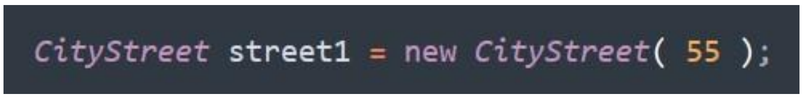
GIT Department of Computer Engineering
CSE 222/505 - Spring 2022
Homework 1 Report

BURAK KOCAUSTA
1901042605

1. SYSTEM REQUIREMENTS

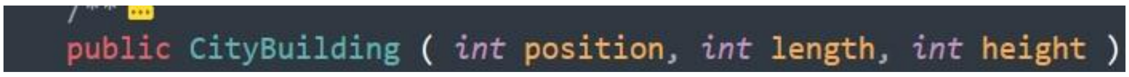
Street is named “CityStreet” in this City Planning Software. It needs two sides to hold buildings. There should be length information to construct to street initially, then buildings can be added to wanted sides.



```
CityStreet street1 = new CityStreet( 55 );
```

In this class buildings are named as “CityBuilding”. There must be position, length information about buildings and street to prevent any conflict between buildings. Sometimes building’s length can be greater than street’s length or some of the building’s position information are not suitable for street. So, position and length information of buildings are required. To print Skyline Silhouette, there must be height information of buildings.

This can be called from derived classes constructors like super(…)

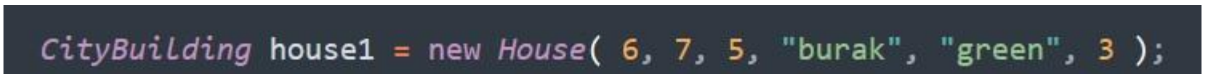


```
    public CityBuilding ( int position, int length, int height )
```

There is no limit for height in this software, but length of street, and buildings are immutable. If you want to change length, you must create new street or building. Building’s position is also immutable, because during insertion of the buildings to street some contracts(conditions) are checked. After insertions, these changes shouldn’t be done from outside. There is delete option in edit mode of street. In this mode that kind of changings can be done.

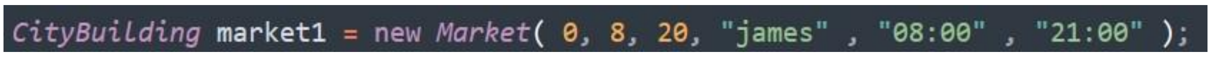
Building is thought as an abstract notion. Many things can be derived from building. In this software, there is house, market, office, and playgrounds. CityBuilding is an abstract reusable class. Other concrete classes are derived from this.

A house requires an owner, color, and number of the room information.



```
CityBuilding house1 = new House( 6, 7, 5, "burak", "green", 3 );
```

A market requires an owner, opening and closing time information.



```
CityBuilding market1 = new Market( 0, 8, 20, "james" , "08:00" , "21:00" );
```

An office also requires an owner, and job type information.

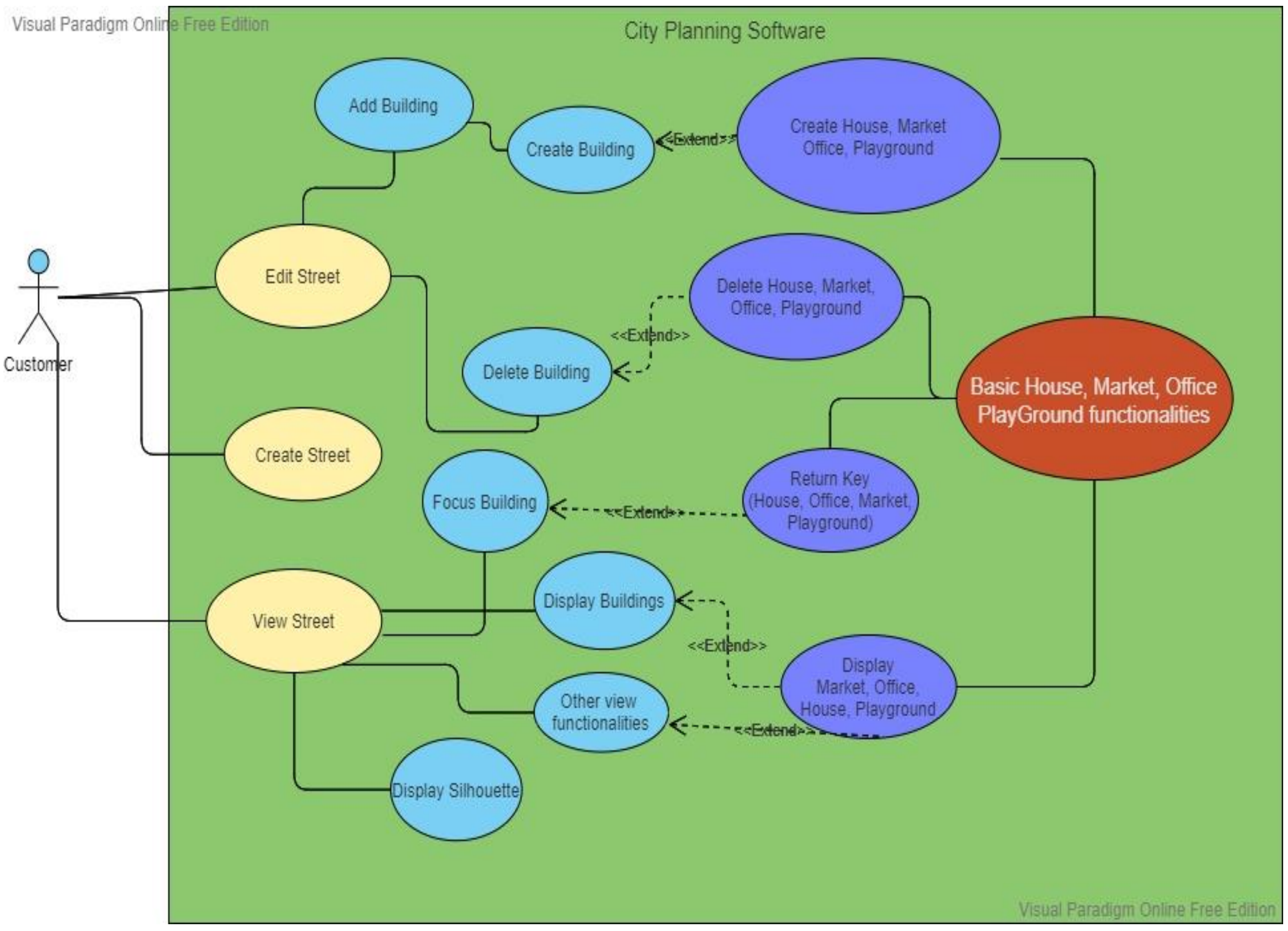
```
CityBuilding office1 = new Office( 10, 12, 20, "rachel", "consulting" );
```

A playground is a simple extension of building it does not require an extra information like others, but its height is automatically 1.

```
CityBuilding playground1 = new Playground ( 31, 4 );
```

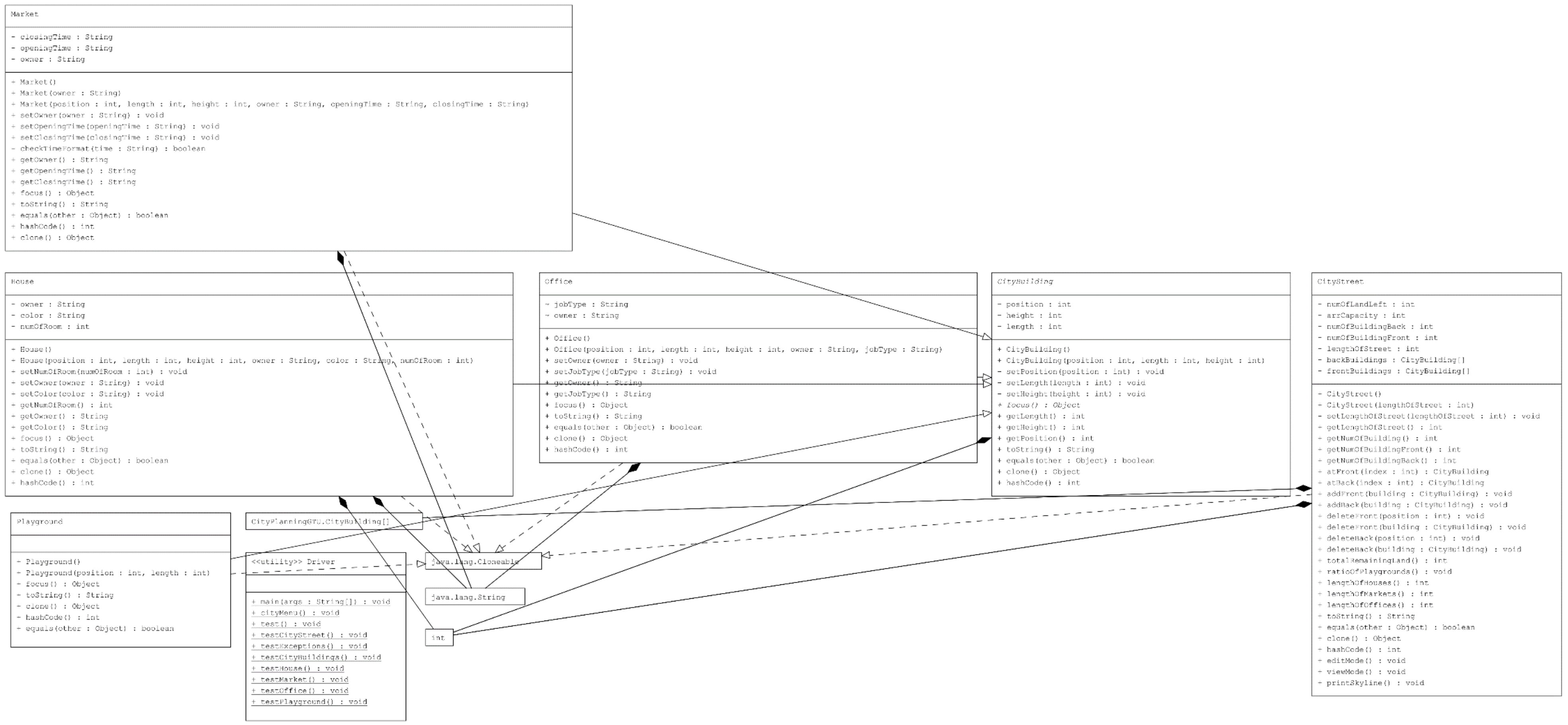
2. **USE CASE AND CLASS DIAGRAMS**

USE CASE DIAGRAM



Visual Paradigm Online Free Edition
City Planning Software
Add Building
Create Building
<<Extend>>
Create House, Market Office, Playground
Customer
Edit Street
Delete House, Market, Office, Playground
<<Extend>>
Delete Building
Basic House, Market, Office PlayGround functionalities
Create Street
Focus Building
<<Extend>>
Return Key (House, Office, Market, Playground)
Display Buildings
View Street
<<Extend>>
Display Market, Office, House, Playground
Other view functionalities
<<Extend>>
Display Silhouette
Visual Paradigm Online Free Edition

CLASS DIAGRAM (High resolution version is inside this directory as .jpg file)


Market
- closingTime : String
- openingTime : String
- owner : String
+ Market()
+ Market(owner : String)
+ Market(position : int, length : int, height : int, owner : String, openingTime : String, closingTime : String)
+ setOwner(owner : String) : void
+ setOpeningTime(openingTime : String) : void
+ setClosingTime(closingTime : String) : void
- checkTimeFormat(time : String) : boolean
+ getOwner() : String
+ getOpeningTime() : String
+ getClosingTime() : String
+ focus() : Object
+ toString() : String
+ equals(other : Object) : boolean
+ hashCode() : int
+ clone() : Object
House
- owner : String
- color : String
- numOfRoom : int
+ House()
+ House(position : int, length : int, height : int, owner : String, color : String, numOfRoom : int)
+ setNumOfRoom(numOfRoom : int) : void
+ setOwner(owner : String) : void
+ setColor(color : String) : void
+ getNumOfRoom() : int
+ getOwner() : String
+ getColor() : String
+ focus() : Object
+ toString() : String
+ equals(other : Object) : boolean
+ clone() : Object
+ hashCode() : int
Office
~ jobType : String
~ owner : String
+ Office()
+ Office(position : int, length : int, height : int, owner : String, jobType : String)
+ setOwner(owner : String) : void
+ setJobType(jobType : String) : void
+ getOwner() : String
+ getJobType() : String
+ focus() : Object
+ toString() : String
+ equals(other : Object) : boolean
+ clone() : Object
+ hashCode() : int
CityBuilding
- position : int
- height : int
- length : int
+ CityBuilding()
+ CityBuilding(position : int, length : int, height : int)
- setPosition(position : int) : void
- setLength(length : int) : void
- setHeight(height : int) : void
+ focus() : Object
+ getLength() : int
+ getHeight() : int
+ getPosition() : int
+ toString() : String
+ equals(other : Object) : boolean
+ clone() : Object
+ hashCode() : int
CityStreet
- numOfLandLeft : int
- arrCapacity : int
- numOfBuildingBack : int
- numOfBuildingFront : int
- lengthOfStreet : int
- backBuildings : CityBuilding[]
- frontBuildings : CityBuilding[]
+ CityStreet()
+ CityStreet(lengthOfStreet : int)
- setLengthOfStreet(lengthOfStreet : int) : void
+ getLengthOfStreet() : int
+ getNumOfBuilding() : int
+ getNumOfBuildingFront() : int
+ getNumOfBuildingBack() : int
+ atFront(index : int) : CityBuilding
+ atBack(index : int) : CityBuilding
+ addFront(building : CityBuilding) : void
+ addBack(building : CityBuilding) : void
+ deleteFront(position : int) : void
+ deleteFront(building : CityBuilding) : void
+ deleteBack(position : int) : void
+ deleteBack(building : CityBuilding) : void
+ totalRemainingLand() : int
+ ratioOfPlaygrounds() : void
+ lengthOfHouses() : int
+ lengthOfMarkets() : int
+ lengthOfOffices() : int
+ toString() : String
+ equals(other : Object) : boolean
+ clone() : Object
+ hashCode() : int
+ editMode() : void
+ viewMode() : void
+ printSkyline() : void
Playground
+ Playground()
+ Playground(position : int, length : int)
+ focus() : Object
+ toString() : String
+ clone() : Object
+ hashCode() : int
+ equals(other : Object) : boolean
CityPlanningGTU.CityBuilding[]
<<utility>> Driver
+ main(args : String[]) : void
+ cityMenu() : void
+ test() : void
+ testCityStreet() : void
+ testExceptions() : void
+ testCityBuildings() : void
+ testHouse() : void
+ testMarket() : void
+ testOffice() : void
+ testPlayground() : void
java.lang.Cloneable
java.lang.String
int

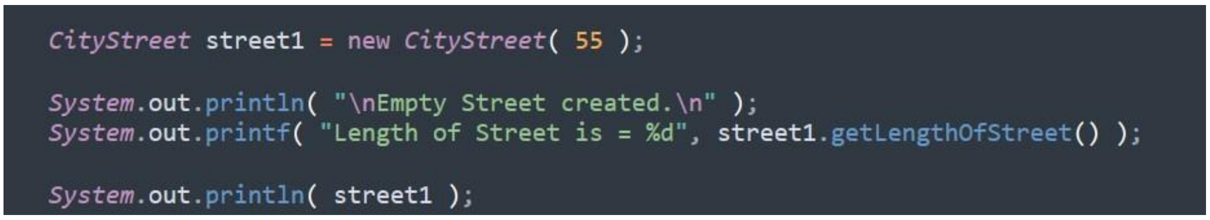
3. PROBLEM SOLUTION APPROACH

In homework document, it is said that there should be one superclass, and as I understand from that statement, an abstract building class must be superclass of house, market, office, and playground. It is ambiguous that playground has "is a" relation with building class, but because of playground holds position, length, and height I decided that playground is an also building. Last problem is where to hold all the buildings, using Java Collection data structure is prohibited, so I decided to hold buildings in dynamically growing array. After that decision, Street has two sides, so two array can be hold in the class. In homework document, it is ambiguous that where should we hold all the buildings. I decided that there must be street class which is named CityStreet and it holds two dynamically growing building array in its private field. I thought these sides as front and back. I doubted if building class should abstract or concrete class. Then I decided to keep it abstract because it can be reusable, and it has a generic enough name. In street class I decided to keep length can only be set during construction. Each buildings setter for length is private. Because after adding buildings to street they can be changing from outside. I decided there is no need to copy during add operation in street. With that decision, it can be easily deleted with references, and changing from outside does not cause anything dangerous because inside building class length, position and height are also can only be set in constructor. There is a focus function in building that causes building to be an abstract class. It returns unique key for every building derived from building. I decided that focus method must return an Object because it can be integer, string, etc. It can be downcasted before usage to see which class calls focus method. And I made this downcasting operation in view mode. I don't set any limitations for street length, but it is recommended that to use it below 120, because when printing skyline silhouette 120 length means 120 characters. After implementing core methods for street class. I should decide where to put edit and view mode methods or how to handle it. They all can be in driver class, but I thought that it is nice option to edit and view for this street class implementation. So, I decided to put edit and view mode inside street class and make them public.

The most difficult part for me in this homework is, printing silhouette. I tried to print silhouette without any storage and step by step. But a very complicated and hard to understand algorithm came up with this idea. I simplified this printing algorithm as far as I can but if I do this printing again, I might choose storage option.

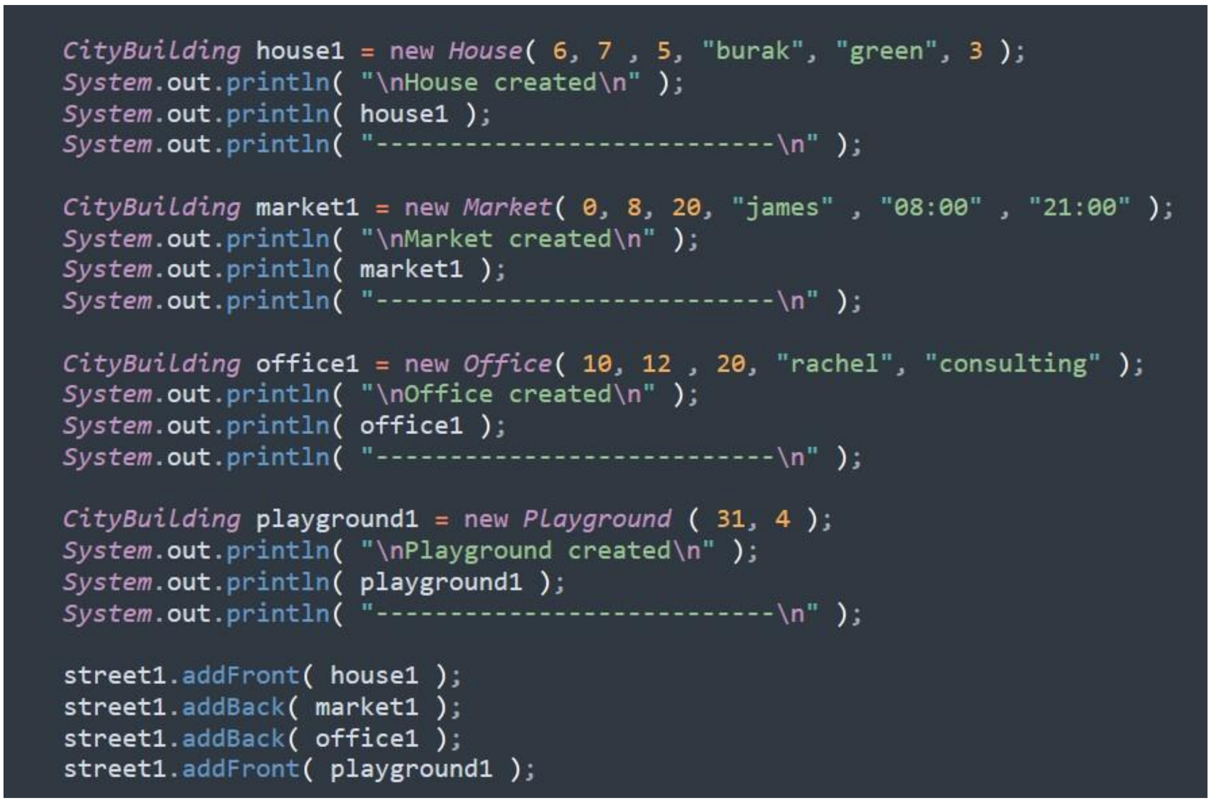
4. TEST CASES

Create a street object use toString(), getStreetLength() methods.



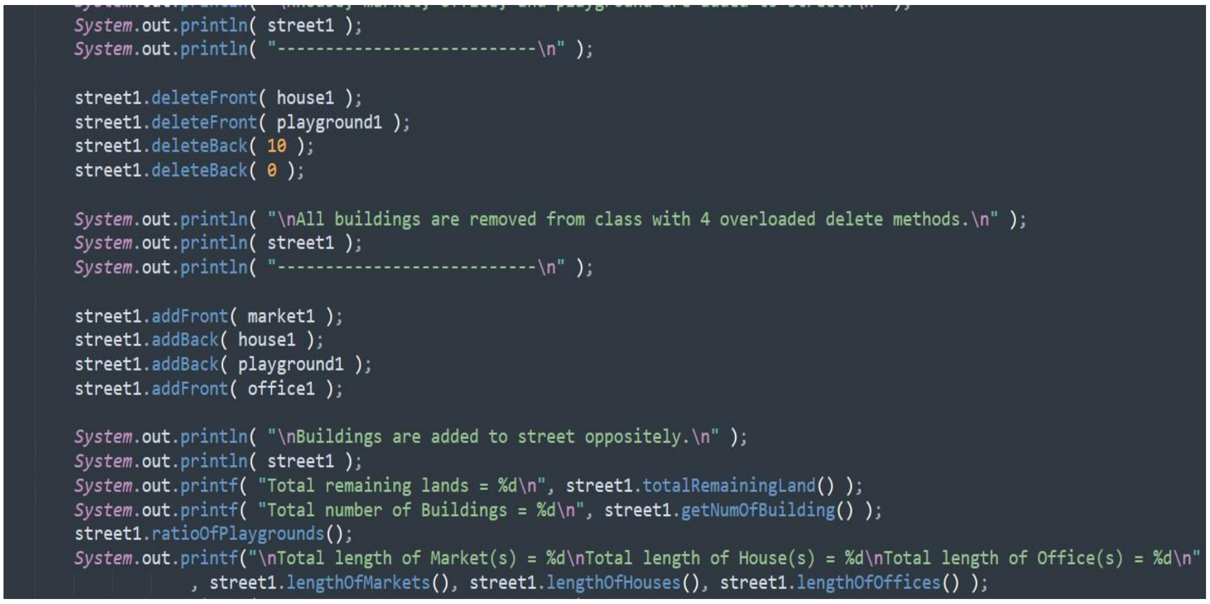
```java
CityStreet street1 = new CityStreet( 55 );

System.out.println( "\nEmpty Street created.\n" );
System.out.printf( "Length of Street is = %d", street1.getLengthOfStreet() );

System.out.println( street1 );
```

Create House, Office, Market, and Playground object. Use their toString() methods and add them to street object.



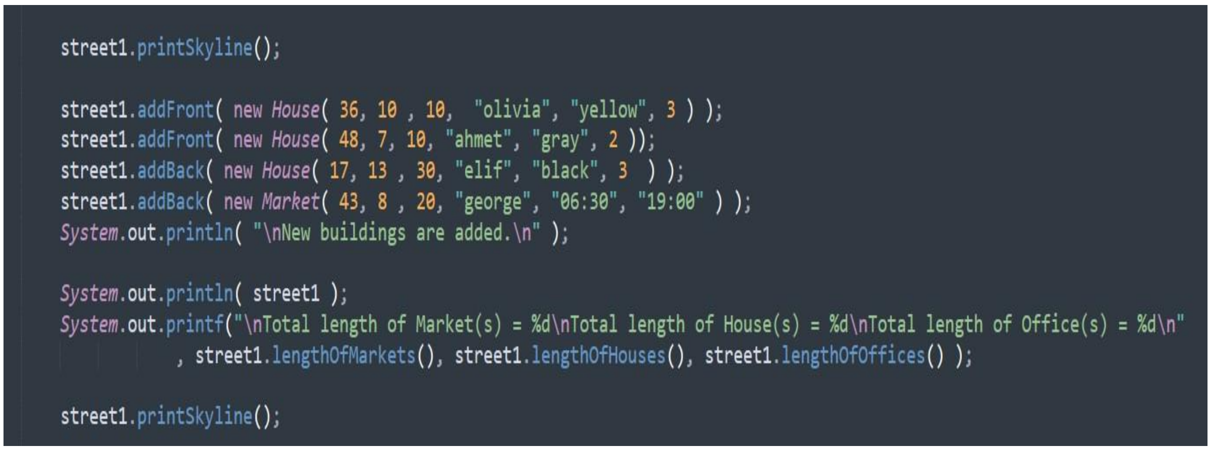
```java
CityBuilding house1 = new House( 6, 7 , 5, "burak", "green", 3 );
System.out.println( "\nHouse created\n" );
System.out.println( house1 );
System.out.println( "--------------------------\n" );

CityBuilding market1 = new Market( 0, 8, 20, "james" , "08:00" , "21:00" );
System.out.println( "\nMarket created\n" );
System.out.println( market1 );
System.out.println( "--------------------------\n" );

CityBuilding office1 = new Office( 10, 12 , 20, "rachel", "consulting" );
System.out.println( "\nOffice created\n" );
System.out.println( office1 );
System.out.println( "--------------------------\n" );

CityBuilding playground1 = new Playground ( 31, 4 );
System.out.println( "\nPlayground created\n" );
System.out.println( playground1 );
System.out.println( "--------------------------\n" );

street1.addFront( house1 );
street1.addBack( market1 );
street1.addBack( office1 );
street1.addFront( playground1 );
```

Test delete methods and add methods again, print the result.



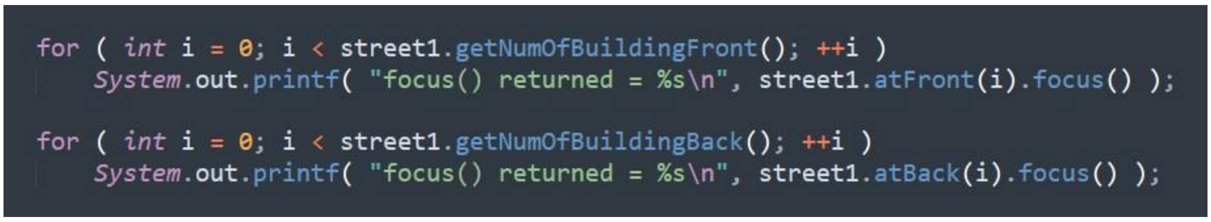
```java
    System.out.println( street1 );
    System.out.println( "--------------------------\n" );

    street1.deleteFront( house1 );
    street1.deleteFront( playground1 );
    street1.deleteBack( 10 );
    street1.deleteBack( 0 );

    System.out.println( "\nAll buildings are removed from class with 4 overloaded delete methods.\n" );
    System.out.println( street1 );
    System.out.println( "--------------------------\n" );

    street1.addFront( market1 );
    street1.addBack( house1 );
    street1.addBack( playground1 );
    street1.addFront( office1 );

    System.out.println( "\nBuildings are added to street oppositely.\n" );
    System.out.println( street1 );
    System.out.printf( "Total remaining lands = %d\n", street1.totalRemainingLand() );
    System.out.printf( "Total number of Buildings = %d\n", street1.getNumOfBuilding() );
    street1.ratioOfPlaygrounds();
    System.out.printf("\nTotal length of Market(s) = %d\nTotal length of House(s) = %d\nTotal length of Office(s) = %d\n"
            , street1.lengthOfMarkets(), street1.lengthOfHouses(), street1.lengthOfOffices() );
```

Print Skyline silhouette, and call some methods of street object.



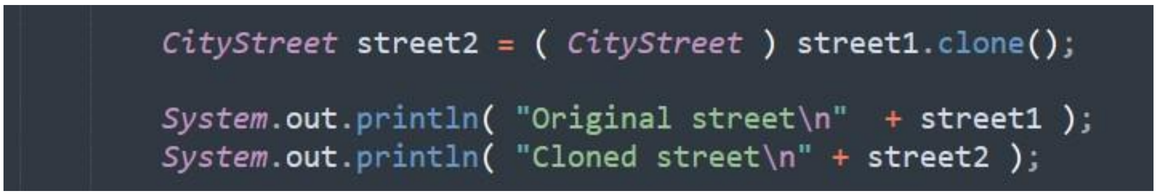
```java
    street1.printSkyline();

    street1.addFront( new House( 36, 10 , 10,  "olivia", "yellow", 3 ) );
    street1.addFront( new House( 48, 7, 10, "ahmet", "gray", 2 ));
    street1.addBack( new House( 17, 13 , 30, "elif", "black", 3  ) );
    street1.addBack( new Market( 43, 8 , 20, "george", "06:30", "19:00" ) );
    System.out.println( "\nNew buildings are added.\n" );

    System.out.println( street1 );
    System.out.printf("\nTotal length of Market(s) = %d\nTotal length of House(s) = %d\nTotal length of Office(s) = %d\n"
            , street1.lengthOfMarkets(), street1.lengthOfHouses(), street1.lengthOfOffices() );

    street1.printSkyline();
```

Make polymorphic call for focus() method, test at methods and some get methods also.



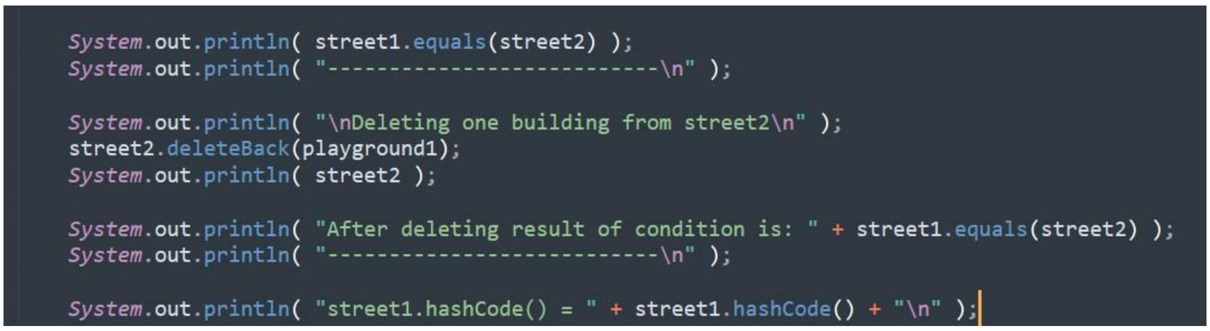
```java
for ( int i = 0; i < street1.getNumOfBuildingFront(); ++i )
    System.out.printf( "focus() returned = %s\n", street1.atFront(i).focus() );

for ( int i = 0; i < street1.getNumOfBuildingBack(); ++i )
    System.out.printf( "focus() returned = %s\n", street1.atBack(i).focus() );
```

Test clone() method, and print cloned object.



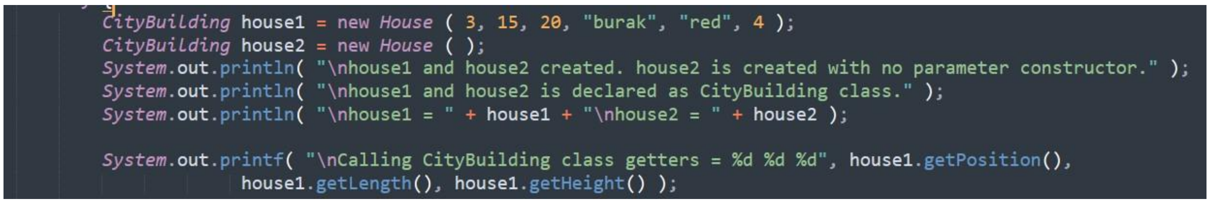
```java
CityStreet street2 = ( CityStreet ) street1.clone();

System.out.println( "Original street\n"  + street1 );
System.out.println( "Cloned street\n" + street2 );
```

Test equals(), hashCode() methods of street object.



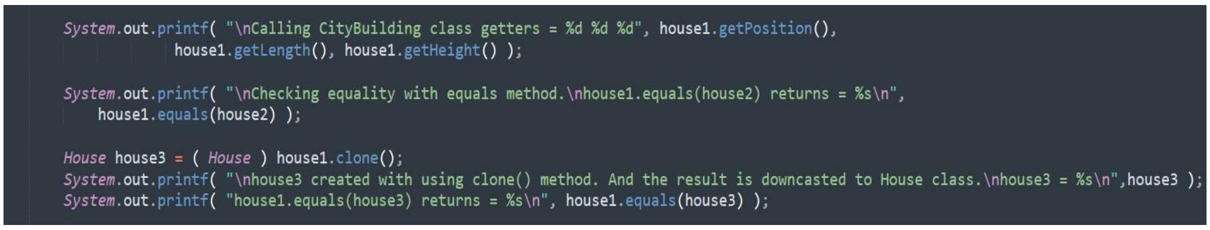
```java
System.out.println( street1.equals(street2) );
System.out.println( "--------------------------\n" );

System.out.println( "\nDeleting one building from street2\n" );
street2.deleteBack(playground1);
System.out.println( street2 );

System.out.println( "After deleting result of condition is: " + street1.equals(street2) );
System.out.println( "--------------------------\n" );

System.out.println( "street1.hashCode() = " + street1.hashCode() + "\n" );
```

Test constructors, toString(), and getters for House.



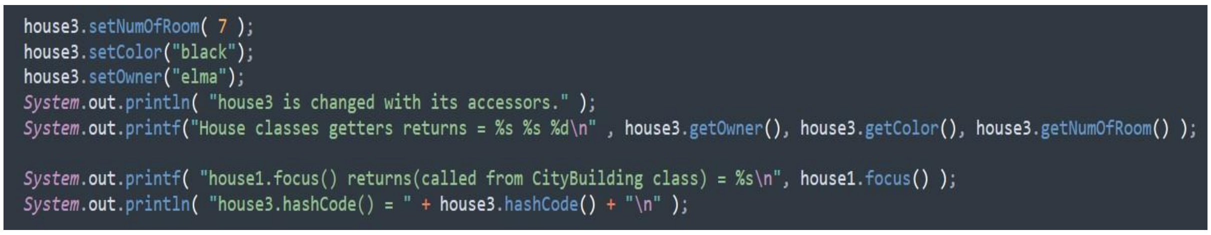
```java
CityBuilding house1 = new House ( 3, 15, 20, "burak", "red", 4 );
CityBuilding house2 = new House ( );
System.out.println( "\nhouse1 and house2 created. house2 is created with no parameter constructor." );
System.out.println( "\nhouse1 and house2 is declared as CityBuilding class." );
System.out.println( "\nhouse1 = " + house1 + "\nhouse2 = " + house2 );

System.out.printf( "\nCalling CityBuilding class getters = %d %d %d", house1.getPosition(),
            house1.getLength(), house1.getHeight() );
```

Test equals() and clone() methods for House.



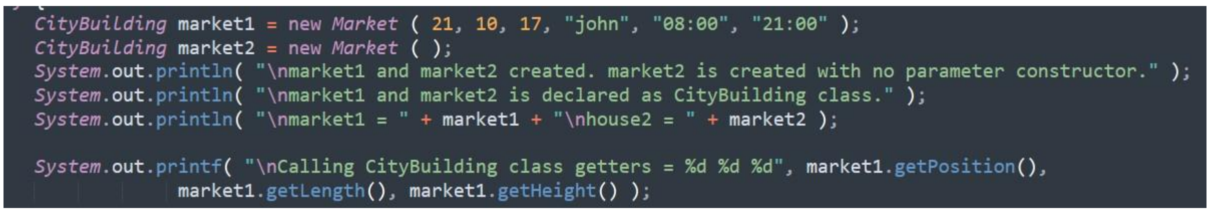
```java
System.out.printf( "\nCalling CityBuilding class getters = %d %d %d", house1.getPosition(),
            house1.getLength(), house1.getHeight() );

System.out.printf( "\nChecking equality with equals method.\nhouse1.equals(house2) returns = %s\n",
    house1.equals(house2) );

House house3 = ( House ) house1.clone();
System.out.printf( "\nhouse3 created with using clone() method. And the result is downcasted to House class.\nhouse3 = %s\n",house3 );
System.out.printf( "house1.equals(house3) returns = %s\n", house1.equals(house3) );
```

Test setters, getters, focus()(Called from base), and hashCode() for House.



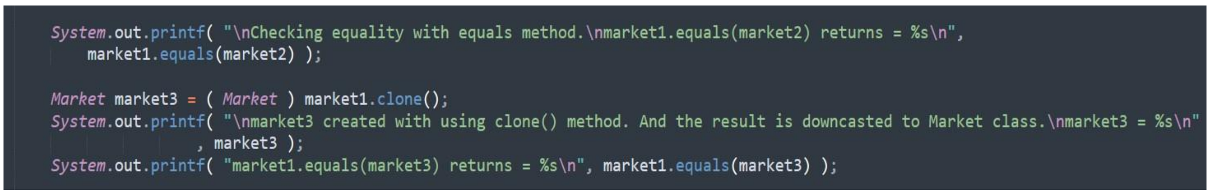
```java
house3.setNumOfRoom( 7 );
house3.setColor("black");
house3.setOwner("elma");
System.out.println( "house3 is changed with its accessors." );
System.out.printf("House classes getters returns = %s %s %d\n" , house3.getOwner(), house3.getColor(), house3.getNumOfRoom() );

System.out.printf( "house1.focus() returns(called from CityBuilding class) = %s\n", house1.focus() );
System.out.println( "house3.hashCode() = " + house3.hashCode() + "\n" );
```

Test constructors, toString(), and getters(CityBuilding) of Market.



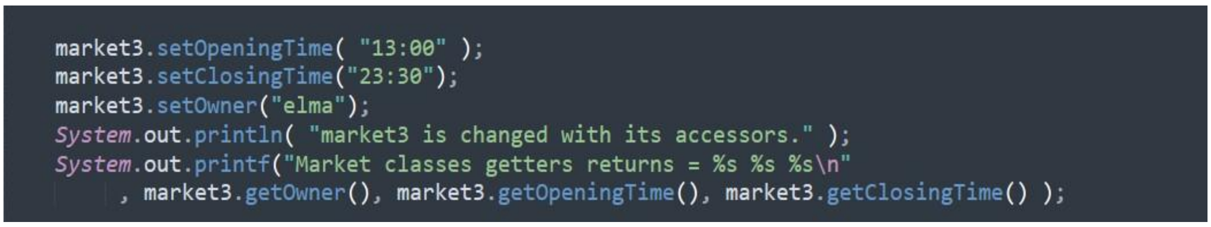
```java
    CityBuilding market1 = new Market ( 21, 10, 17, "john", "08:00", "21:00" );
    CityBuilding market2 = new Market ( );
    System.out.println( "\nmarket1 and market2 created. market2 is created with no parameter constructor." );
    System.out.println( "\nmarket1 and market2 is declared as CityBuilding class." );
    System.out.println( "\nmarket1 = " + market1 + "\nhouse2 = " + market2 );

    System.out.printf( "\nCalling CityBuilding class getters = %d %d %d", market1.getPosition(),
                    market1.getLength(), market1.getHeight() );
```

Test clone(), equals() of Market.



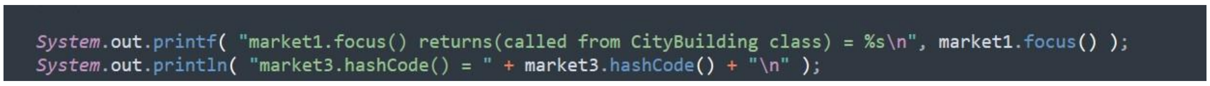
```java
    System.out.printf( "\nChecking equality with equals method.\nmarket1.equals(market2) returns = %s\n",
        market1.equals(market2) );

    Market market3 = ( Market ) market1.clone();
    System.out.printf( "\nmarket3 created with using clone() method. And the result is downcasted to Market class.\nmarket3 = %s\n"
                    , market3 );
    System.out.printf( "market1.equals(market3) returns = %s\n", market1.equals(market3) );
```

Test setters and getters of Market.



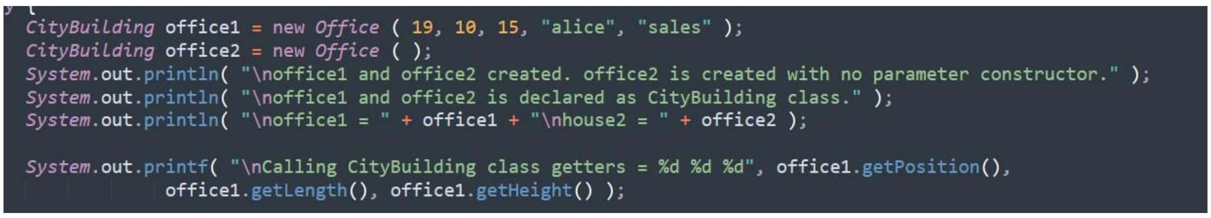
```java
    market3.setOpeningTime( "13:00" );
    market3.setClosingTime("23:30");
    market3.setOwner("elma");
    System.out.println( "market3 is changed with its accessors." );
    System.out.printf("Market classes getters returns = %s %s %s\n"
        , market3.getOwner(), market3.getOpeningTime(), market3.getClosingTime() );
```

Test focus(), hashCode() of Market.



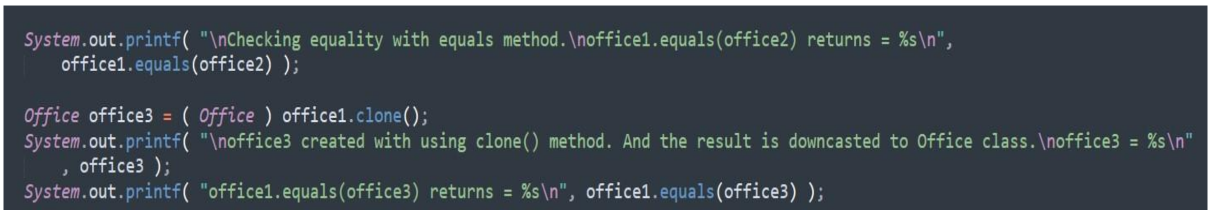
```java
    System.out.printf( "market1.focus() returns(called from CityBuilding class) = %s\n", market1.focus() );
    System.out.println( "market3.hashCode() = " + market3.hashCode() + "\n" );
```

Test constructors, getters, and toString() of Office.



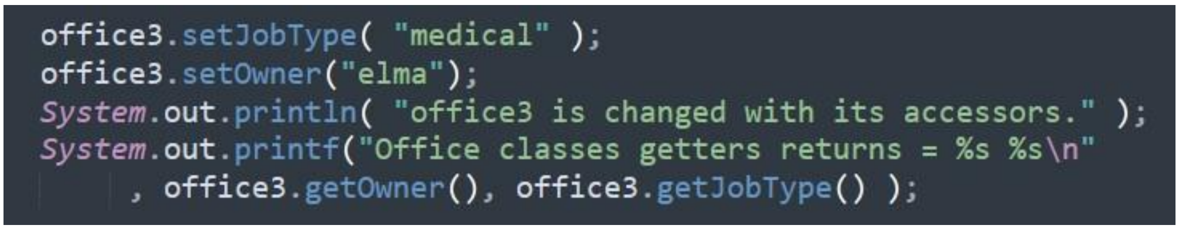
```java
    CityBuilding office1 = new Office ( 19, 10, 15, "alice", "sales" );
    CityBuilding office2 = new Office ( );
    System.out.println( "\noffice1 and office2 created. office2 is created with no parameter constructor." );
    System.out.println( "\noffice1 and office2 is declared as CityBuilding class." );
    System.out.println( "\noffice1 = " + office1 + "\nhouse2 = " + office2 );

    System.out.printf( "\nCalling CityBuilding class getters = %d %d %d", office1.getPosition(),
                    office1.getLength(), office1.getHeight() );
```

Test clone(), and equals() of Office.



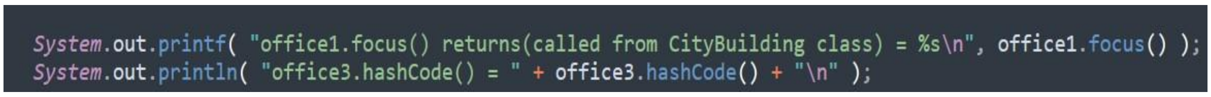
```java
    System.out.printf( "\nChecking equality with equals method.\noffice1.equals(office2) returns = %s\n",
        office1.equals(office2) );

    Office office3 = ( Office ) office1.clone();
    System.out.printf( "\noffice3 created with using clone() method. And the result is downcasted to Office class.\noffice3 = %s\n"
        , office3 );
    System.out.printf( "office1.equals(office3) returns = %s\n", office1.equals(office3) );
```

Test setters and getters of Office.



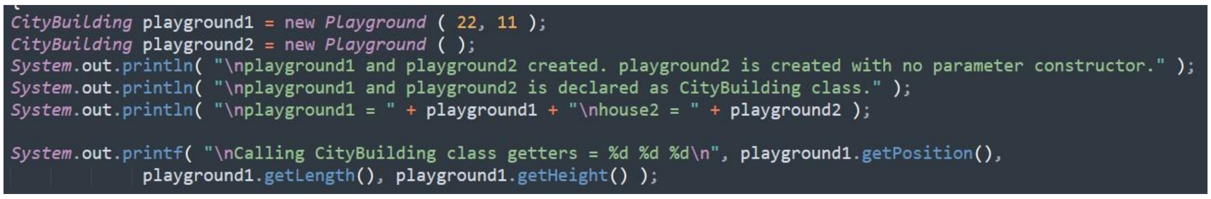
```
office3.setJobType( "medical" );
office3.setOwner("elma");
System.out.println( "office3 is changed with its accessors." );
System.out.printf("Office classes getters returns = %s %s\n"
        , office3.getOwner(), office3.getJobType() );
```

Test focus(), and hashCode() of Office.



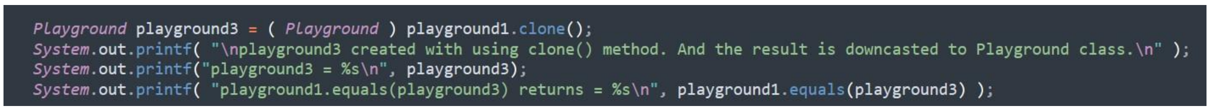
```
System.out.printf( "office1.focus() returns(called from CityBuilding class) = %s\n", office1.focus() );
System.out.println( "office3.hashCode() = " + office3.hashCode() + "\n" );
```

Test constructors, and getters of Playground.



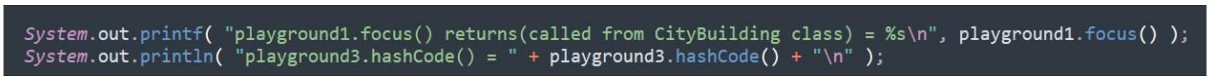
```
CityBuilding playground1 = new Playground ( 22, 11 );
CityBuilding playground2 = new Playground ( );
System.out.println( "\nplayground1 and playground2 created. playground2 is created with no parameter constructor." );
System.out.println( "\nplayground1 and playground2 is declared as CityBuilding class." );
System.out.println( "\nplayground1 = " + playground1 + "\nhouse2 = " + playground2 );

System.out.printf( "\nCalling CityBuilding class getters = %d %d %d\n", playground1.getPosition(),
            playground1.getLength(), playground1.getHeight() );
```

Test clone(), and equals() method of Playground.



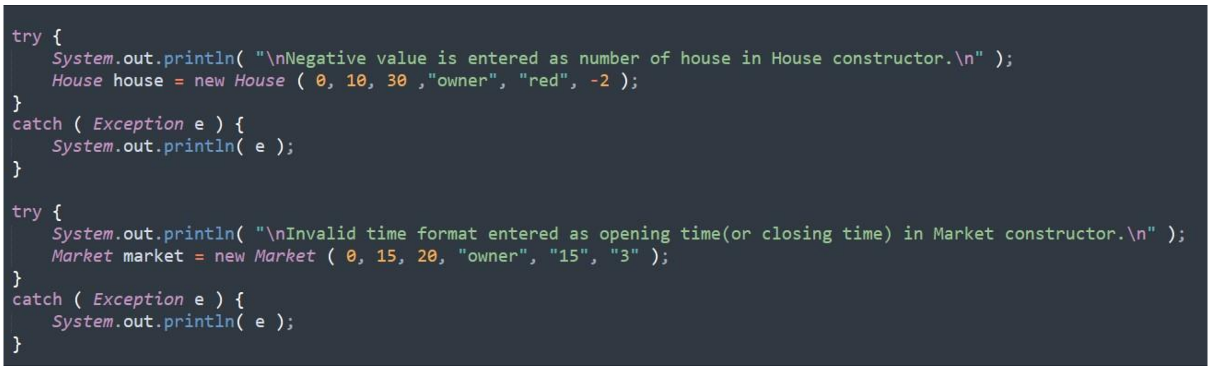
```
Playground playground3 = ( Playground ) playground1.clone();
System.out.printf( "\nplayground3 created with using clone() method. And the result is downcasted to Playground class.\n" );
System.out.printf("playground3 = %s\n", playground3);
System.out.printf( "playground1.equals(playground3) returns = %s\n", playground1.equals(playground3) );
```

Test focus(), and hashCode() method of Playground.



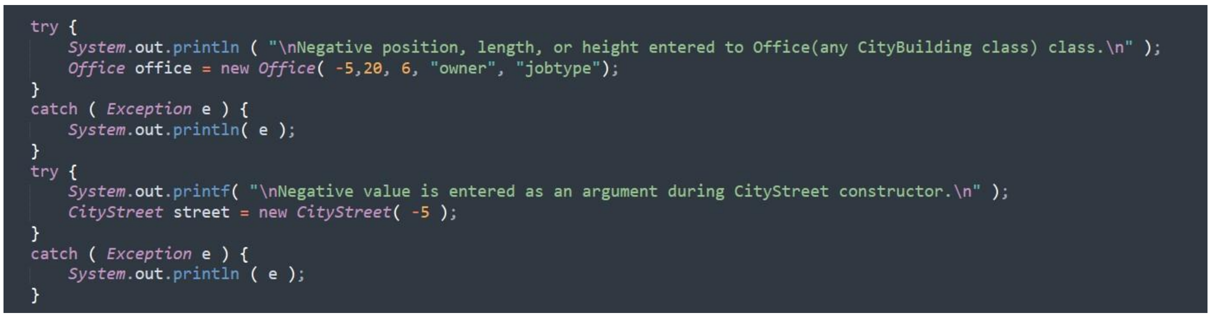
```
System.out.printf( "playground1.focus() returns(called from CityBuilding class) = %s\n", playground1.focus() );
System.out.println( "playground3.hashCode() = " + playground3.hashCode() + "\n" );
```

Test spesific exceptions of House and Market.



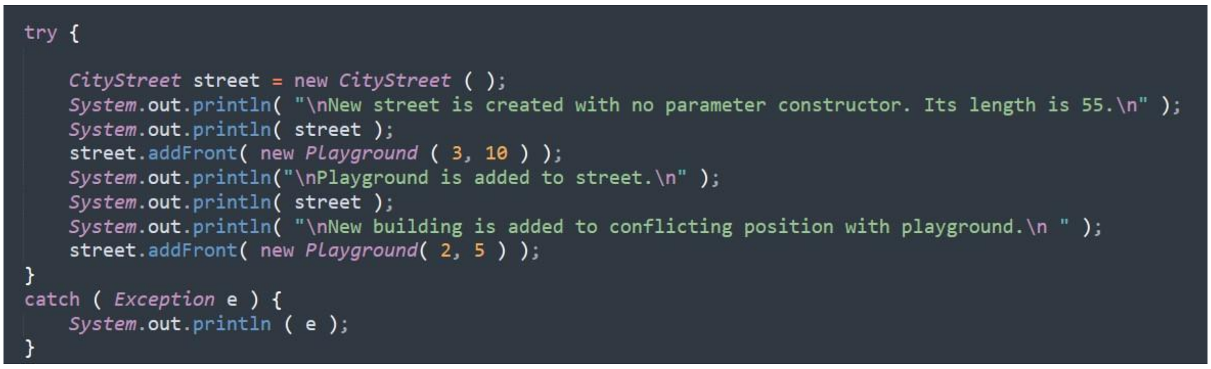
```
try {
    System.out.println( "\nNegative value is entered as number of house in House constructor.\n" );
    House house = new House ( 0, 10, 30 ,"owner", "red", -2 );
}
catch ( Exception e ) {
    System.out.println( e );
}

try {
    System.out.println( "\nInvalid time format entered as opening time(or closing time) in Market constructor.\n" );
    Market market = new Market ( 0, 15, 20, "owner", "15", "3" );
}
catch ( Exception e ) {
    System.out.println( e );
}
```

Test exception after negative input entered for Office and CityStreet.



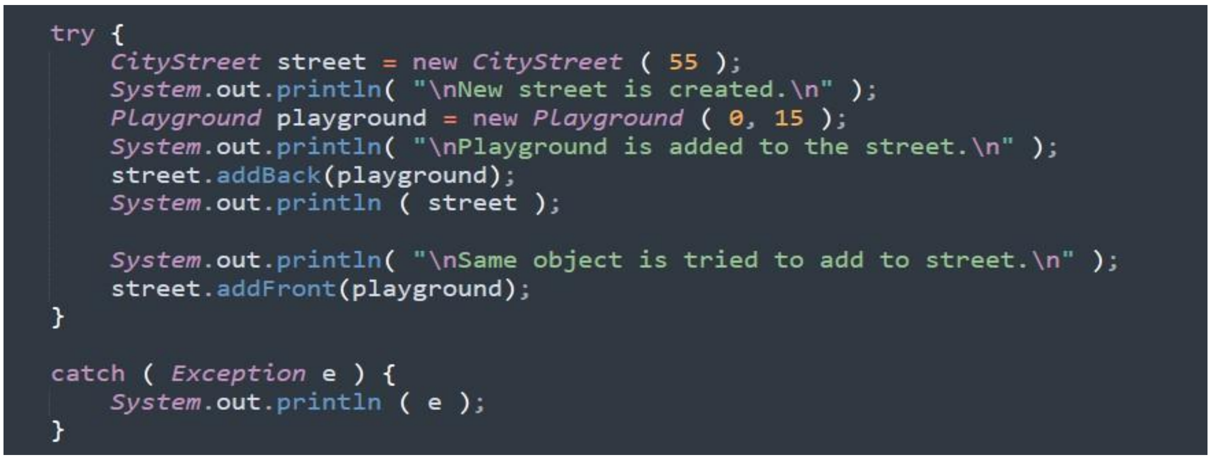
```
try {
    System.out.println ( "\nNegative position, length, or height entered to Office(any CityBuilding class) class.\n" );
    Office office = new Office( -5,20, 6, "owner", "jobtype");
}
catch ( Exception e ) {
    System.out.println( e );
}
try {
    System.out.printf( "\nNegative value is entered as an argument during CityStreet constructor.\n" );
    CityStreet street = new CityStreet( -5 );
}
catch ( Exception e ) {
    System.out.println ( e );
}
```

Test exception when a building is tried to add which has already occupied position.



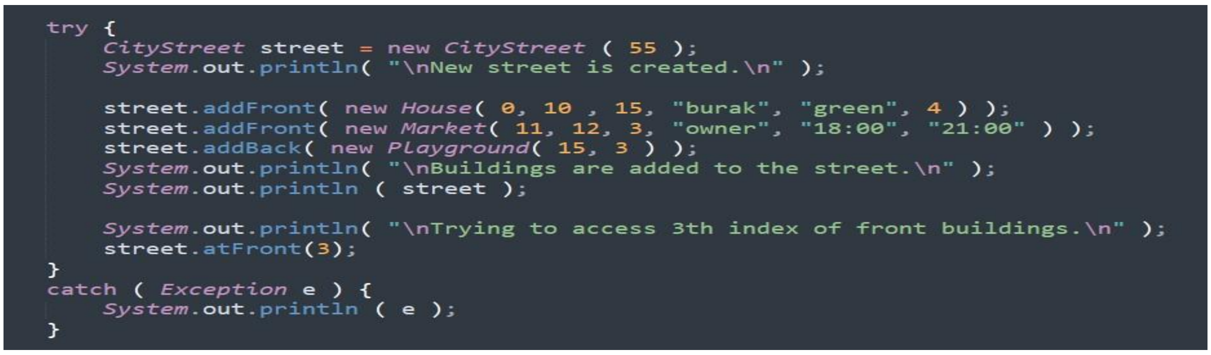
```
try {

    CityStreet street = new CityStreet ( );
    System.out.println( "\nNew street is created with no parameter constructor. Its length is 55.\n" );
    System.out.println( street );
    street.addFront( new Playground ( 3, 10 ) );
    System.out.println("\nPlayground is added to street.\n" );
    System.out.println( street );
    System.out.println( "\nNew building is added to conflicting position with playground.\n " );
    street.addFront( new Playground( 2, 5 ) );
}
catch ( Exception e ) {
    System.out.println ( e );
}
```

Test exception when same object is tried to add the street.



```
try {
    CityStreet street = new CityStreet ( 55 );
    System.out.println( "\nNew street is created.\n" );
    Playground playground = new Playground ( 0, 15 );
    System.out.println( "\nPlayground is added to the street.\n" );
    street.addBack(playground);
    System.out.println ( street );

    System.out.println( "\nSame object is tried to add to street.\n" );
    street.addFront(playground);
}

catch ( Exception e ) {
    System.out.println ( e );
}
```

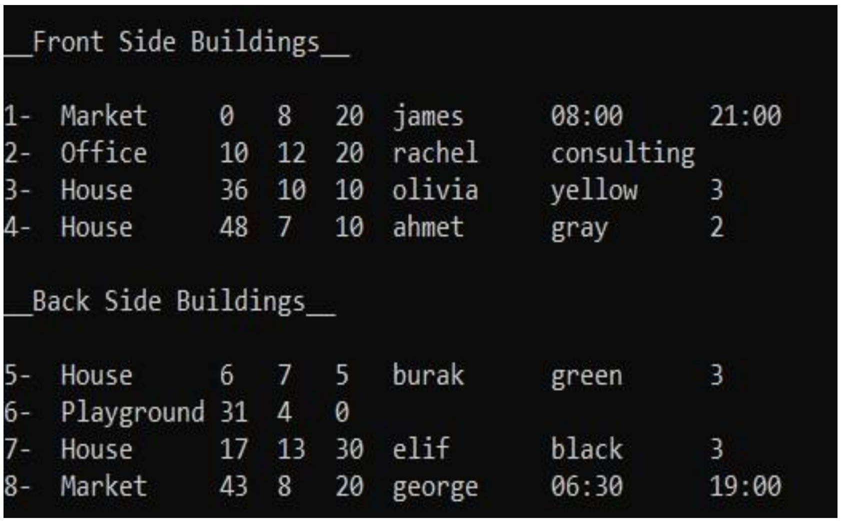
Test exception when invalid array index is tried to be accessed.



```
try {
    CityStreet street = new CityStreet ( 55 );
    System.out.println( "\nNew street is created.\n" );

    street.addFront( new House( 0, 10 , 15, "burak", "green", 4 ) );
    street.addFront( new Market( 11, 12, 3, "owner", "18:00", "21:00" ) );
    street.addBack( new Playground( 15, 3 ) );
    System.out.println( "\nBuildings are added to the street.\n" );
    System.out.println ( street );

    System.out.println( "\nTrying to access 3th index of front buildings.\n" );
    street.atFront(3);
}
catch ( Exception e ) {
    System.out.println ( e );
}
```

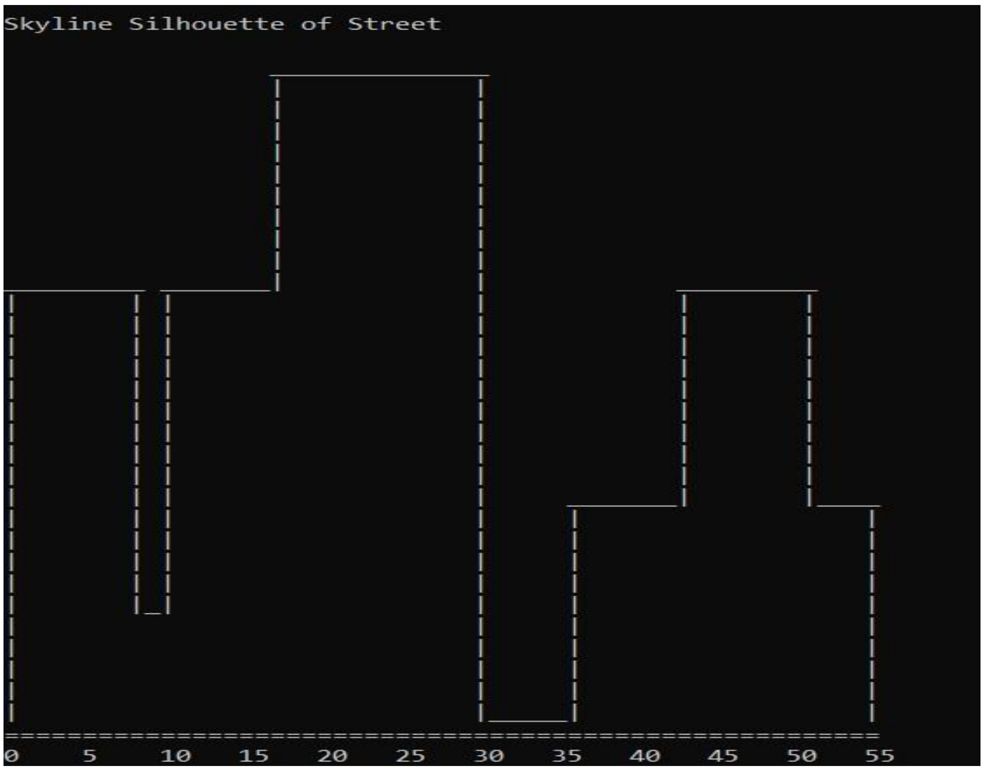
5. RUNNING AND RESULTS

Sample Street



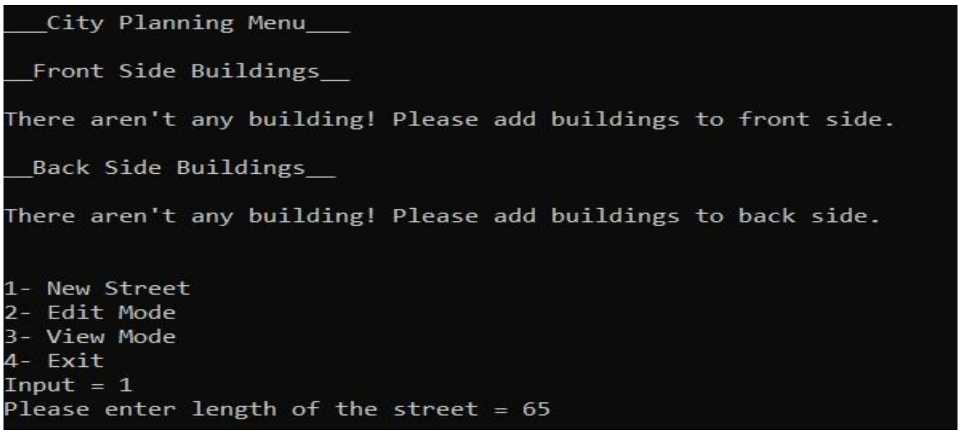
```
__Front Side Buildings__

1-  Market     0   8   20  james      08:00      21:00
2-  Office     10  12  20  rachel     consulting
3-  House      36  10  10  olivia     yellow     3
4-  House      48  7   10  ahmet      gray       2

__Back Side Buildings__

5-  House      6   7   5   burak      green      3
6-  Playground 31  4   0
7-  House      17  13  30  elif       black      3
8-  Market     43  8   20  george     06:30      19:00
```

This street's Skyline Silhouette



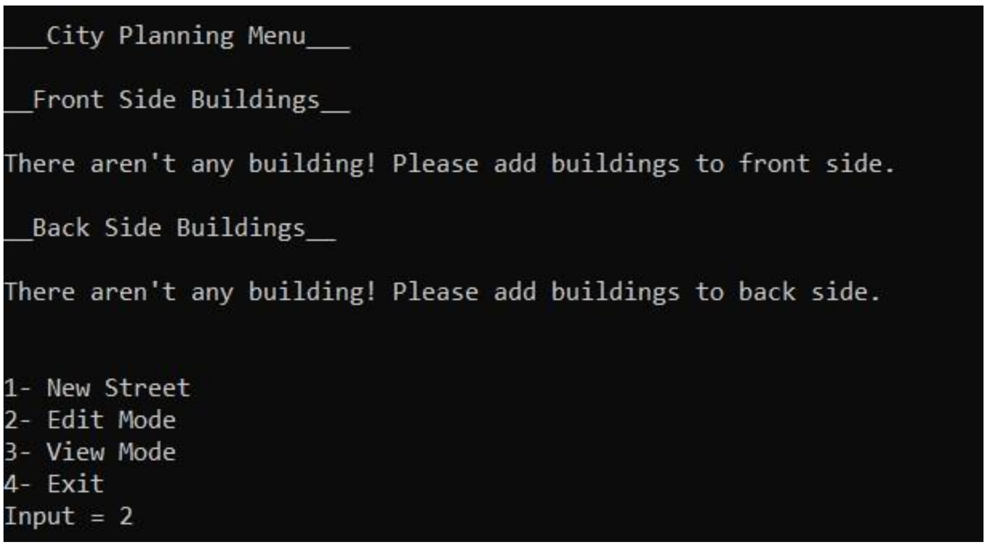
```
Skyline Silhouette of Street
```

Creating a new street



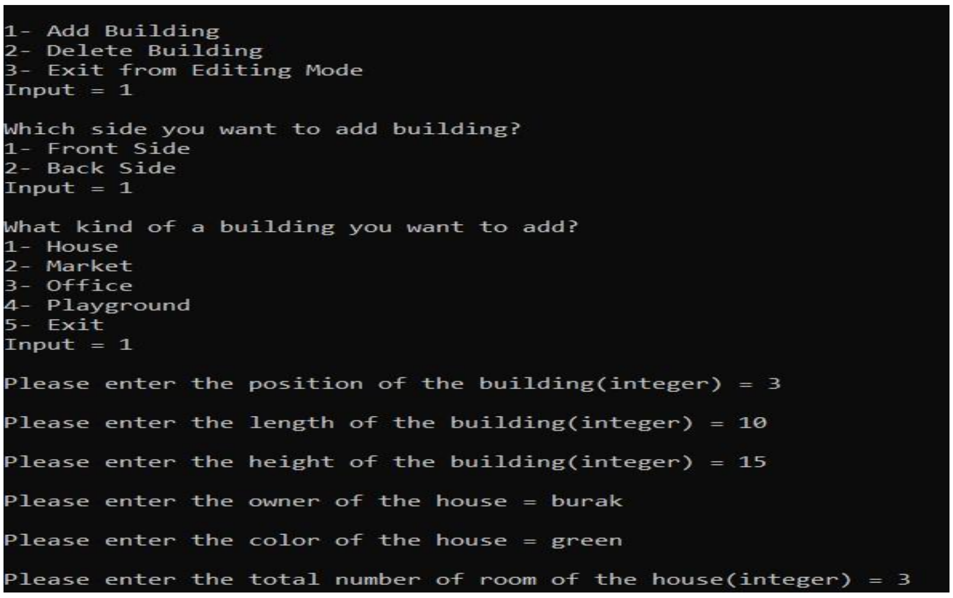
```
___City Planning Menu___

__Front Side Buildings__

There aren't any building! Please add buildings to front side.

__Back Side Buildings__

There aren't any building! Please add buildings to back side.


1- New Street
2- Edit Mode
3- View Mode
4- Exit
Input = 1
Please enter length of the street = 65
```

Opening edit mode of street



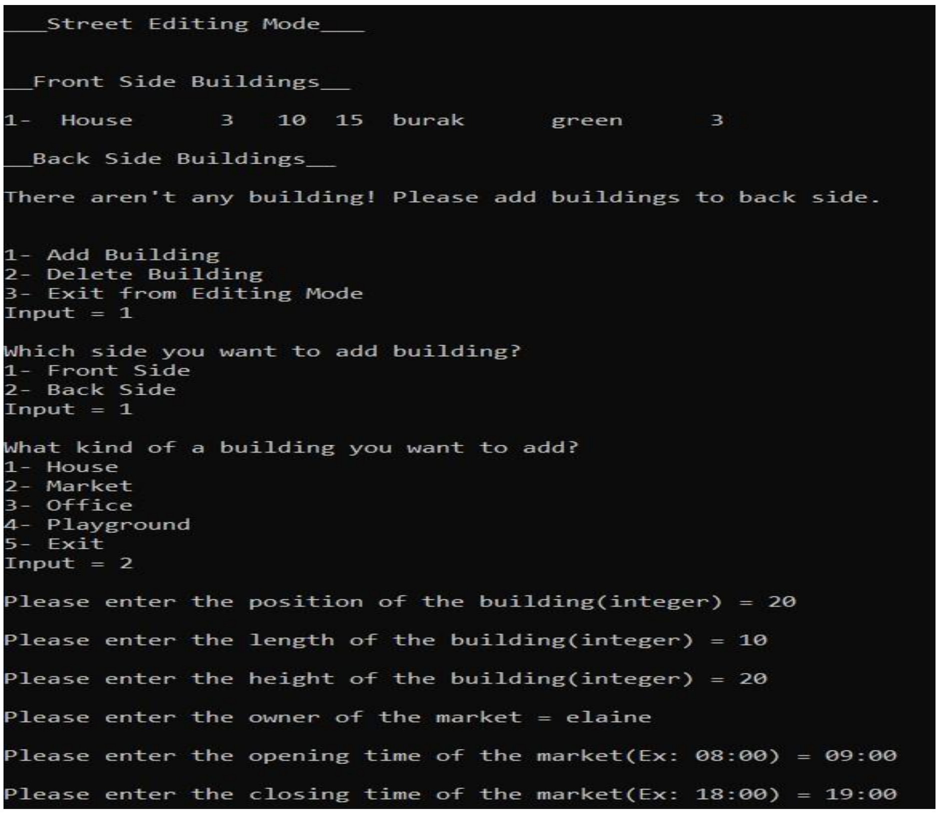
```
___City Planning Menu___

__Front Side Buildings__

There aren't any building! Please add buildings to front side.

__Back Side Buildings__

There aren't any building! Please add buildings to back side.


1- New Street
2- Edit Mode
3- View Mode
4- Exit
Input = 2
```

Add house to street



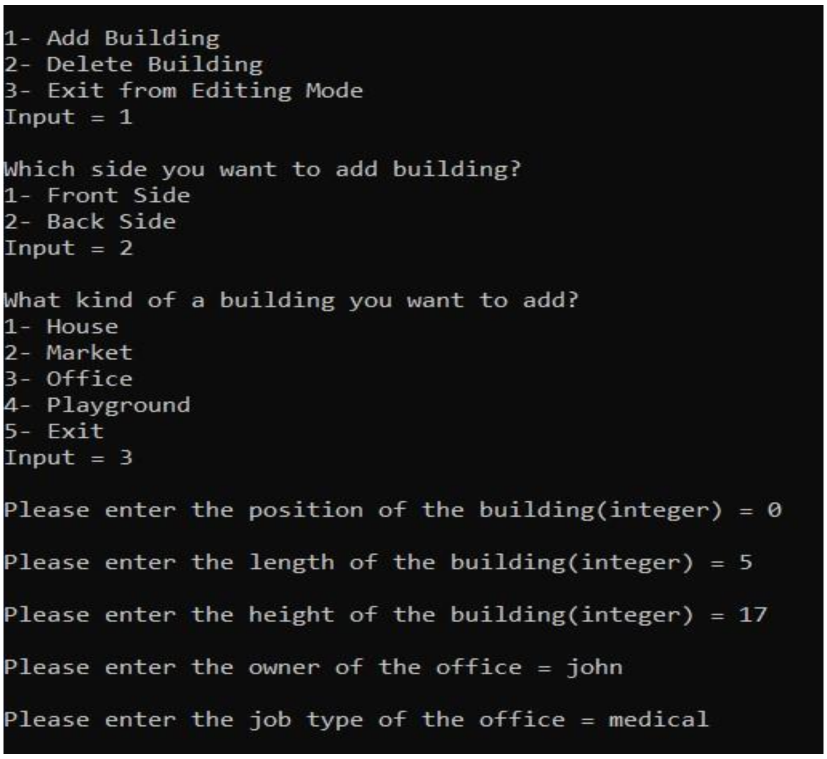
```
1- Add Building
2- Delete Building
3- Exit from Editing Mode
Input = 1

Which side you want to add building?
1- Front Side
2- Back Side
Input = 1

What kind of a building you want to add?
1- House
2- Market
3- Office
4- Playground
5- Exit
Input = 1

Please enter the position of the building(integer) = 3

Please enter the length of the building(integer) = 10

Please enter the height of the building(integer) = 15

Please enter the owner of the house = burak

Please enter the color of the house = green

Please enter the total number of room of the house(integer) = 3
```

Add market to street



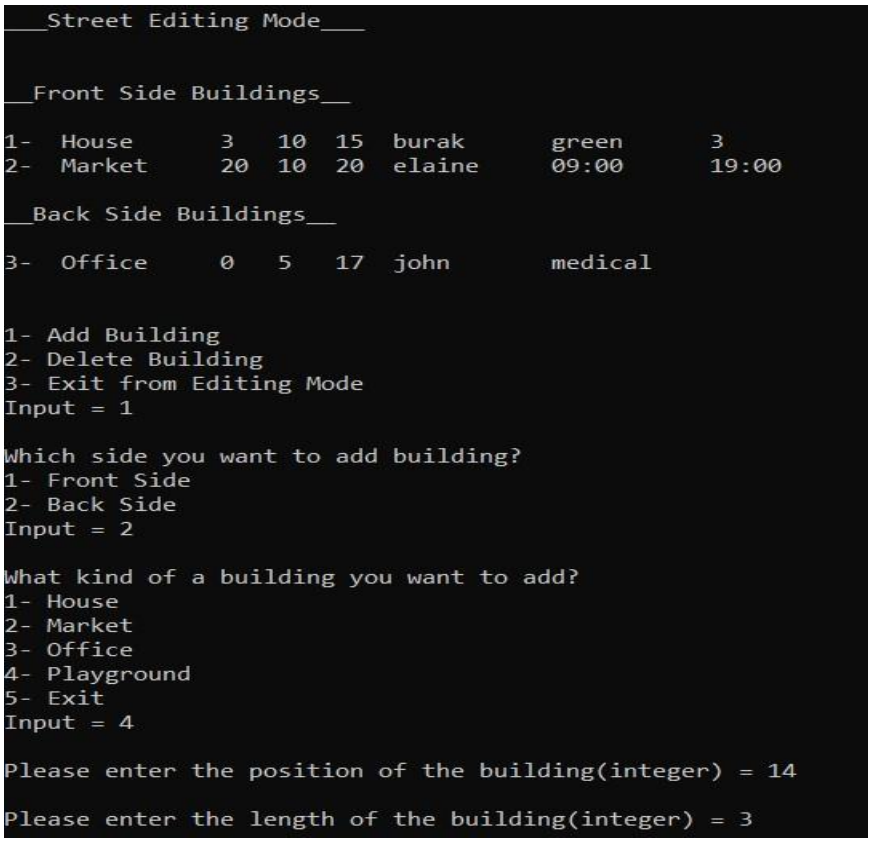
```
___Street Editing Mode___

__Front Side Buildings__

1-  House      3   10  15  burak      green      3

__Back Side Buildings__

There aren't any building! Please add buildings to back side.


1- Add Building
2- Delete Building
3- Exit from Editing Mode
Input = 1

Which side you want to add building?
1- Front Side
2- Back Side
Input = 1

What kind of a building you want to add?
1- House
2- Market
3- Office
4- Playground
5- Exit
Input = 2

Please enter the position of the building(integer) = 20

Please enter the length of the building(integer) = 10

Please enter the height of the building(integer) = 20

Please enter the owner of the market = elaine

Please enter the opening time of the market(Ex: 08:00) = 09:00

Please enter the closing time of the market(Ex: 18:00) = 19:00
```

Add office to street

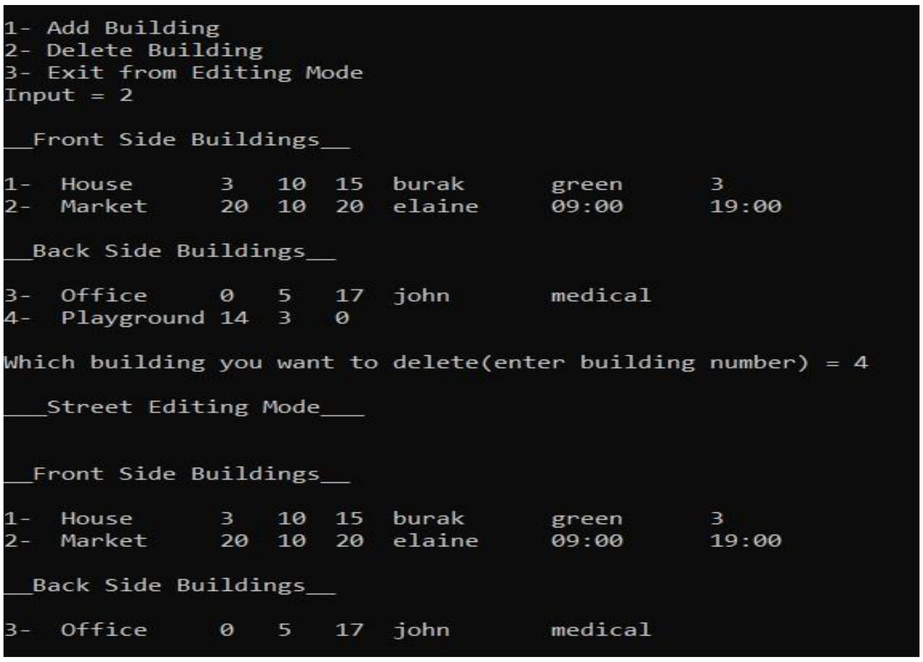


```
1- Add Building
2- Delete Building
3- Exit from Editing Mode
Input = 1

Which side you want to add building?
1- Front Side
2- Back Side
Input = 2

What kind of a building you want to add?
1- House
2- Market
3- Office
4- Playground
5- Exit
Input = 3

Please enter the position of the building(integer) = 0

Please enter the length of the building(integer) = 5

Please enter the height of the building(integer) = 17

Please enter the owner of the office = john

Please enter the job type of the office = medical
```

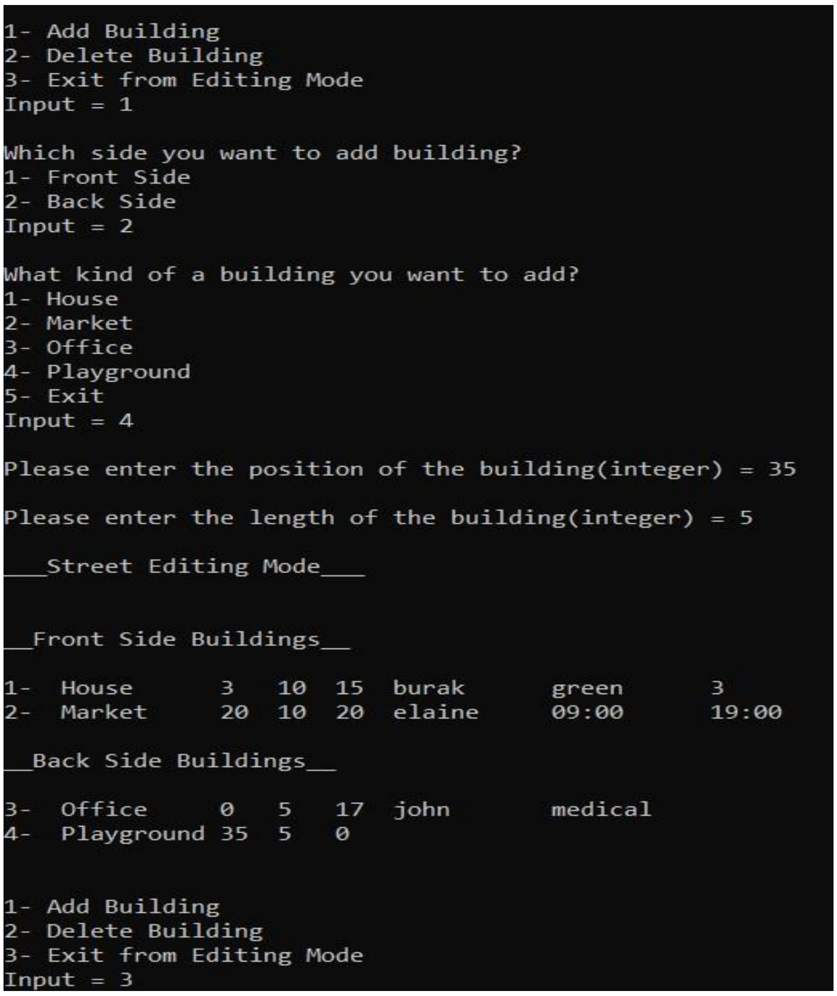
Add playground to street


```
___Street Editing Mode___


__Front Side Buildings__

1-   House       3    10   15   burak      green      3
2-   Market      20   10   20   elaine     09:00      19:00

__Back Side Buildings__

3-   Office      0    5    17   john       medical


1- Add Building
2- Delete Building
3- Exit from Editing Mode
Input = 1

Which side you want to add building?
1- Front Side
2- Back Side
Input = 2

What kind of a building you want to add?
1- House
2- Market
3- Office
4- Playground
5- Exit
Input = 4

Please enter the position of the building(integer) = 14

Please enter the length of the building(integer) = 3
```

Delete one building from street

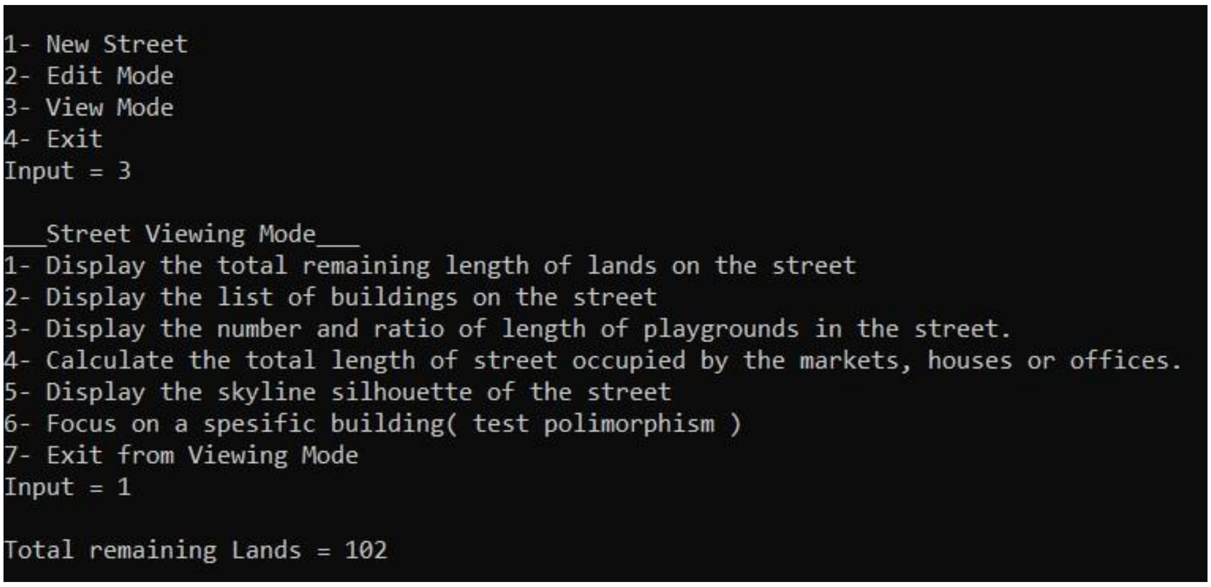

```
1- Add Building
2- Delete Building
3- Exit from Editing Mode
Input = 2

__Front Side Buildings__

1-   House       3    10   15   burak      green      3
2-   Market      20   10   20   elaine     09:00      19:00

__Back Side Buildings__

3-   Office      0    5    17   john       medical
4-   Playground 14    3    0

Which building you want to delete(enter building number) = 4

___Street Editing Mode___


__Front Side Buildings__

1-   House       3    10   15   burak      green      3
2-   Market      20   10   20   elaine     09:00      19:00

__Back Side Buildings__

3-   Office      0    5    17   john       medical
```

Add playground again



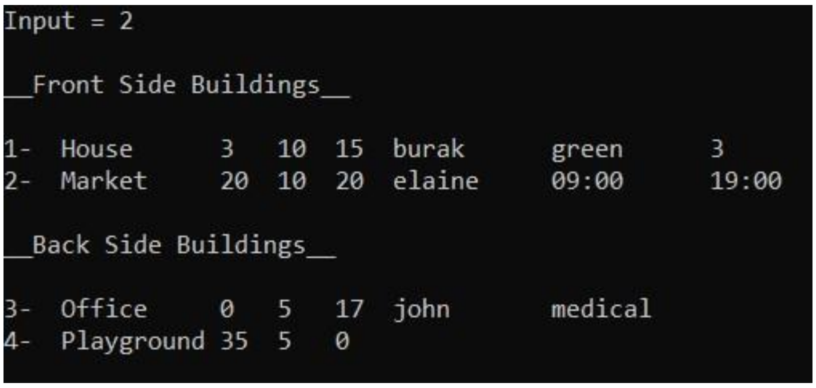
```
1- Add Building
2- Delete Building
3- Exit from Editing Mode
Input = 1

Which side you want to add building?
1- Front Side
2- Back Side
Input = 2

What kind of a building you want to add?
1- House
2- Market
3- Office
4- Playground
5- Exit
Input = 4

Please enter the position of the building(integer) = 35

Please enter the length of the building(integer) = 5

___Street Editing Mode___


__Front Side Buildings__

1-  House      3   10  15  burak      green      3
2-  Market     20  10  20  elaine     09:00      19:00

__Back Side Buildings__

3-  Office     0   5   17  john       medical
4-  Playground 35  5   0


1- Add Building
2- Delete Building
3- Exit from Editing Mode
Input = 3
```

Open the view mode and display number of remaining lands



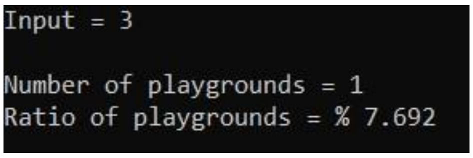
```
1- New Street
2- Edit Mode
3- View Mode
4- Exit
Input = 3

___Street Viewing Mode___
1- Display the total remaining length of lands on the street
2- Display the list of buildings on the street
3- Display the number and ratio of length of playgrounds in the street.
4- Calculate the total length of street occupied by the markets, houses or offices.
5- Display the skyline silhouette of the street
6- Focus on a spesific building( test polimorphism )
7- Exit from Viewing Mode
Input = 1

Total remaining Lands = 102
```

Display the list of buildings



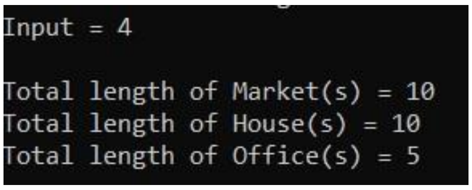
```
Input = 2

__Front Side Buildings__

1-   House      3   10  15  burak      green      3
2-   Market     20  10  20  elaine     09:00      19:00

__Back Side Buildings__

3-   Office     0   5   17  john       medical
4-   Playground 35  5   0
```

Display the number and ratio of playgrounds



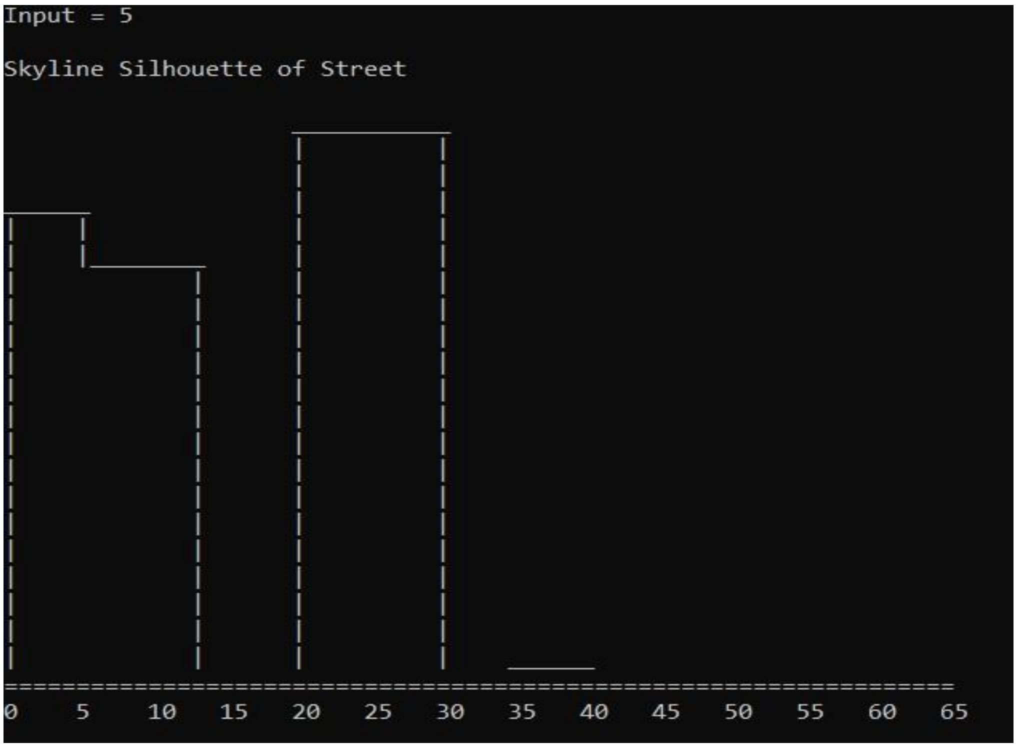
```
Input = 3

Number of playgrounds = 1
Ratio of playgrounds = % 7.692
```

After choosing 4th option



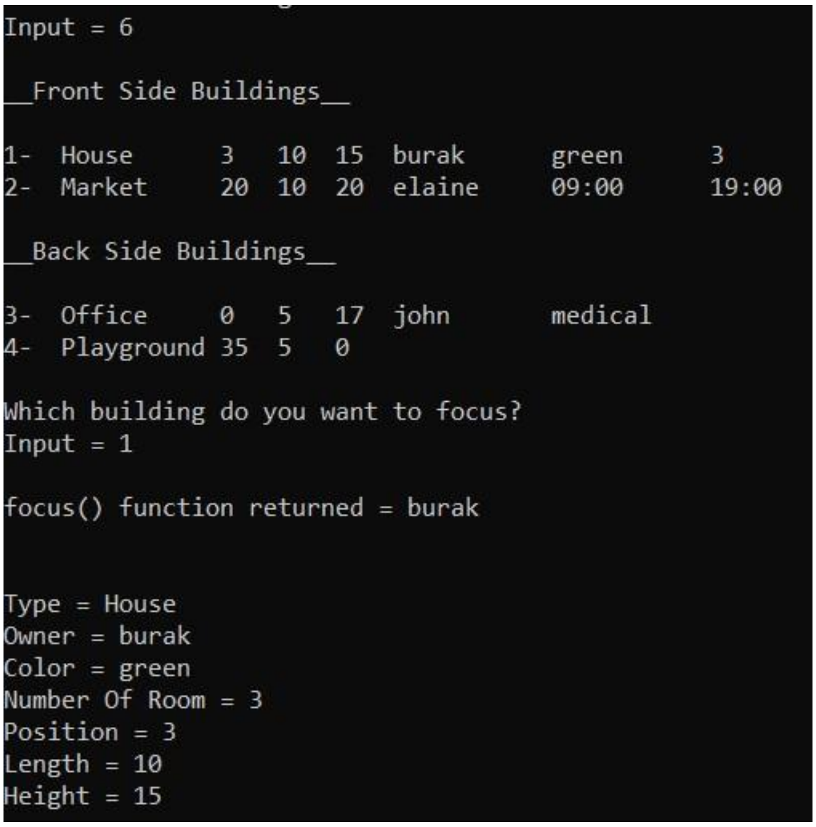
```
Input = 4

Total length of Market(s) = 10
Total length of House(s) = 10
Total length of Office(s) = 5
```

Display the skyline silhouette



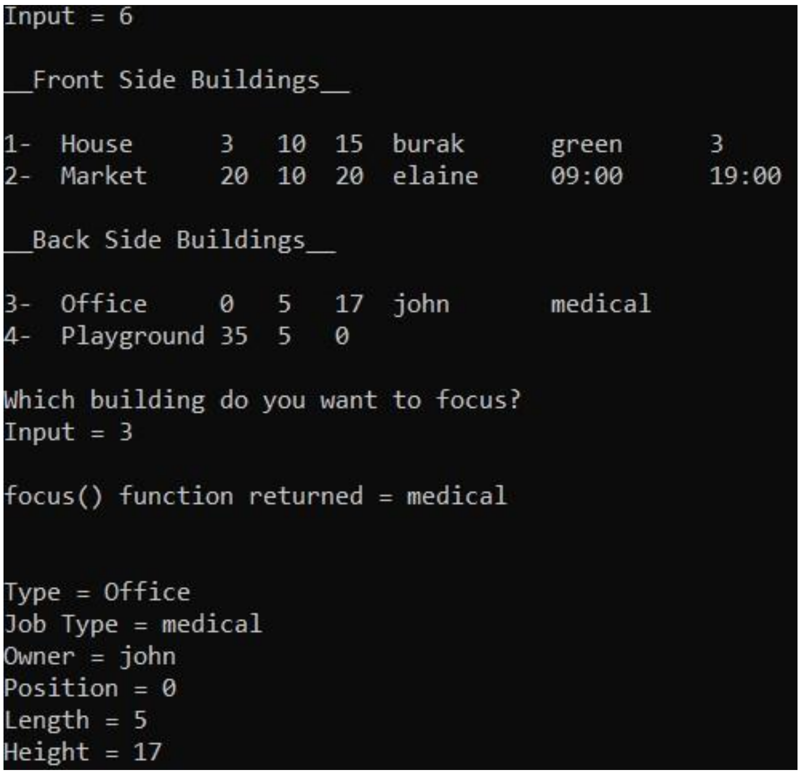
```
Input = 5

Skyline Silhouette of Street

                    ___________
                    |         |
                    |         |
______              |         |
|    |              |         |
|    |________      |         |
|            |      |         |
|            |      |         |
|            |      |         |
|            |      |         |
|            |      |         |
|            |      |         |
|            |      |         |
|            |      |         |
|            |      |         |
|            |      |         |
|            |      |         |
|            |      |         |
|            |      |         |
|            |      |         |
|            |      |         |    ______
==================================================================
0    5    10   15   20   25   30   35   40   45   50   55   60   65
```

Focus a spesific building



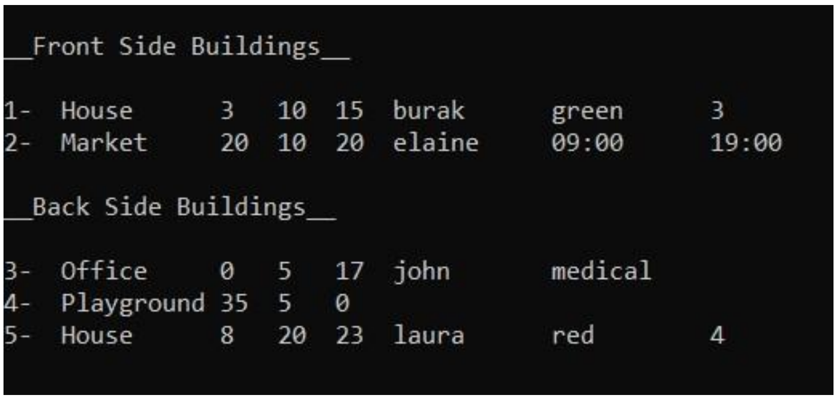
```
Input = 6

__Front Side Buildings__

1-  House      3   10  15  burak      green      3
2-  Market     20  10  20  elaine     09:00      19:00

__Back Side Buildings__

3-  Office     0   5   17  john       medical
4-  Playground 35  5   0

Which building do you want to focus?
Input = 1

focus() function returned = burak


Type = House
Owner = burak
Color = green
Number Of Room = 3
Position = 3
Length = 10
Height = 15
```

Focus another building



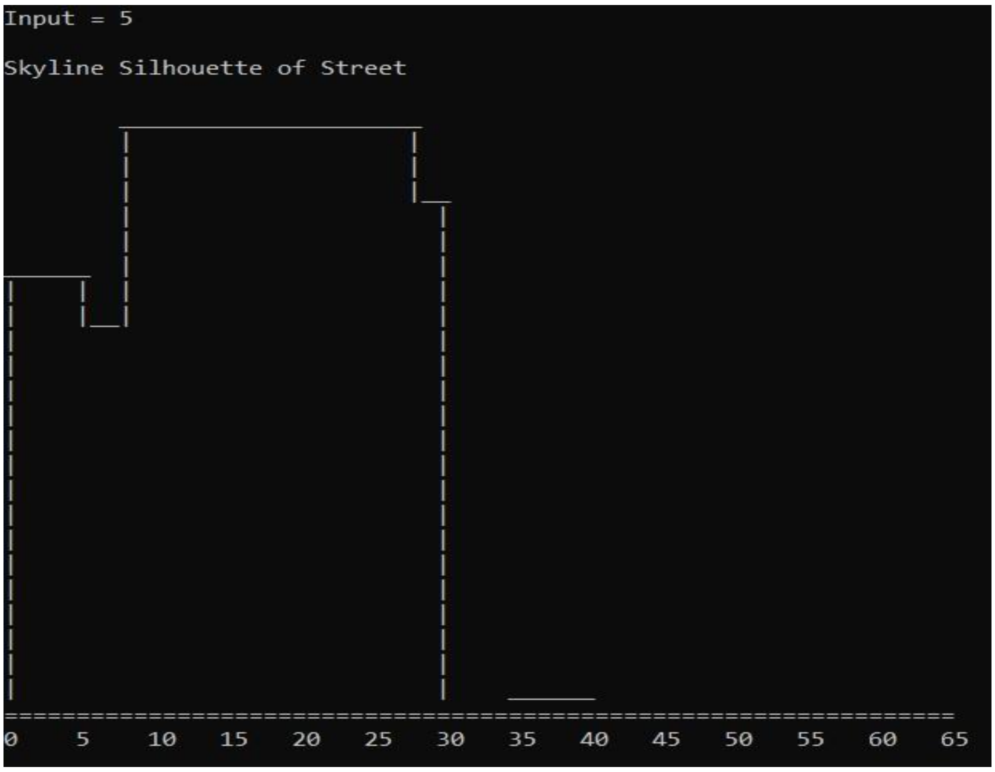
```
Input = 6

__Front Side Buildings__

1-  House      3   10  15  burak      green      3
2-  Market     20  10  20  elaine     09:00      19:00

__Back Side Buildings__

3-  Office     0   5   17  john       medical
4-  Playground 35  5   0

Which building do you want to focus?
Input = 3

focus() function returned = medical


Type = Office
Job Type = medical
Owner = john
Position = 0
Length = 5
Height = 17
```

After adding a house



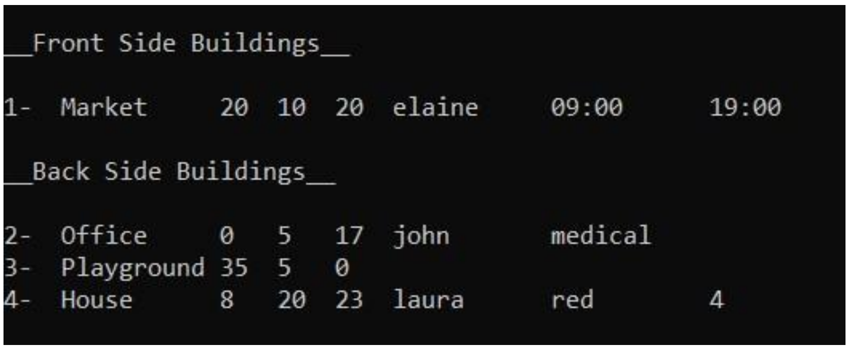
```
__Front Side Buildings__

1-  House      3   10  15  burak      green      3
2-  Market     20  10  20  elaine     09:00      19:00

__Back Side Buildings__

3-  Office     0   5   17  john       medical
4-  Playground 35  5   0
5-  House      8   20  23  laura      red        4
```

Silhouette after addition



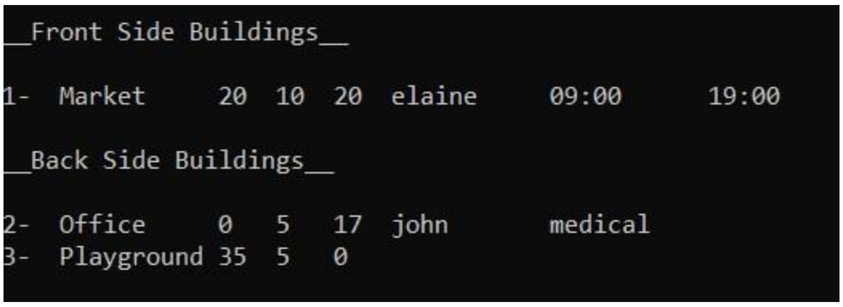
```
Input = 5

Skyline Silhouette of Street

        _____________________
        |                   |
        |                   |
        |                   |__
        |                     |
        |                     |
        |                     |
______  |                     |
|    |  |                     |
|    |__|                     |
|                             |
|                             |
|                             |
|                             |
|                             |
|                             |
|                             |
|                             |
|                             |
|                             |
|                             |
|                             |
|                             |
|                             |
|                             |    ______
==================================================================
0    5    10   15   20   25   30   35   40   45   50   55   60   65
```

Delete one building



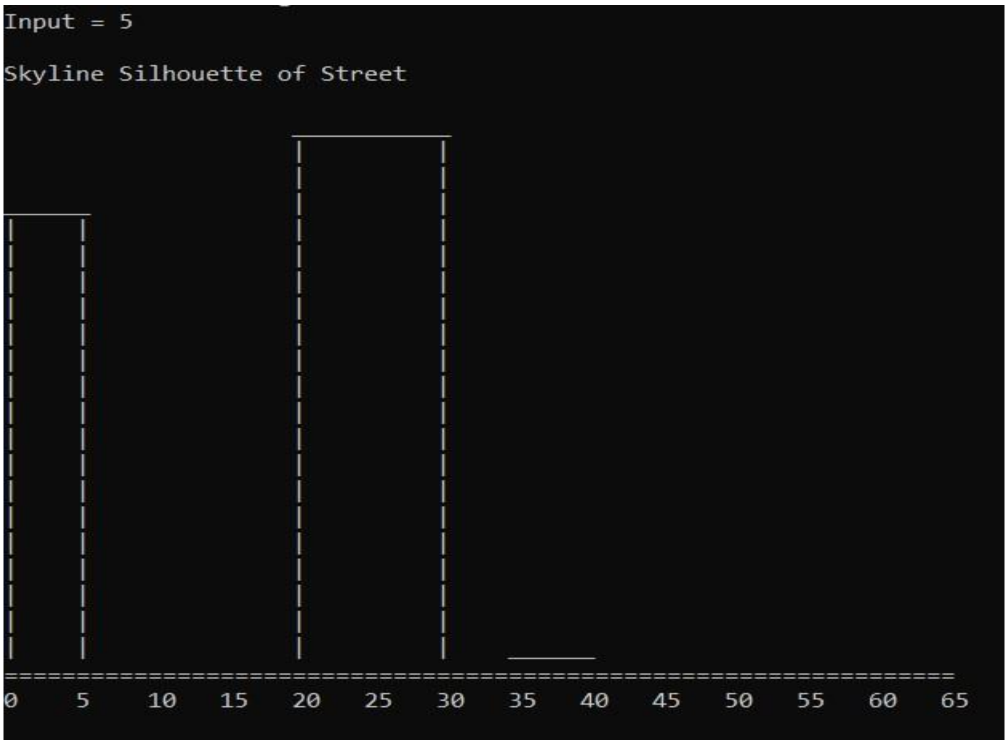
```
__Front Side Buildings__

1-  Market     20  10  20  elaine     09:00      19:00

__Back Side Buildings__

2-  Office     0   5   17  john       medical
3-  Playground 35  5   0
4-  House      8   20  23  laura      red        4
```

Delete another building



```
__Front Side Buildings__

1-  Market     20  10  20  elaine     09:00      19:00

__Back Side Buildings__

2-  Office     0   5   17  john       medical
3-  Playground 35  5   0
```

Silhouette after deleting



```
Input = 5

Skyline Silhouette of Street

                    ___________
                    |         |
                    |         |
______              |         |
|    |              |         |
|    |              |         |
|    |              |         |
|    |              |         |
|    |              |         |
|    |              |         |
|    |              |         |
|    |              |         |
|    |              |         |
|    |              |         |
|    |              |         |
|    |              |         |
|    |              |         |
|    |              |         |
|    |              |         |
|    |              |         |
|    |              |         |    ______
==================================================================
0    5    10   15   20   25   30   35   40   45   50   55   60   65
```

Test Case results are inside test.txt, and these cases are run before menu shows up. “make” command compiles, and “make run” command runs the program.