Functional Requirements:
- Street class can take structures and show them in silhouette view.
- Structures can be added to the street either directly to the first available empty spaces or a position that user can spe cify.
- Home, Office, Market and Playground are child classes. These have different properties from each other. Program can distinguish them and show which ones of them has inclueded into the street.
- User can change the street view by adding and deleting structures until they decide to exit the program.

Non-functional Requirements:
- Program is shown in the console
- Program end whenever the user ask.
- Program response instantly, storage space requirement is negligible
- Program can run anyplace that jvm can run.

Problem Solution Approach:
LDLinkedList container behaves like a single LinkedList. It uses lazy delegatiton strategy which keeps deleted node s in a LDLinkedList for future use.
There are 3 different programs. One of them uses ArrayList as container other one uses LinkedList and the last one uses LDLinkedList.
StructureContainer holds Structures which is the base abstract class of the Building and Playground.
StreetContainer holds boolean values which indicates what places are empty.
These two classes interact with Street class to keep record of the Structures and show them properly. House, Office, Market and Playground classes thus can be added or removed dynamically.
Street length is determined by user at the start of the program. Height value of the street is determined by the highes t structure which user added and it can dynamically change whenever the user adds higher structure
Program can show the added structures in silhouette view
Program has Menu class which is the class make the user interact with the program. User can add/delete structures a nd can see which ones are on the street from this class.
Driver function is in the Main class. Whenever the program starts it is executed and shows the functions are workin g. Prints silhouette view to show the changes that has been made.
AnalysisResult function shows three different running time. Adding structure, removing structure and printing silho uette.

Test Case:
Add Structures to a street to get a view like in the Homework 1 document.
Delete rightmost structure and show the result.
Delete a structure that doesn't exist to verify error handling is working.
Try to add a structure where is occupied with another structure.
Try to exceed the street by addind larger structure than street.
Show the structures in viewing mode.
Show the running times of adding structure, removing structure, printing silhouette.
Do these using Array(HW1), ArrayList, LinkedList, LDLinkedList.

Adding Structure:

StructureConratiner -> time complexity of adding element to the structure container (end or middle of it)
StreetContainer -> length of the building * height of the building * adding to the street container (end or middle of i t)

Removing Structure:

StructureConratiner -> time complexity of removeing element from the structure container (end or middle of it)
StreetContainer -> length of the building * height of the building * setting the street container element (end or middl e of it)

printSilhouette -> length of the street * height of the street * get or set elements of the container

Array (HW1):
Adding Structure: O(n) + n*n*c -> O(n^2)
Removing Structure: O(n) + n*n*c -> O(n^2)
printSilhouette: n*n*c -> O(n^2)

ArrayList:
Adding Structure: O(n) + n*n*c -> O(n^2)
Removing Structure: O(n) + n*n*c -> O(n^2)
printSilhouette: n*n*c -> O(n^2)

LinkedList:
Adding Structure: O(n) + n*n*n -> O(n^3)
Removing Structure: O(n) + n*n*n -> O(n^3)
printSilhouette: n*n*O(n) -> O(n^3)

LDLinkedList:
Adding Structure: O(n) + n*n*n -> O(n^3)
Removing Structure: O(n) + n*n*n -> O(n^3)
printSilhouette: n*n*O(n) -> O(n^3)