);
                System.out.println("--Building has deleted.");
            } else
                System.out.println("--There is no building.");

            break;

        case 2:
            if (checkDeletion(backStreet, build)) {
                System.out.println("\n" + build);
                System.out.println("--Building has deleted.");
            } else
                System.out.println("--There is no building.");

            break;

        default:
            System.out.println("--Invalid choice!");
            break;
    }

    displayBuildings();

}
```

Basic Array : O(n) – findBuilding Θ(1) + checkDeletion Θ(1) + displayBuilding O(n) = O(n)
Array List : O(n) – findBuilding O(n) + checkDeletion O(n) + displayBuilding O(n) = O(n)
Linked List : O(n^2) – findBuilding O(n^2) + checkDeletion O(n) + displayBuilding O(n^2) = O(n^2)
LD Linked List : O(n^2) – findBuilding O(n^2) + checkDeletion O(n) + displayBuilding O(n^2) = O(n^2)

```java
private Object findBuilding(int sideChoice, int position) {

    int buildPos = 0;
    int buildLength = 0;
    int buildBound = 0;

    Object build = new Object();

    if (sideChoice == 1) { // frontStreet
        for (int i = 0; i < frontStreet.size(); i++) {
            buildPos = (frontStreet.get(i) instanceof Playground)
                    ? ((Playground) frontStreet.get(i)).getPosition()
                    : ((Building) frontStreet.get(i)).getPosition();
            buildLength = (frontStreet.get(i) instanceof Playground)
                    ? ((Playground) frontStreet.get(i)).getLength()
                    : ((Building) frontStreet.get(i)).getLength();

            buildBound = buildPos + buildLength;
            if (position < buildBound && position >= buildPos) {
                build = frontStreet.get(i);
                break;
            }
        }
    } else if (sideChoice == 2) { // backStreet
        for (int i = 0; i < backStreet.size(); i++) {
            buildPos = (backStreet.get(i) instanceof Playground)
                    ? ((Playground) backStreet.get(i)).getPosition()
                    : ((Building) backStreet.get(i)).getPosition();
            buildLength = (backStreet.get(i) instanceof Playground)
                    ? ((Playground) backStreet.get(i)).getLength()
                    : ((Building) backStreet.get(i)).getLength();

            buildBound = buildPos + buildLength;
            if (position < buildBound && position >= buildPos) {
                build = backStreet.get(i);
                break;
            }
        }
    }

    return build;
}
```

Basic Array : Θ(1) – It has reference (streetArray[position])
Array List : O(n) - get(i) time comp.(O(1)) * for loop = O(n)
Linked List : O(n^2) – get(i) time comp.(O(n)) * for loop = O(n^2)
LD Linked List : O(n^2) - get(i) time comp.(O(n)) * for loop = O(n^2)

```java
private boolean checkDeletion(ArrayList<Object> streetArray, Object build) {

    boolean flag = false;
    int temp, temp2;

    if (build instanceof Playground) {
        temp = Playground.getPlaygroundNum();
        temp2 = Playground.getPlaygroundLength();
        Playground.setPlaygroundNum(temp - 1);
        Playground.setPlaygroundLength(temp2 - ((Playground) build).getLength());
    }

    if (streetArray.remove(build)) {
        flag = true;
    }

    return flag;
}
```

Basic Array : Θ(1) – It has reference (streetArray[position])
Array List : O(n) - remove
Linked List : O(n) - remove
LD Linked List : O(n) - remove

```java
public int remainingLength() {

    int count = 0;
    for (int i = 0; i < frontStreet.size(); i++) {
        if (frontStreet.get(i) instanceof Building)
            count += ((Building) frontStreet.get(i)).getLength();
        else if (frontStreet.get(i) instanceof Playground)
            count += ((Playground) frontStreet.get(i)).getLength();
    }
    for (int i = 0; i < backStreet.size(); i++) {
        if (backStreet.get(i) instanceof Building)
            count += ((Building) backStreet.get(i)).getLength();
        else if (backStreet.get(i) instanceof Playground)
            count += ((Playground) backStreet.get(i)).getLength();
    }

    return streetLength * 2 - count;
}
```

Basic Array : O(n) –for loop through streetArray
Array List : O(n) - get(i) time comp.(O(1)) * for loop = O(n)
Linked List : O(n^2) – get(i) time comp.(O(n)) * for loop = O(n^2)
LD Linked List : O(n^2) - get(i) time comp.(O(n)) * for loop = O(n^2)

```java
public boolean displayBuildings() {

    boolean emptyFlag = false;
    boolean emptyFlag2 = false;

    System.out.println("\nFront of street: ");

    if (frontStreet.isEmpty()) {
        System.out.println("--This side is empty.");
        emptyFlag = true;
    } else {
        for (int i = 0; i < frontStreet.size(); i++) {
            System.out.println(frontStreet.get(i));
        }
    }

    System.out.println("\nBack of street: ");

    if (backStreet.isEmpty()) {
        System.out.println("--This side is empty.\n");
        emptyFlag2 = true;
    } else {
        for (int i = 0; i < backStreet.size(); i++) {
            System.out.println(backStreet.get(i));
        }
        System.out.println();
    }

    if (emptyFlag && emptyFlag2)
        return false;
    else
        return true;

}
```

Basic Array : O(n) – for loop(O(n))
Array List : O(n) – get(i) time comp.(O(1)) * for loop = O(n)
Linked List : O(n^2) - get(i) time comp.(O(n)) * for loop = O(n^2)
LD Linked List : O(n^2) - get(i) time comp.(O(n)) * for loop = O(n^2)

```java
public int playgroundNumber() {
    return Playground.getPlaygroundNum();
}
```

Basic Array : Θ(1)
Array List : Θ(1)
Linked List : Θ(1)
LD Linked List : Θ(1)

```java
public String playgroundRatio() {
    return Playground.getPlaygroundLength() + "/" + (streetLength * 2);
}
```

Basic Array : Θ(1)
Array List : Θ(1)
Linked List : Θ(1)
LD Linked List : Θ(1)

```java
public int totalBuildingLength() {
    int result = 0;

    for (int i = 0; i < backStreet.size(); i++) {
        if(backStreet.get(i) instanceof Building)
            result += ((Building) backStreet.get(i)).getLength();
    }
    for (int i = 0; i < frontStreet.size(); i++) {
        if(frontStreet.get(i) instanceof Building)
            result += ((Building) frontStreet.get(i)).getLength();
    }

    return result;
}
```

Basic Array : O(n) – for loop
Array List : O(n) – get(i) time comp.(O(1)) * for loop = O(n)
Linked List : Θ(n^2) - get(i) time comp.(O(n)) * for loop = O(n^2)
LD Linked List : Θ(n^2) - get(i) time comp.(O(n)) * for loop = O(n^2)

```java
*/
public void displaySilhouette() {

    boolean flag = true;
    int index = 0;
    int maxHeight = 0;
    int lastHeight = 0;
    int frontBuilding = 0;
    int backBuilding = 0;
    int column = 0;
    int temp = 0;
    boolean check = false;

    int frontHeight = 0;
    int backHeight = 0;
    int maxSize = (frontStreet.size() >= backStreet.size()) ? frontStreet.size() : backStreet.size();
                    int maxSize - StreetAL.displaySilhouette()
    for (int i = 0; i < maxSize; i++) {
        if (i < frontStreet.size()) {
            frontHeight = (frontStreet.get(i) instanceof Building)
                    ? ((Building) frontStreet.get(i)).getHeight()
                    : (frontStreet.get(i) instanceof Playground) ? Playground.getHeight() : 0;
        } else
            frontHeight = 0;

        if (i < backStreet.size()) {
            backHeight = (backStreet.get(i) instanceof Building)
                    ? ((Building) backStreet.get(i)).getHeight()
                    : (backStreet.get(i) instanceof Playground) ? Playground.getHeight() : 0;
        } else
            backHeight = 0;

        column = (frontHeight >= backHeight) ? frontHeight : backHeight;
        column = (column > temp) ? column : temp;
        temp = column;

    }

    int bound = (streetLength > (column + 1)) ? streetLength : (column + 1);

    List<List<Character>> silhouette = new ArrayList<List<Character>>(bound);

    for (int i = 0; i < bound; i++) {
        silhouette.add(new ArrayList<Character>());
    }

    for (int i = 0; i < bound; i++) {
        for (int j = 0; j < bound; j++) {
            silhouette.get(i).add(j, ' ');
        }
    }


    Object build = new Object();
    Object build2 = new Object();
```

```
        for (int i = 0; i < streetLength; i++) {

            check = false;
            if (checkPosition(i) == 1) {

                if (flag) {
                    index = i;
                    flag = false;
                }

                build = findBuilding(1, i);
                build2 = findBuilding(2, i);
                if ((!(build instanceof Playground) && !(build instanceof Building))
                        && !(build2 instanceof Playground) && !(build2 instanceof Building)) {
                    check = true;
                }

                if (check)
                    continue;

                frontBuilding = (build instanceof Building)
                        ? ((Building) build).getHeight()
                        : (build instanceof Playground) ? Playground.getHeight() : 0;
                backBuilding = (build2 instanceof Building)
                        ? ((Building) build2).getHeight()
                        : (build2 instanceof Playground) ? Playground.getHeight() : 0;

                maxHeight = frontBuilding >= backBuilding ? frontBuilding : backBuilding;

                if (lastHeight == maxHeight) {
                    silhouette.get(index).set(maxHeight, '_');
                    index++;
                } else {
                    if (lastHeight < maxHeight) {
                        for (int j = lastHeight; j < maxHeight; j++) {
                            silhouette.get(index).set(j, '|');
                            if (j == maxHeight - 1) {
                                silhouette.get(index + 1).set(j + 1, '_');
                                index += 2;
                            }
                        }
                    }

                    else if (lastHeight > maxHeight) {
                        for (int j = lastHeight - 1; j >= maxHeight; j--) {
                            silhouette.get(index).set(j, '|');
                            if (j == maxHeight) {
                                silhouette.get(index + 1).set(maxHeight, '_');
                                index += 2;
                            }
                        }
                    }
                    lastHeight = maxHeight;
                }

            }
```

```
            else {
                if (lastHeight == 0 && index < bound - 1) {
                    silhouette.get(index).set(0, '_');
                    index++;
                } else {
                    for (int j = lastHeight; lastHeight != 0 && j > 0; j--) {
                        silhouette.get(index).set(j - 1, '|');
                        if (j == 1 && index + 1 != streetLength) {
                            silhouette.get(index + 1).set(0, '_');
                            index += 2;
                        }
                    }
                    lastHeight = 0;
                }
            }

        }

        for (int i = column; i >= 0; i--) {
            for (int j = 0; j < bound; j++) {
                System.out.print(silhouette.get(j).get(i));
            }
            System.out.println();
        }

        for (int j = 0; j < bound; j++) {
            if (j == 0 || j == bound - 1)
                System.out.print('|');
            else
                System.out.print('_');
        }

        System.out.println();

    }
```

Basic Array : O(n^2) – nested loop
Array List : O(n^2) – get(i)^2 -> (O(1)) * nested loop = O(n^2)
Linked List : Θ(n^4) - get(i)^2 -> (O(n^2)) * nested loop = O(n^4)
LD Linked List : Θ(n^4) - get(i)^2 - > (O(n^2)) * nested loop = O(n^4)

```java
private int checkPosition(int index) {

    int buildPos = 0;
    int buildLength = 0;
    int buildBound = 0;
    for (int i = 0; i < frontStreet.size(); i++) {
        buildPos = (frontStreet.get(i) instanceof Playground)
                ? ((Playground) frontStreet.get(i)).getPosition()
                : ((Building) frontStreet.get(i)).getPosition();
        buildLength = (frontStreet.get(i) instanceof Playground)
                ? ((Playground) frontStreet.get(i)).getLength()
                : ((Building) frontStreet.get(i)).getLength();

        buildBound = buildPos + buildLength;
        if (index < buildBound && index >= buildPos)
            return 1;
    }
    for (int i = 0; i < backStreet.size(); i++) {
        buildPos = (backStreet.get(i) instanceof Playground)
                ? ((Playground) backStreet.get(i)).getPosition()
                : ((Building) backStreet.get(i)).getPosition();
        buildLength = (backStreet.get(i) instanceof Playground)
                ? ((Playground) backStreet.get(i)).getLength()
                : ((Building) backStreet.get(i)).getLength();

        buildBound = buildPos + buildLength;
        if (index < buildBound && index >= buildPos)
            return 1;
    }

    return 0;
}
```

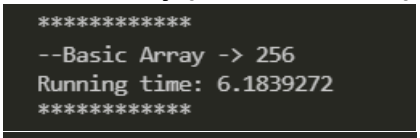
Basic Array : O(n) – for loop
Array List : O(n) – get(i) time comp.(O(1)) * for loop = O(n)
Linked List : Θ(n^2) - get(i) time comp.(O(n)) * for loop = O(n^2)
LD Linked List : Θ(n^2) - get(i) time comp.(O(n)) * for loop = O(n^2)

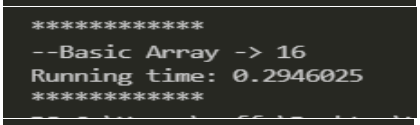
Theoretical running times

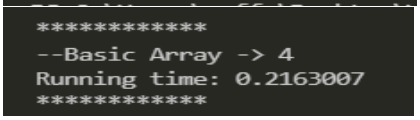
Basic Array: O(n^2)
Array List: O(n^2)
Linked List: O(n^4)
LD Linked List: O(n^4)

Experimental running times

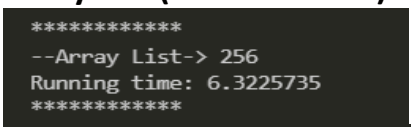
Basic Array (Different size)

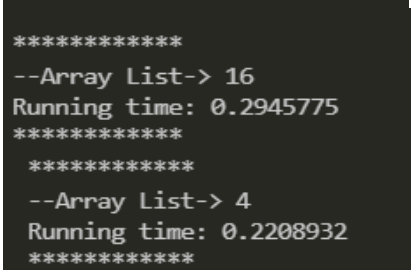




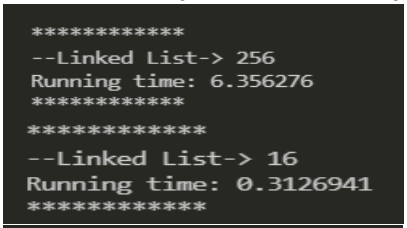


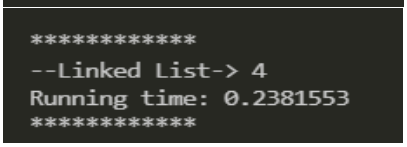
Array List (Different size)



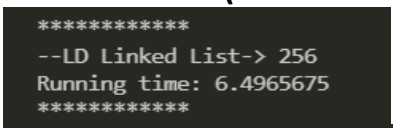


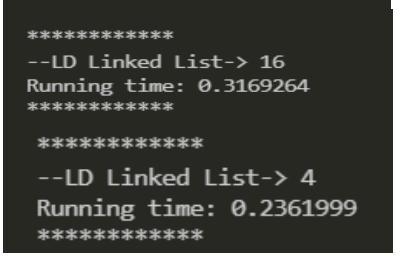
Linked List (Different size)





LD Linked List (Different size)





Conclusion

They have more or less the same runtimes. It may be due to the operating system.

IMPORTANT NOTE!

Driver functions on ArrayList, LinkedList and LDLinkedList are accidentally got mixed up with what I wrote earlier. All parts are working correctly but may not be displayed when you call driver function. You can add anything you wanted.