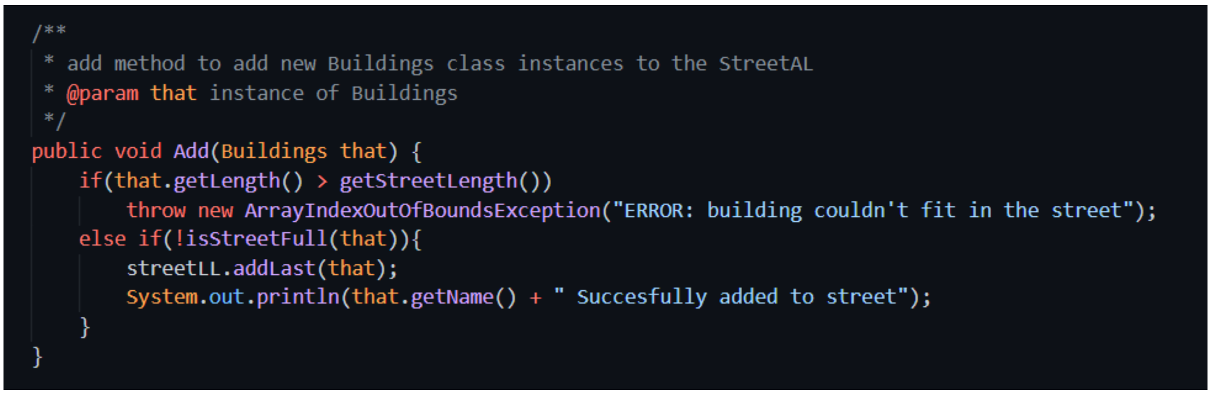
GIT Department of Computer Engineering
CSE 222/505 - Spring 2022
Homework 3
Time Complexity Report

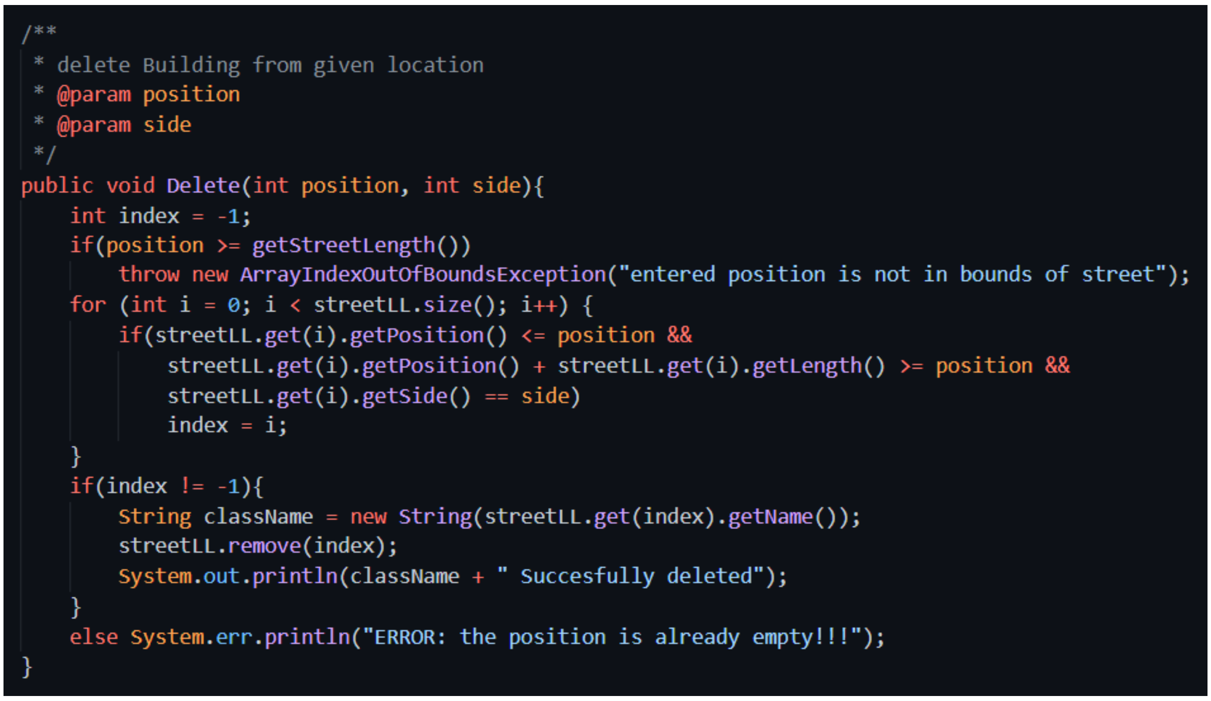
Emircan Demirel
1901042674

Linked List Container:



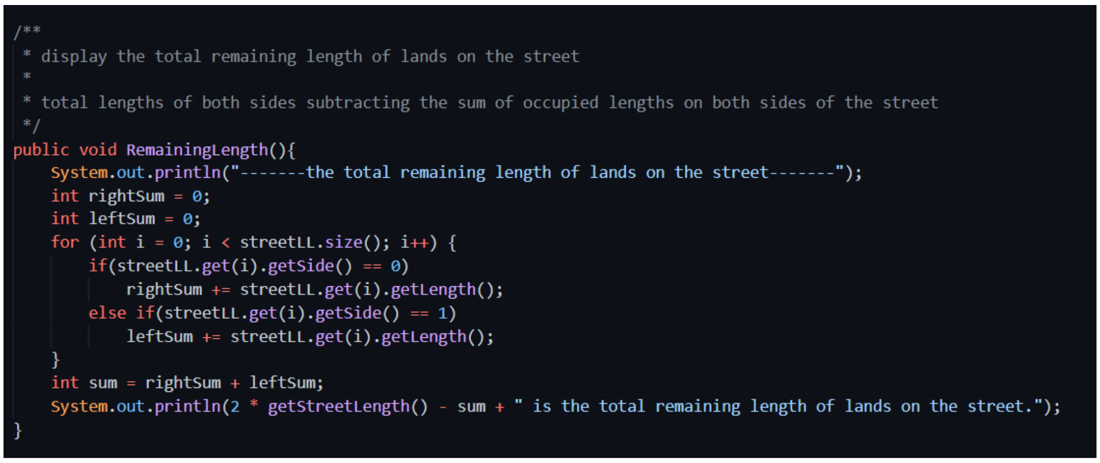
```java
/**
 * add method to add new Buildings class instances to the StreetAL
 * @param that instance of Buildings
 */
public void Add(Buildings that) {
    if(that.getLength() > getStreetLength())
        throw new ArrayIndexOutOfBoundsException("ERROR: building couldn't fit in the street");
    else if(!isStreetFull(that)){
        streetLL.addLast(that);
        System.out.println(that.getName() + " Succesfully added to street");
    }
}
```

Linear time: θ(n)



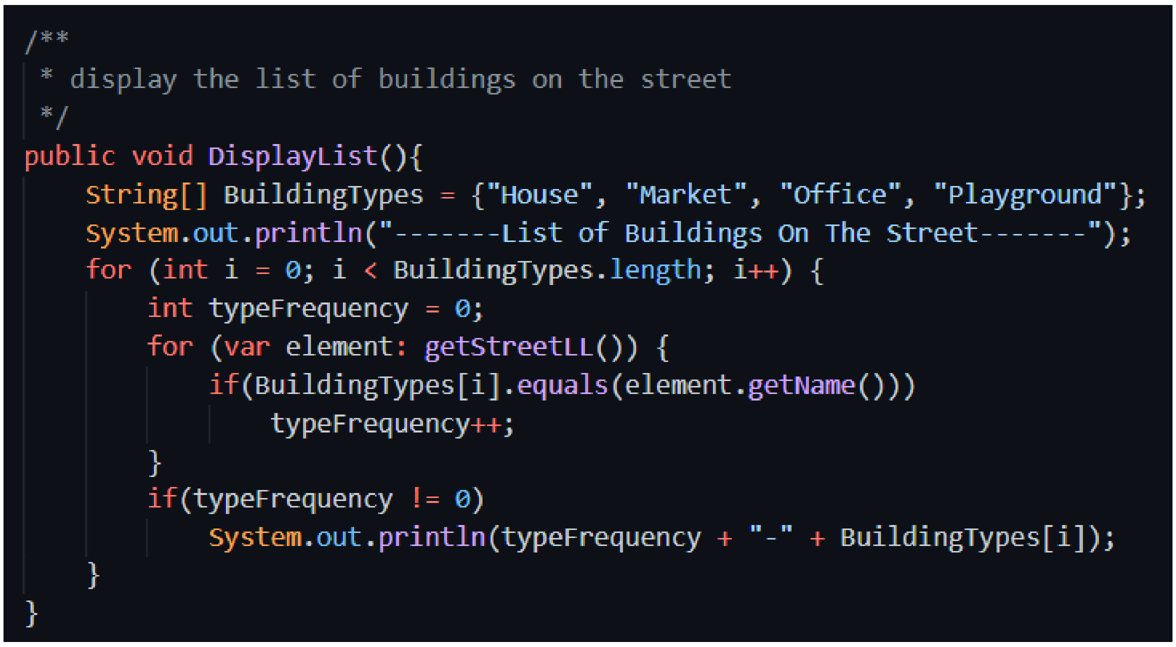
```java
/**
 * delete Building from given location
 * @param position
 * @param side
 */
public void Delete(int position, int side){
    int index = -1;
    if(position >= getStreetLength())
        throw new ArrayIndexOutOfBoundsException("entered position is not in bounds of street");
    for (int i = 0; i < streetLL.size(); i++) {
        if(streetLL.get(i).getPosition() <= position &&
            streetLL.get(i).getPosition() + streetLL.get(i).getLength() >= position &&
            streetLL.get(i).getSide() == side)
            index = i;
    }
    if(index != -1){
        String className = new String(streetLL.get(index).getName());
        streetLL.remove(index);
        System.out.println(className + " Succesfully deleted");
    }
    else System.err.println("ERROR: the position is already empty!!!");
}
```

Linear time: θ(n)



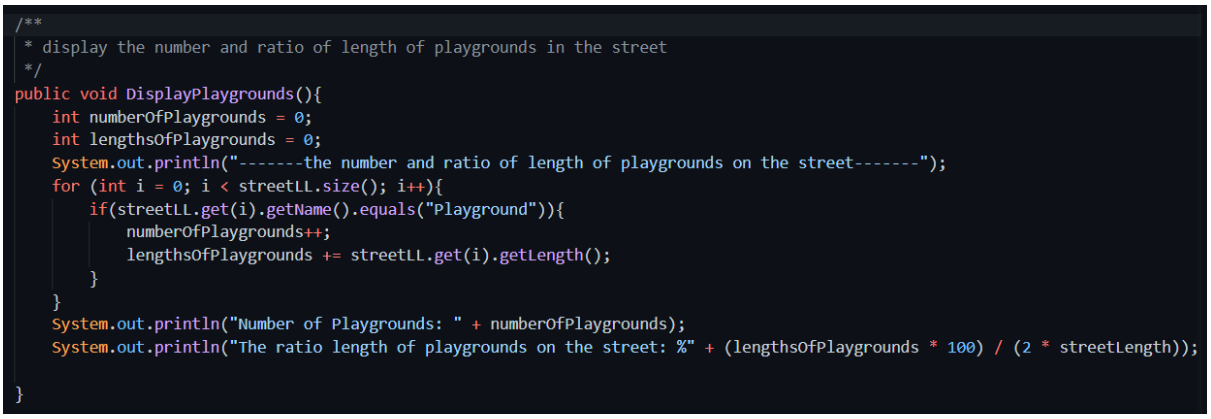
```java
/**
 * display the total remaining length of lands on the street
 *
 * total lengths of both sides subtracting the sum of occupied lengths on both sides of the street
 */
public void RemainingLength(){
    System.out.println("-------the total remaining length of lands on the street-------");
    int rightSum = 0;
    int leftSum = 0;
    for (int i = 0; i < streetLL.size(); i++) {
        if(streetLL.get(i).getSide() == 0)
            rightSum += streetLL.get(i).getLength();
        else if(streetLL.get(i).getSide() == 1)
            leftSum += streetLL.get(i).getLength();
    }
    int sum = rightSum + leftSum;
    System.out.println(2 * getStreetLength() - sum + " is the total remaining length of lands on the street.");
}
```

Linear time: θ(n)



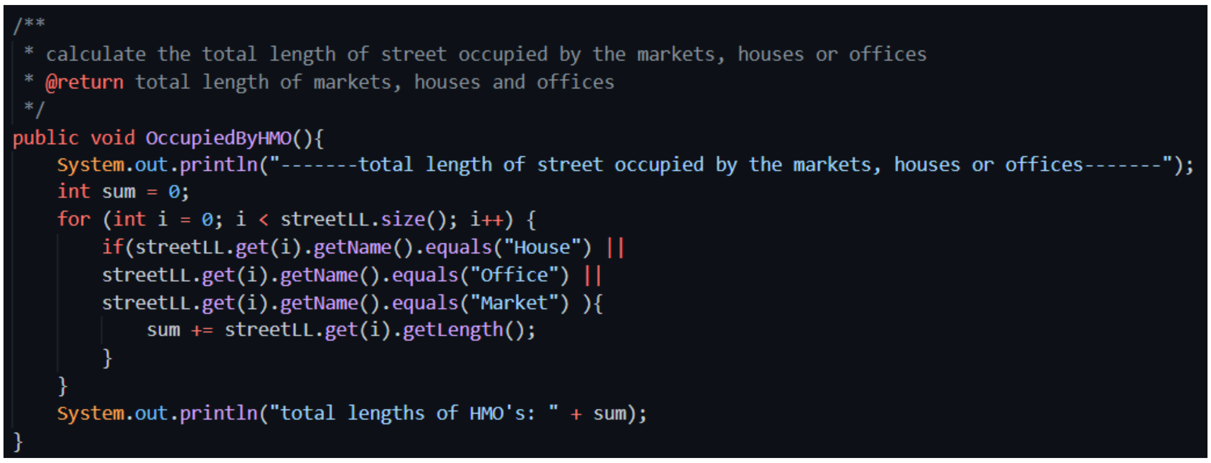
```java
/**
 * display the list of buildings on the street
 */
public void DisplayList(){
    String[] BuildingTypes = {"House", "Market", "Office", "Playground"};
    System.out.println("-------List of Buildings On The Street-------");
    for (int i = 0; i < BuildingTypes.length; i++) {
        int typeFrequency = 0;
        for (var element: getStreetLL()) {
            if(BuildingTypes[i].equals(element.getName()))
                typeFrequency++;
        }
        if(typeFrequency != 0)
            System.out.println(typeFrequency + "-" + BuildingTypes[i]);
    }
}
```

Quadratic time: θ(n^2)



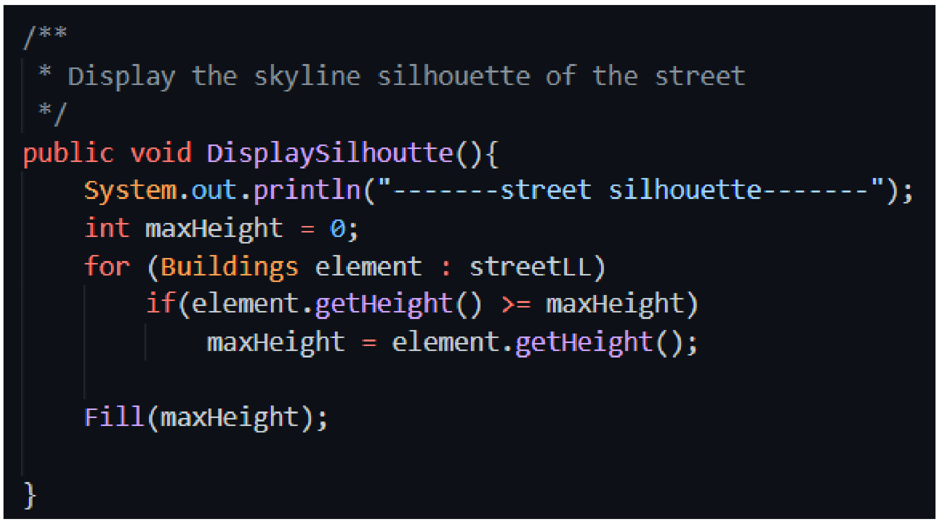
```java
/**
 * display the number and ratio of length of playgrounds in the street
 */
public void DisplayPlaygrounds(){
    int numberOfPlaygrounds = 0;
    int lengthsOfPlaygrounds = 0;
    System.out.println("-------the number and ratio of length of playgrounds on the street-------");
    for (int i = 0; i < streetLL.size(); i++){
        if(streetLL.get(i).getName().equals("Playground")){
            numberOfPlaygrounds++;
            lengthsOfPlaygrounds += streetLL.get(i).getLength();
        }
    }
    System.out.println("Number of Playgrounds: " + numberOfPlaygrounds);
    System.out.println("The ratio length of playgrounds on the street: %" + (lengthsOfPlaygrounds * 100) / (2 * streetLength));

}
```

Linear Time: θ(n)



```java
/**
 * calculate the total length of street occupied by the markets, houses or offices
 * @return total length of markets, houses and offices
 */
public void OccupiedByHMO(){
    System.out.println("-------total length of street occupied by the markets, houses or offices-------");
    int sum = 0;
    for (int i = 0; i < streetLL.size(); i++) {
        if(streetLL.get(i).getName().equals("House") ||
        streetLL.get(i).getName().equals("Office") ||
        streetLL.get(i).getName().equals("Market") ){
            sum += streetLL.get(i).getLength();
        }
    }
    System.out.println("total lengths of HMO's: " + sum);
}
```

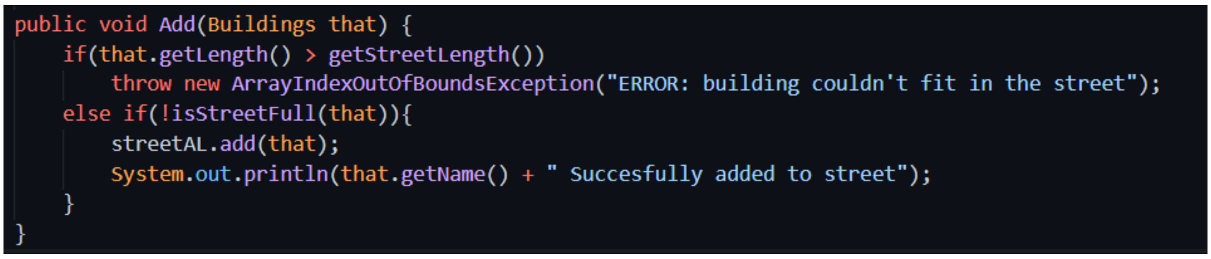
Linear Time: θ(n)



```java
/**
 * Display the skyline silhouette of the street
 */
public void DisplaySilhoutte(){
    System.out.println("-------street silhouette-------");
    int maxHeight = 0;
    for (Buildings element : streetLL)
        if(element.getHeight() >= maxHeight)
            maxHeight = element.getHeight();

    Fill(maxHeight);

}
```

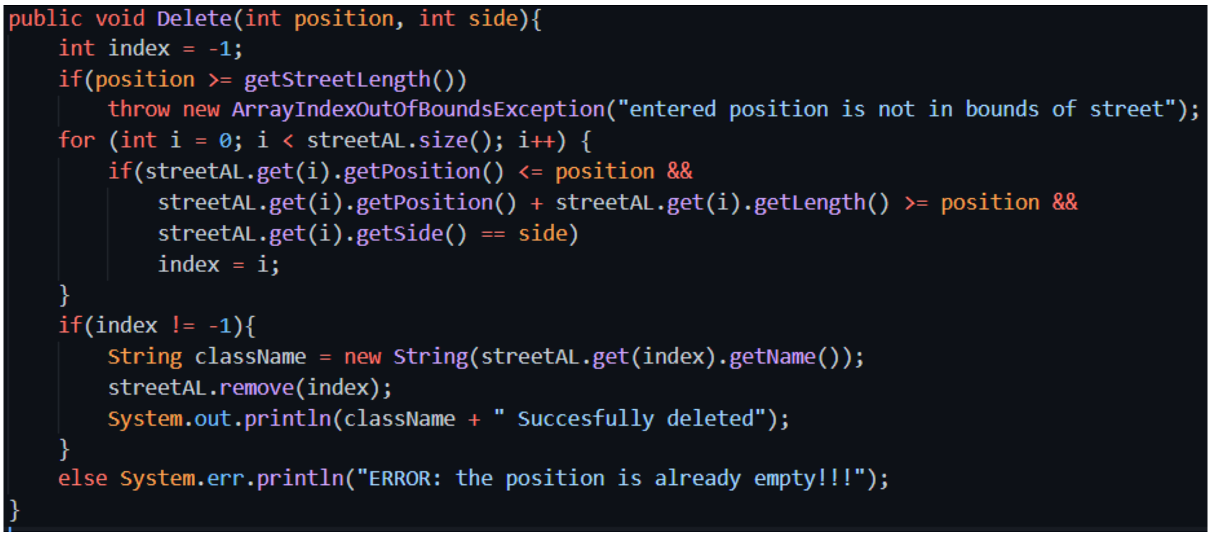
Fill method has θ(n^3) so DisplaySilhouette has cubic time: θ(n^3)

ARRAY LIST CONTAINER:



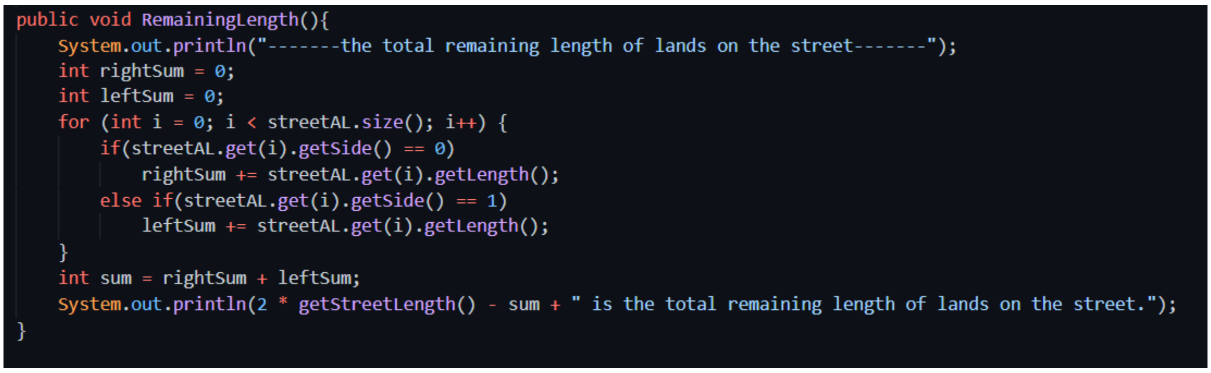
```java
public void Add(Buildings that) {
    if(that.getLength() > getStreetLength())
        throw new ArrayIndexOutOfBoundsException("ERROR: building couldn't fit in the street");
    else if(!isStreetFull(that)){
        streetAL.add(that);
        System.out.println(that.getName() + " Succesfully added to street");
    }
}
```

Linear Time: θ(n)



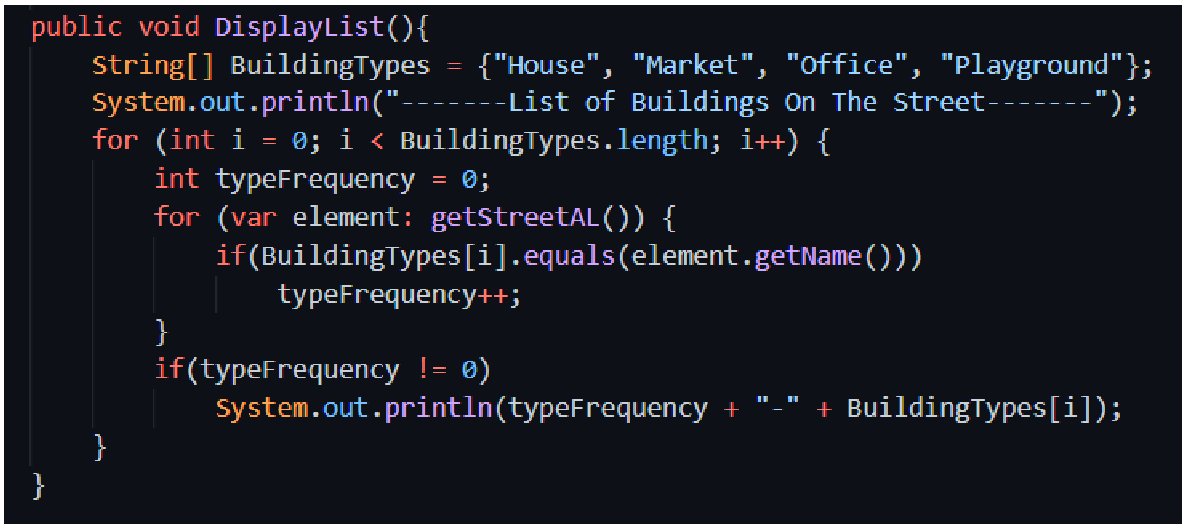
```java
public void Delete(int position, int side){
    int index = -1;
    if(position >= getStreetLength())
        throw new ArrayIndexOutOfBoundsException("entered position is not in bounds of street");
    for (int i = 0; i < streetAL.size(); i++) {
        if(streetAL.get(i).getPosition() <= position &&
            streetAL.get(i).getPosition() + streetAL.get(i).getLength() >= position &&
            streetAL.get(i).getSide() == side)
            index = i;
    }
    if(index != -1){
        String className = new String(streetAL.get(index).getName());
        streetAL.remove(index);
        System.out.println(className + " Succesfully deleted");
    }
    else System.err.println("ERROR: the position is already empty!!!");
}
```

Constant Time: θ(1)



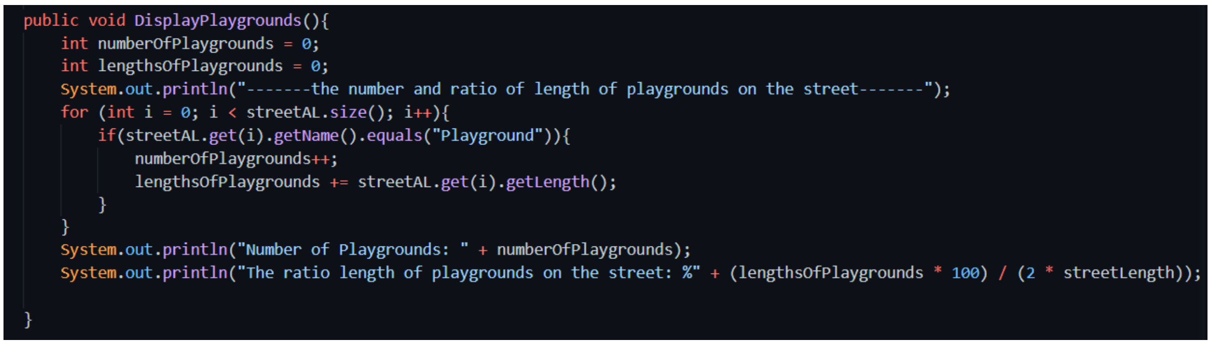
```java
public void RemainingLength(){
    System.out.println("-------the total remaining length of lands on the street-------");
    int rightSum = 0;
    int leftSum = 0;
    for (int i = 0; i < streetAL.size(); i++) {
        if(streetAL.get(i).getSide() == 0)
            rightSum += streetAL.get(i).getLength();
        else if(streetAL.get(i).getSide() == 1)
            leftSum += streetAL.get(i).getLength();
    }
    int sum = rightSum + leftSum;
    System.out.println(2 * getStreetLength() - sum + " is the total remaining length of lands on the street.");
}
```

Linear Time: θ(n)



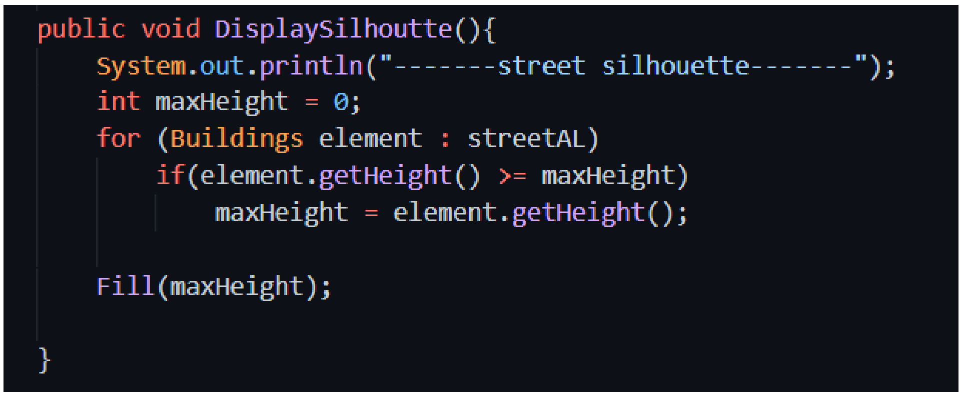
```java
public void DisplayList(){
    String[] BuildingTypes = {"House", "Market", "Office", "Playground"};
    System.out.println("-------List of Buildings On The Street-------");
    for (int i = 0; i < BuildingTypes.length; i++) {
        int typeFrequency = 0;
        for (var element: getStreetAL()) {
            if(BuildingTypes[i].equals(element.getName()))
                typeFrequency++;
        }
        if(typeFrequency != 0)
            System.out.println(typeFrequency + "-" + BuildingTypes[i]);
    }
}
```

Quadratic Time: θ(n^2)



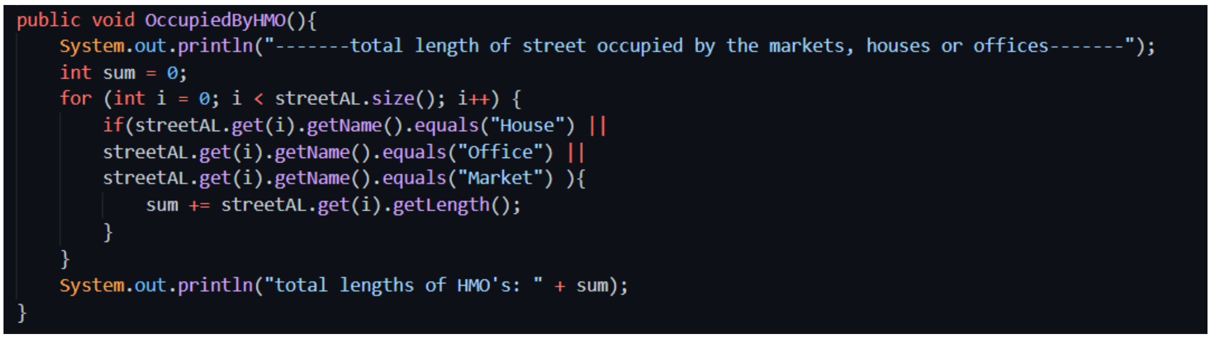
```java
public void DisplayPlaygrounds(){
    int numberOfPlaygrounds = 0;
    int lengthsOfPlaygrounds = 0;
    System.out.println("-------the number and ratio of length of playgrounds on the street-------");
    for (int i = 0; i < streetAL.size(); i++){
        if(streetAL.get(i).getName().equals("Playground")){
            numberOfPlaygrounds++;
            lengthsOfPlaygrounds += streetAL.get(i).getLength();
        }
    }
    System.out.println("Number of Playgrounds: " + numberOfPlaygrounds);
    System.out.println("The ratio length of playgrounds on the street: %" + (lengthsOfPlaygrounds * 100) / (2 * streetLength));

}
```

Linear Time: θ(n)



```java
public void DisplaySilhoutte(){
    System.out.println("-------street silhouette-------");
    int maxHeight = 0;
    for (Buildings element : streetAL)
        if(element.getHeight() >= maxHeight)
            maxHeight = element.getHeight();

    Fill(maxHeight);

}
```

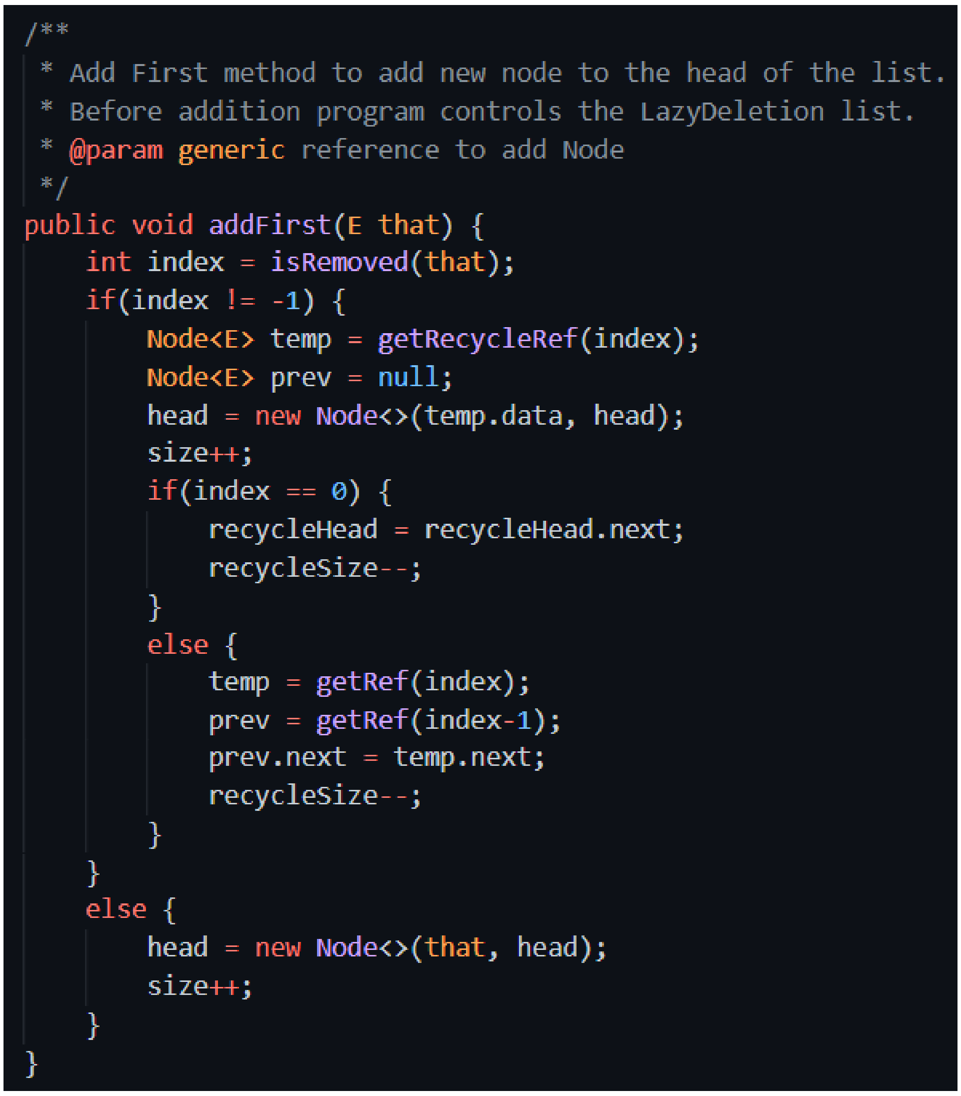
Cubic Time: θ(n^3)



```java
public void OccupiedByHMO(){
    System.out.println("-------total length of street occupied by the markets, houses or offices-------");
    int sum = 0;
    for (int i = 0; i < streetAL.size(); i++) {
        if(streetAL.get(i).getName().equals("House") ||
        streetAL.get(i).getName().equals("Office") ||
        streetAL.get(i).getName().equals("Market") ){
            sum += streetAL.get(i).getLength();
        }
    }
    System.out.println("total lengths of HMO's: " + sum);
}
```

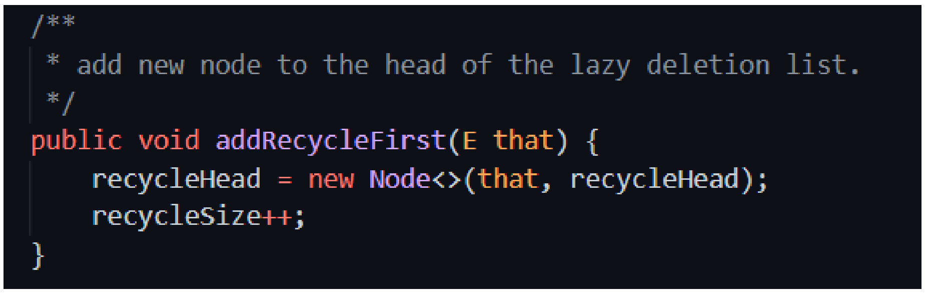
Linear Time: θ(n)

LDLinkedList Container:



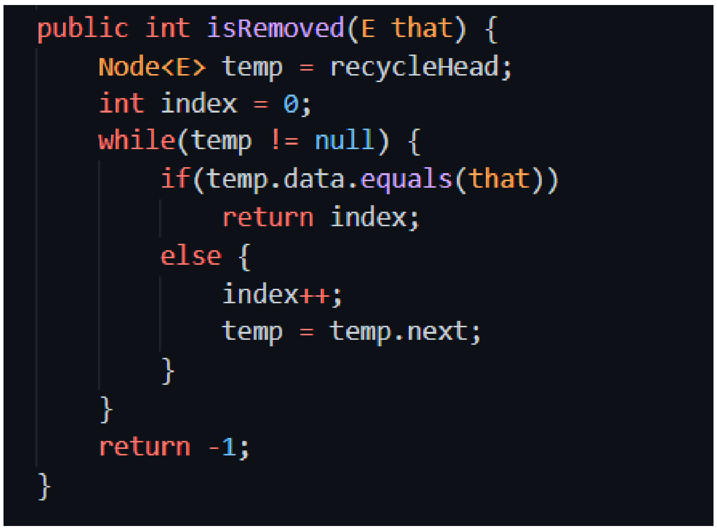
```java
/**
 * Add First method to add new node to the head of the list.
 * Before addition program controls the LazyDeletion list.
 * @param generic reference to add Node
 */
public void addFirst(E that) {
    int index = isRemoved(that);
    if(index != -1) {
        Node<E> temp = getRecycleRef(index);
        Node<E> prev = null;
        head = new Node<>(temp.data, head);
        size++;
        if(index == 0) {
            recycleHead = recycleHead.next;
            recycleSize--;
        }
        else {
            temp = getRef(index);
            prev = getRef(index-1);
            prev.next = temp.next;
            recycleSize--;
        }
    }
    else {
        head = new Node<>(that, head);
        size++;
    }
}
```

Linear Time: θ(n)



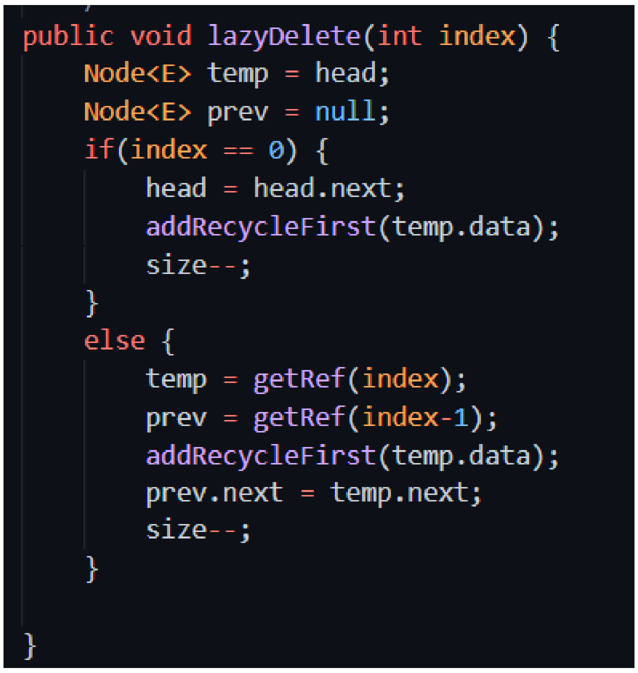
```java
/**
 * add new node to the head of the lazy deletion list.
 */
public void addRecycleFirst(E that) {
    recycleHead = new Node<>(that, recycleHead);
    recycleSize++;
}
```

Constant Time: θ(1)



```java
public int isRemoved(E that) {
    Node<E> temp = recycleHead;
    int index = 0;
    while(temp != null) {
        if(temp.data.equals(that))
            return index;
        else {
            index++;
            temp = temp.next;
        }
    }
    return -1;
}
```

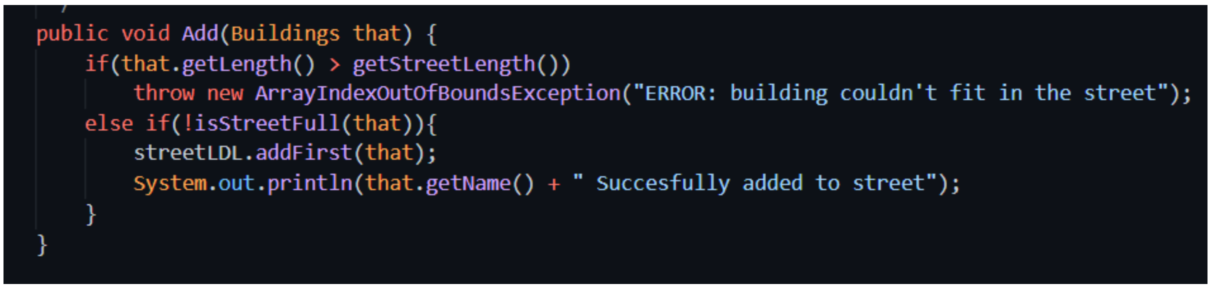
Linear Time: θ(n)



```java
public void lazyDelete(int index) {
    Node<E> temp = head;
    Node<E> prev = null;
    if(index == 0) {
        head = head.next;
        addRecycleFirst(temp.data);
        size--;
    }
    else {
        temp = getRef(index);
        prev = getRef(index-1);
        addRecycleFirst(temp.data);
        prev.next = temp.next;
        size--;
    }

}
```

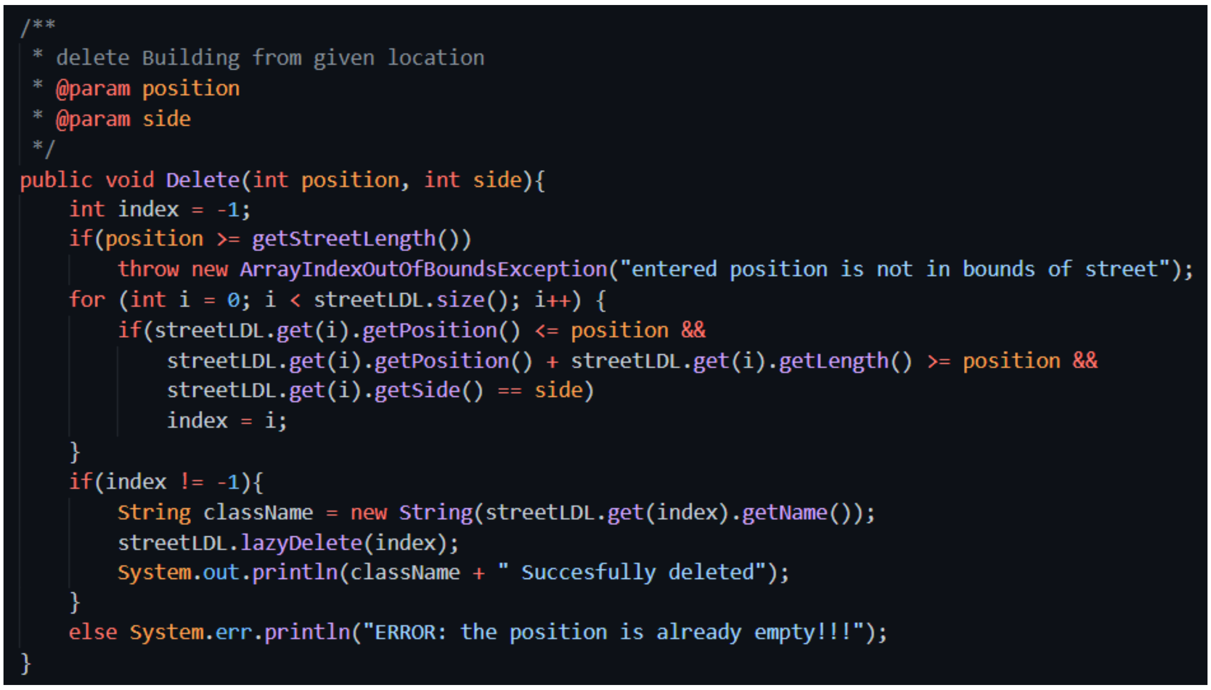
Best Case: θ(1)

Worst Case: θ(n)



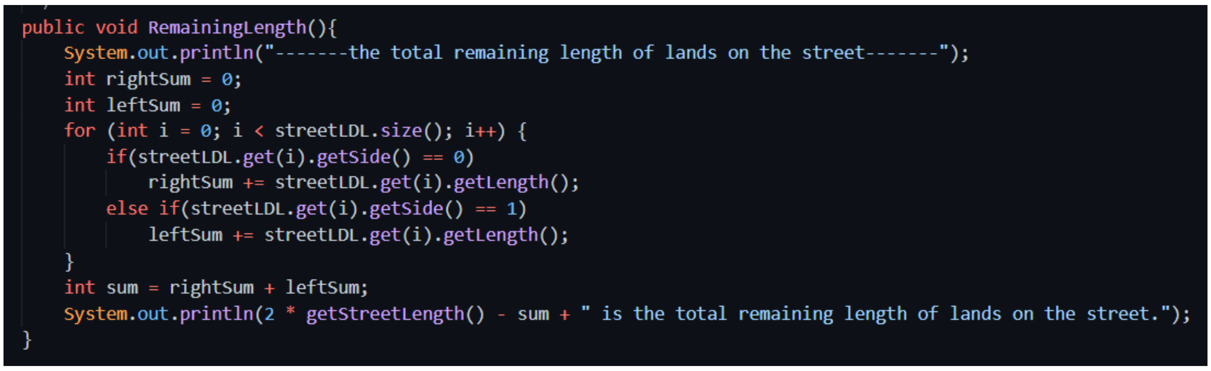
```java
public void Add(Buildings that) {
    if(that.getLength() > getStreetLength())
        throw new ArrayIndexOutOfBoundsException("ERROR: building couldn't fit in the street");
    else if(!isStreetFull(that)){
        streetLDL.addFirst(that);
        System.out.println(that.getName() + " Succesfully added to street");
    }
}
```

Linear Time: θ(n)



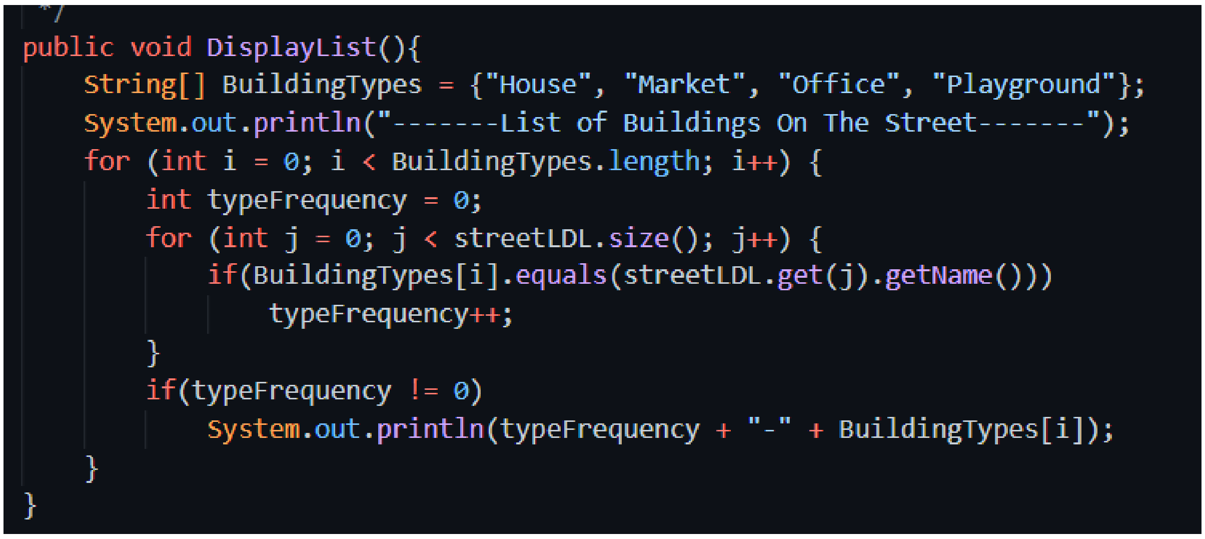
```java
/**
 * delete Building from given location
 * @param position
 * @param side
 */
public void Delete(int position, int side){
    int index = -1;
    if(position >= getStreetLength())
        throw new ArrayIndexOutOfBoundsException("entered position is not in bounds of street");
    for (int i = 0; i < streetLDL.size(); i++) {
        if(streetLDL.get(i).getPosition() <= position &&
            streetLDL.get(i).getPosition() + streetLDL.get(i).getLength() >= position &&
            streetLDL.get(i).getSide() == side)
            index = i;
    }
    if(index != -1){
        String className = new String(streetLDL.get(index).getName());
        streetLDL.lazyDelete(index);
        System.out.println(className + " Succesfully deleted");
    }
    else System.err.println("ERROR: the position is already empty!!!");
}
```

Linear Time: O(n)



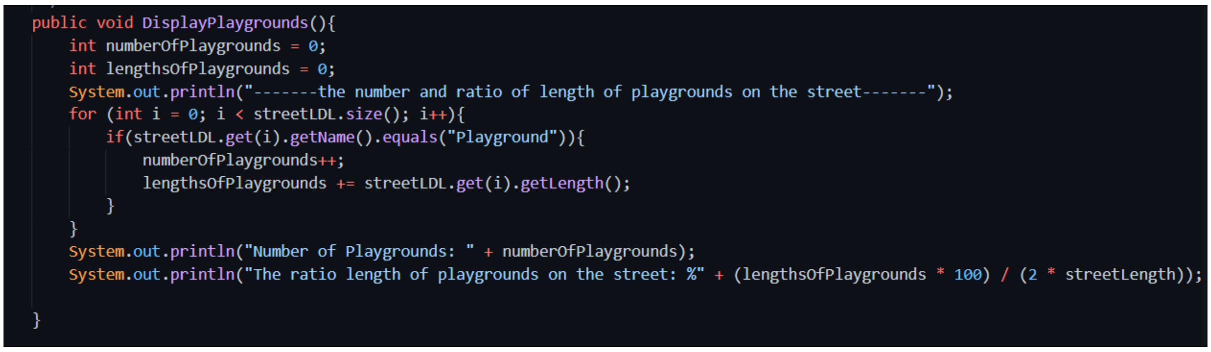
```java
public void RemainingLength(){
    System.out.println("-------the total remaining length of lands on the street-------");
    int rightSum = 0;
    int leftSum = 0;
    for (int i = 0; i < streetLDL.size(); i++) {
        if(streetLDL.get(i).getSide() == 0)
            rightSum += streetLDL.get(i).getLength();
        else if(streetLDL.get(i).getSide() == 1)
            leftSum += streetLDL.get(i).getLength();
    }
    int sum = rightSum + leftSum;
    System.out.println(2 * getStreetLength() - sum + " is the total remaining length of lands on the street.");
}
```

Linear Time: θ(n)

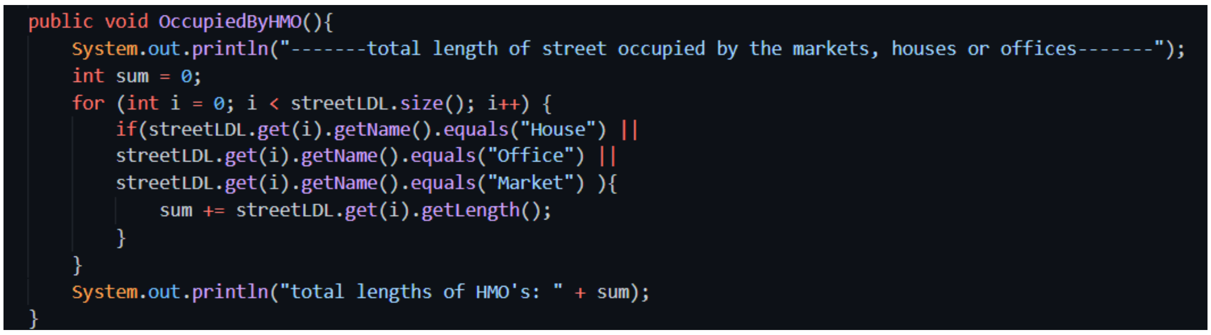


```java
 */
public void DisplayList(){
    String[] BuildingTypes = {"House", "Market", "Office", "Playground"};
    System.out.println("-------List of Buildings On The Street-------");
    for (int i = 0; i < BuildingTypes.length; i++) {
        int typeFrequency = 0;
        for (int j = 0; j < streetLDL.size(); j++) {
            if(BuildingTypes[i].equals(streetLDL.get(j).getName()))
                typeFrequency++;
        }
        if(typeFrequency != 0)
            System.out.println(typeFrequency + "-" + BuildingTypes[i]);
    }
}
```

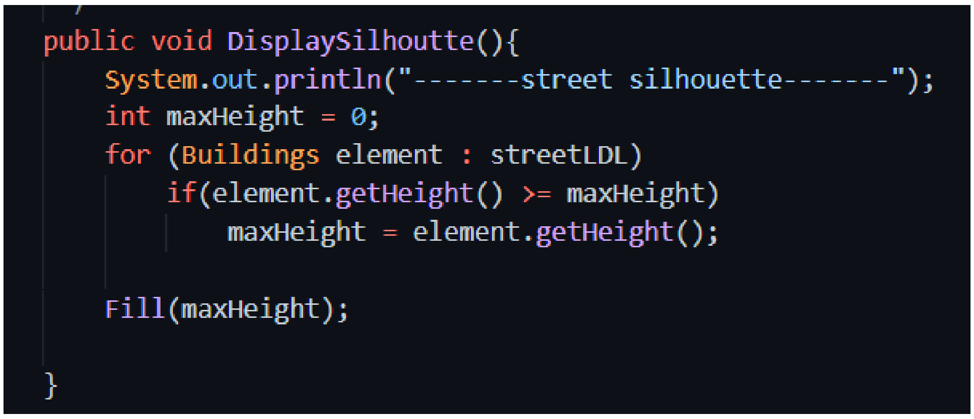
Quadratic Time: θ(n^2)


```java
public void DisplayPlaygrounds(){
    int numberOfPlaygrounds = 0;
    int lengthsOfPlaygrounds = 0;
    System.out.println("-------the number and ratio of length of playgrounds on the street-------");
    for (int i = 0; i < streetLDL.size(); i++){
        if(streetLDL.get(i).getName().equals("Playground")){
            numberOfPlaygrounds++;
            lengthsOfPlaygrounds += streetLDL.get(i).getLength();
        }
    }
    System.out.println("Number of Playgrounds: " + numberOfPlaygrounds);
    System.out.println("The ratio length of playgrounds on the street: %" + (lengthsOfPlaygrounds * 100) / (2 * streetLength));

}
```

Linear Time: θ(n)

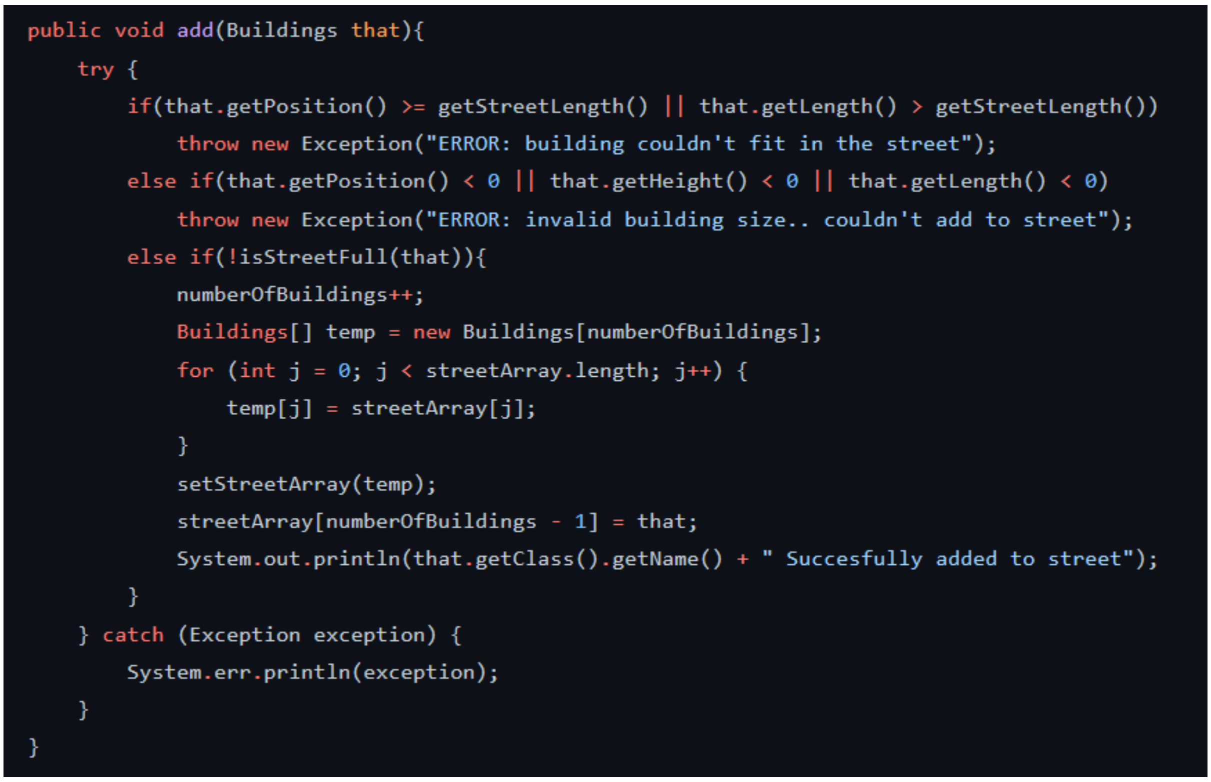

```java
public void OccupiedByHMO(){
    System.out.println("-------total length of street occupied by the markets, houses or offices-------");
    int sum = 0;
    for (int i = 0; i < streetLDL.size(); i++) {
        if(streetLDL.get(i).getName().equals("House") ||
        streetLDL.get(i).getName().equals("Office") ||
        streetLDL.get(i).getName().equals("Market") ){
            sum += streetLDL.get(i).getLength();
        }
    }
    System.out.println("total lengths of HMO's: " + sum);
}
```

Linear Time: θ(n)


```java
public void DisplaySilhoutte(){
    System.out.println("-------street silhouette-------");
    int maxHeight = 0;
    for (Buildings element : streetLDL)
        if(element.getHeight() >= maxHeight)
            maxHeight = element.getHeight();

    Fill(maxHeight);

}
```

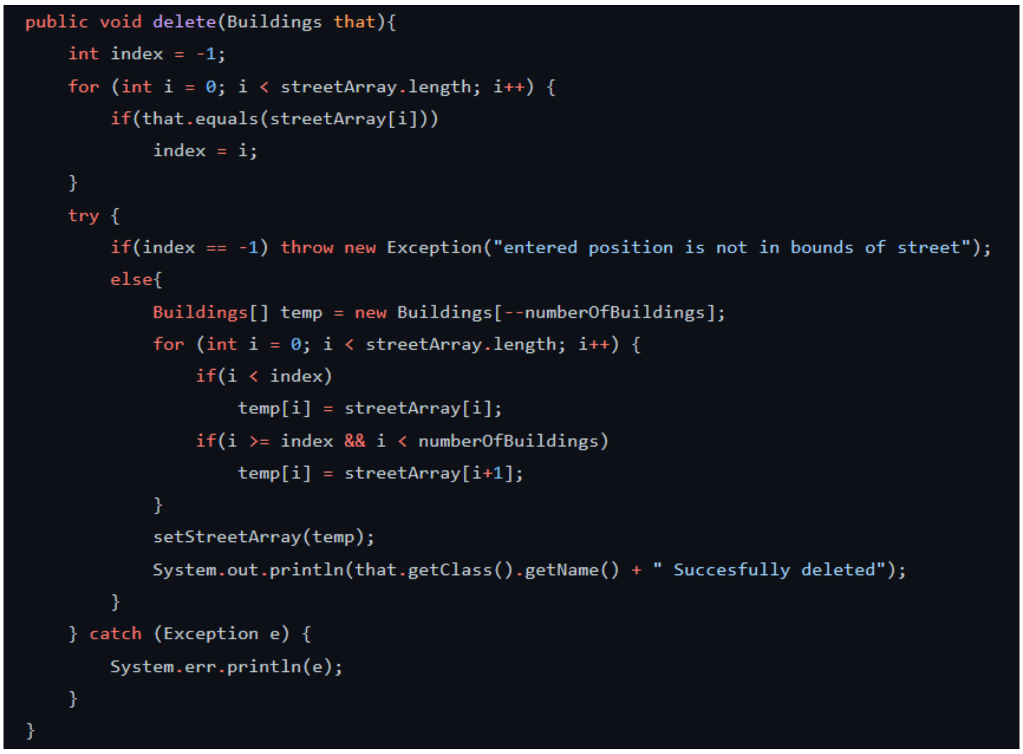
Cubic Time: θ(n^3)

Array Container:



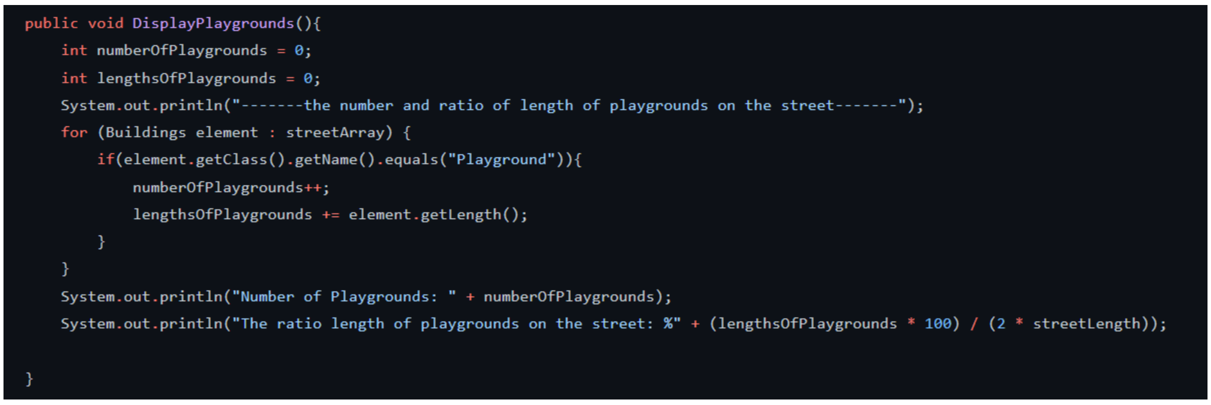
```java
public void add(Buildings that){
    try {
        if(that.getPosition() >= getStreetLength() || that.getLength() > getStreetLength())
            throw new Exception("ERROR: building couldn't fit in the street");
        else if(that.getPosition() < 0 || that.getHeight() < 0 || that.getLength() < 0)
            throw new Exception("ERROR: invalid building size.. couldn't add to street");
        else if(!isStreetFull(that)){
            numberOfBuildings++;
            Buildings[] temp = new Buildings[numberOfBuildings];
            for (int j = 0; j < streetArray.length; j++) {
                temp[j] = streetArray[j];
            }
            setStreetArray(temp);
            streetArray[numberOfBuildings - 1] = that;
            System.out.println(that.getClass().getName() + " Succesfully added to street");
        }
    } catch (Exception exception) {
        System.err.println(exception);
    }
}
```

Linear Time: O(n)



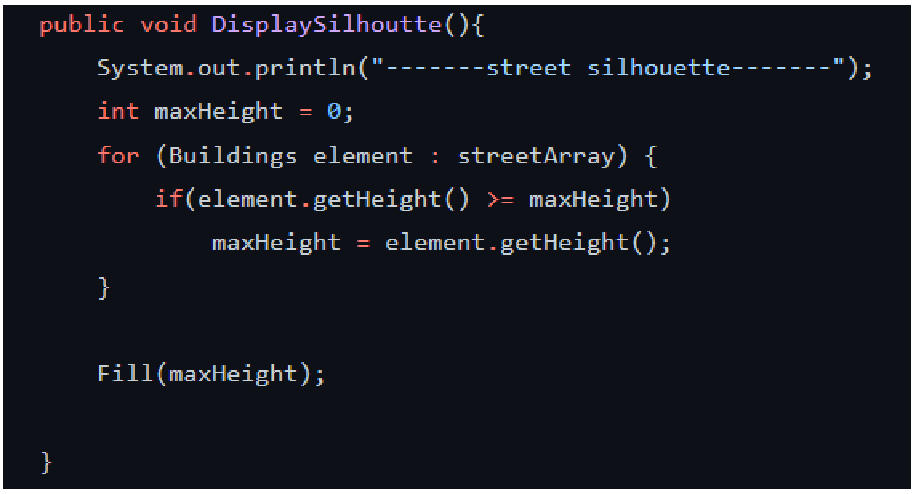
```java
public void delete(Buildings that){
    int index = -1;
    for (int i = 0; i < streetArray.length; i++) {
        if(that.equals(streetArray[i]))
            index = i;
    }
    try {
        if(index == -1) throw new Exception("entered position is not in bounds of street");
        else{
            Buildings[] temp = new Buildings[--numberOfBuildings];
            for (int i = 0; i < streetArray.length; i++) {
                if(i < index)
                    temp[i] = streetArray[i];
                if(i >= index && i < numberOfBuildings)
                    temp[i] = streetArray[i+1];
            }
            setStreetArray(temp);
            System.out.println(that.getClass().getName() + " Succesfully deleted");
        }
    } catch (Exception e) {
        System.err.println(e);
    }
}
```

Linear Time: O(n)



```java
public void DisplayPlaygrounds(){
    int numberOfPlaygrounds = 0;
    int lengthsOfPlaygrounds = 0;
    System.out.println("-------the number and ratio of length of playgrounds on the street-------");
    for (Buildings element : streetArray) {
        if(element.getClass().getName().equals("Playground")){
            numberOfPlaygrounds++;
            lengthsOfPlaygrounds += element.getLength();
        }
    }
    System.out.println("Number of Playgrounds: " + numberOfPlaygrounds);
    System.out.println("The ratio length of playgrounds on the street: %" + (lengthsOfPlaygrounds * 100) / (2 * streetLength));

}
```

Linear Time: θ(n)



```java
public void DisplaySilhoutte(){
    System.out.println("-------street silhouette-------");
    int maxHeight = 0;
    for (Buildings element : streetArray) {
        if(element.getHeight() >= maxHeight)
            maxHeight = element.getHeight();
    }

    Fill(maxHeight);

}
```

Cubic Time: θ(n^3)