**“Java Project – Elevator Simulator”**

**Analysis**

For this coursework, I was given the task to create a simulation of a lift system in Java. I was also given a scenario in which the lift system should be based on. The scenario is as follows:

*“A building has four floors serviced by a single lift. There are request buttons at each floor for ‘Up’ and ‘Down’ (except the top floor, which only has a ‘Down’ button and the ground floor, which only has an ‘Up’ button).”*

*“The lift also has request buttons inside for each floor. The user will press the button for the destination floor and it will light up and remain lit until the lift stops at that floor.”*

*“The lift has a set of doors, which can be opened and closed and has a motor which can make the lift go up, go down or stop at the next floor.”*

*“When there are no requests the lift will wait with the doors closed at the last floor it reached.”*

*“The lift is sent a signal each time it reaches a floor and another signal when it leaves a floor, so that it can detect where it is and control its motor.”*

*“Each floor shows a display of which direction the lift is moving (Up, Down or Waiting) and where the lift is (at a particular floor, or between one floor and the next).”*

*“A control system must manage the requests for the lift, the opening and closing of the doors, and the movement of the lift to service requests. For safety, it must ensure that the doors have been closed before the motor is set to go up or down and that the doors cannot be opened until the motor is stopped.”*

After looking at the scenario given, it is necessary to find the different Classes, objects, attributes and methods that will be used when making the lift simulator. Below is the list of Classes, Object, Attributes and Methods that I have chosen for this task:

**Classes** – A template that states the what different types of behaviour an object supports

* LiftController: Holds many of the methods including the main methods
* Lift: Holds the methods and attributes of the lift such as the door of the lift
* Floor: Holds the methods and attributes of the floor such as the floor elevator doors
* Buttons: Holds the methods and attributes for the buttons such as button to call the lift

**Objects** – An instance of a Class

* LiftController
* Lift
* Floor
* Buttons

**Attributes** – Another name for a field, it is a public variable that can be constant or static

* liftFloor: An integer variable that stores which floor the lift is in currently
* floorDestination: An integer variable that stores which floor number the user wants to go
* liftDoor: A Boolean variable (true or false) that stores the information about the lift’s door state which is either open (true) or closed (false)
* floorDoor: A Boolean variable (true or false) that stores the information about the floor’s door state which is either open (true) or closed (false)
* currentFloor: An integer that holds which floor is the user

**Methods** – A grouped statement that used or called to perform an operation in the program

* openLiftDoor(): used or called when the door of the lift needs opening
* closeLiftDoor(): used of called when the door of the lift needs closing
* moveUp(): used or called when the lift needs to move up
* moveDown(): used or called when the lift needs to move down
* liftWaiting(): used or called if the lift is on the same floor as the user
* floorDoorOpen(): used or called when the door of the floor to the lift needs opening
* floorDoorClosed(): used or called when the door of the floor to the lift needs opening
* floorButton(): used or called when the user calls for the lift. It also stores information such as if the user wants to go up or go down. Of course when limited such as of the user is on the top floor, the user can only go down or when the user is at the bottom floor, which the user can only go up.
* liftButton(): used or called when the user is inside the lift and used to select which floor to go to.
* callLift(): used or called when the lift is summoned by the user

**Design**

After finding the classes, objects, attributes and the methods that is necessary for the project, the next step is to design the class diagram and object diagram for this project. For this project, have decided to use Bluej, a Java IDE that is free and is well supported.

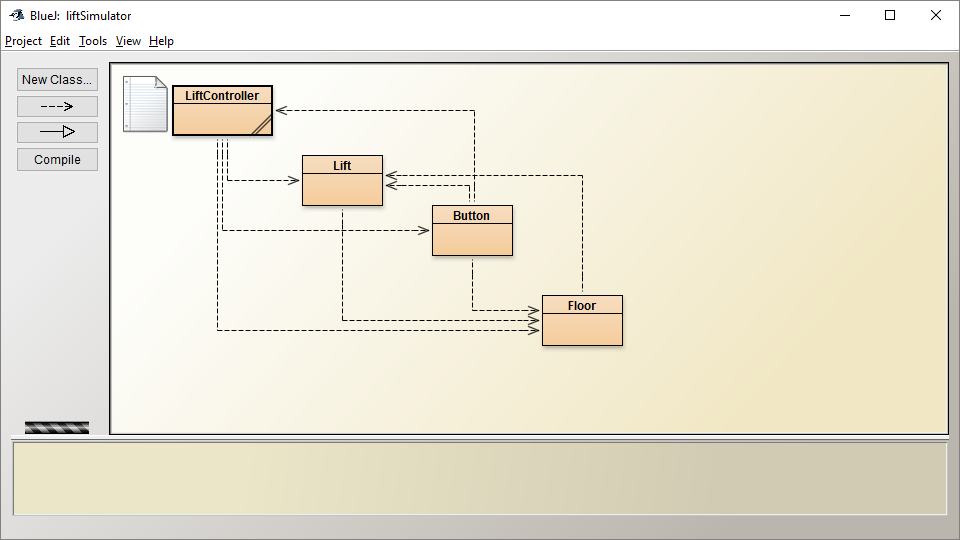


Figure 1 – Class diagram

The figure 1 shows the Bluej class diagram. Its show the relation ship between the classes.

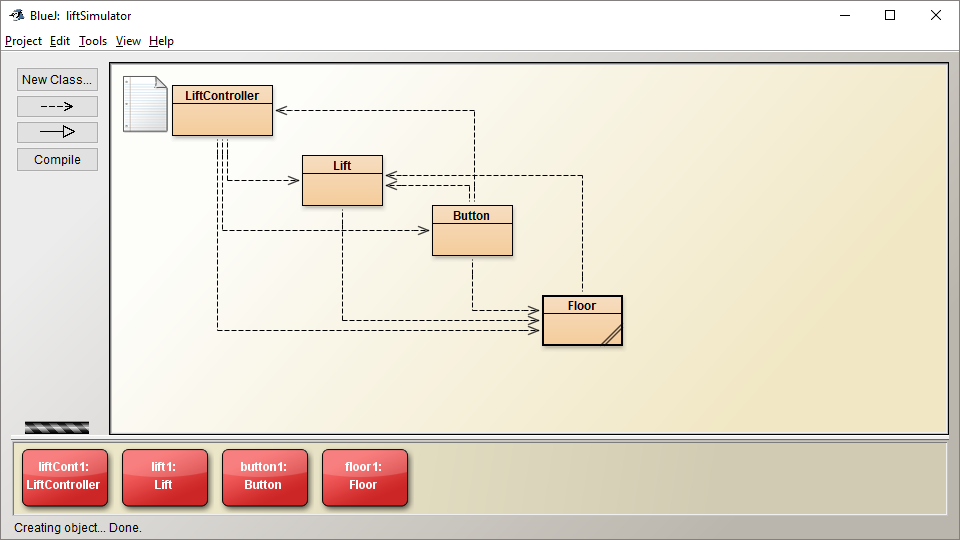


Figure 2 - Objects

Figure 2 shows all the cobjects that needs to be created in order to run the program.

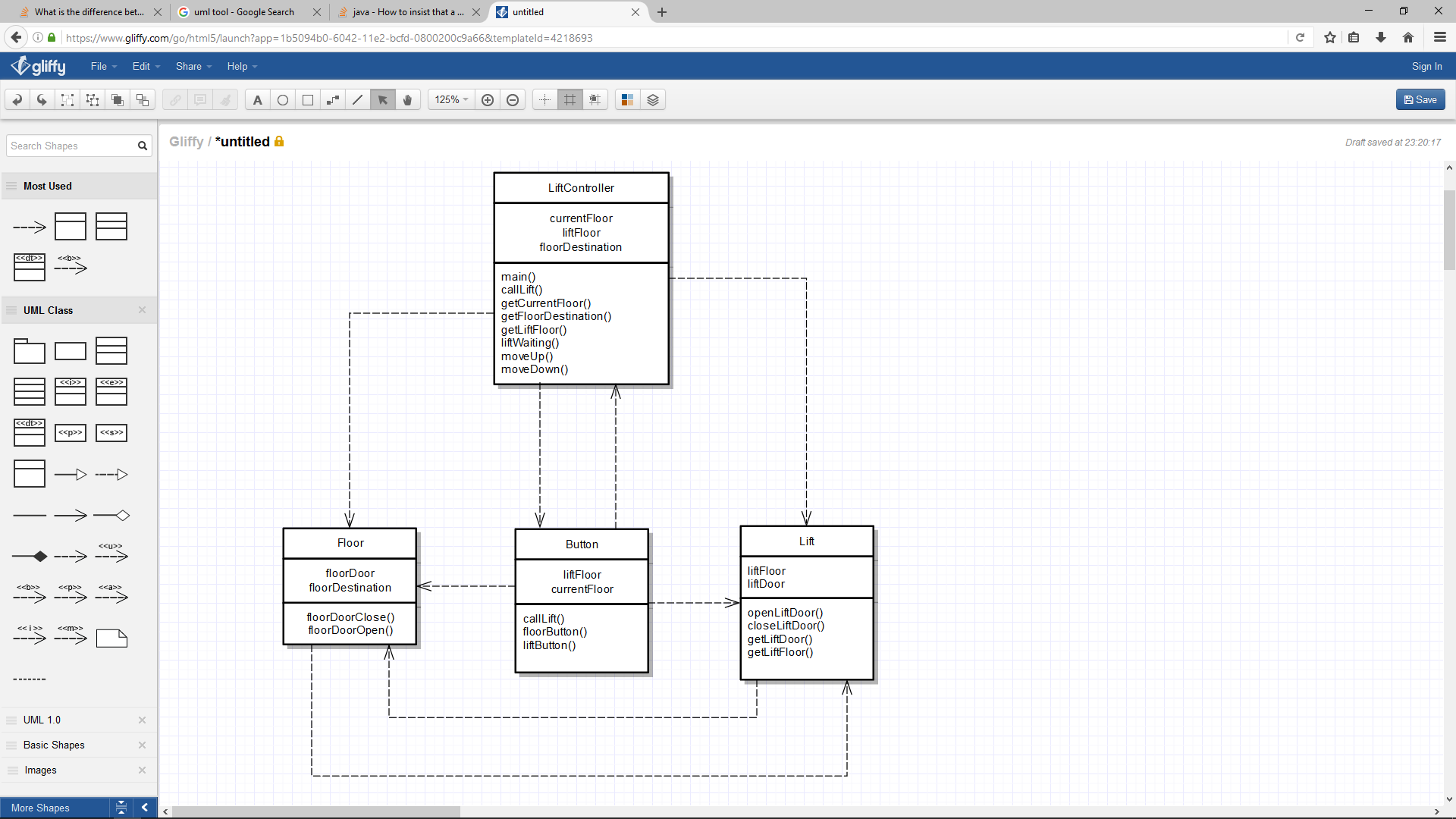


Figure 3 – Object Diagram

Figure 3 shows the Object Diagram. It show the different methods and the attributes of the objects and also shows the different relationships between each other.

**Implementation**

For the implementation, figure such as figure 4 – 8 shows the class hierarchy of the objects and the summary for each of the classes. The source code for all the classes is also included in this section.

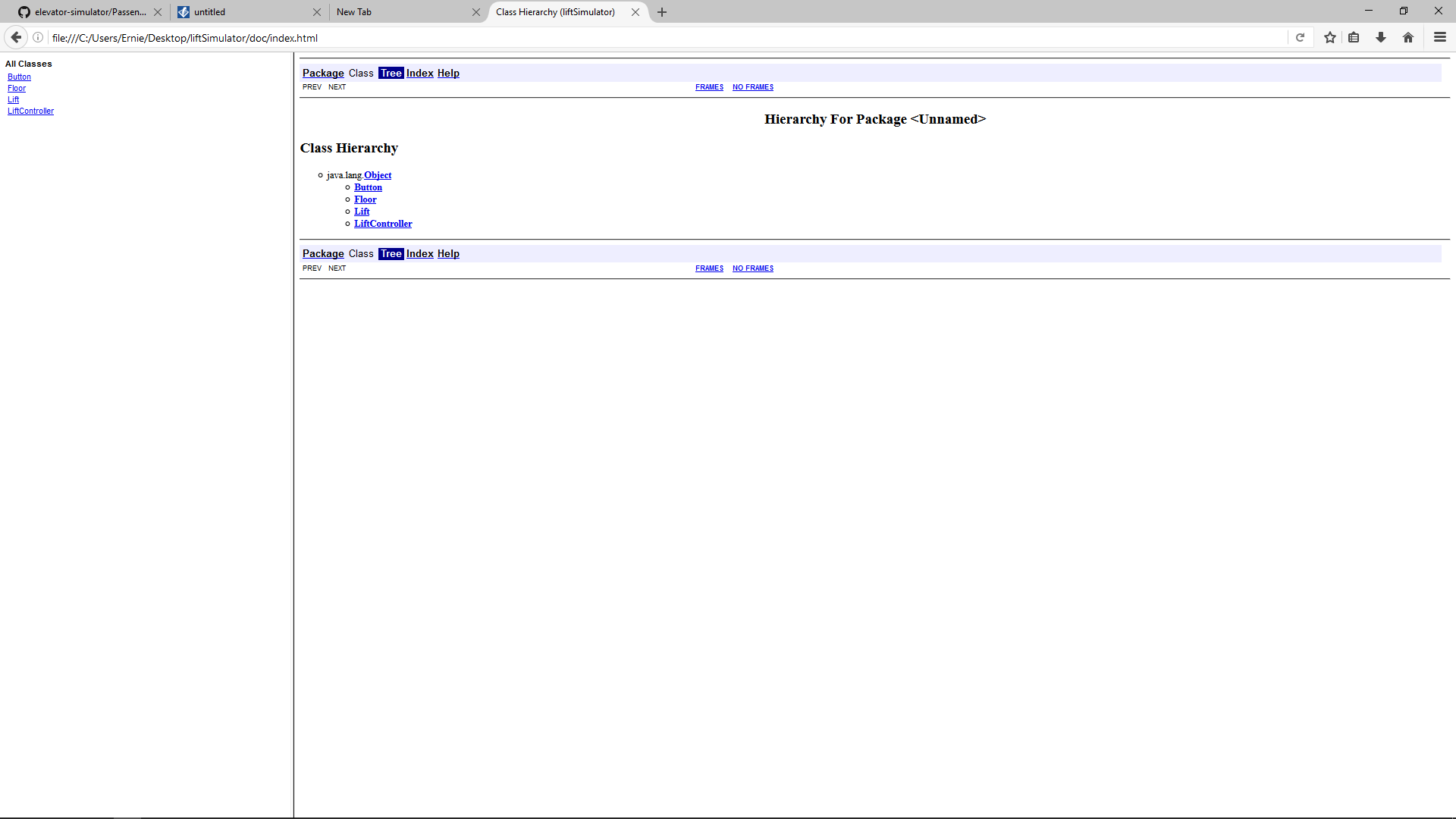


Figure 4 – Class Hierarchy

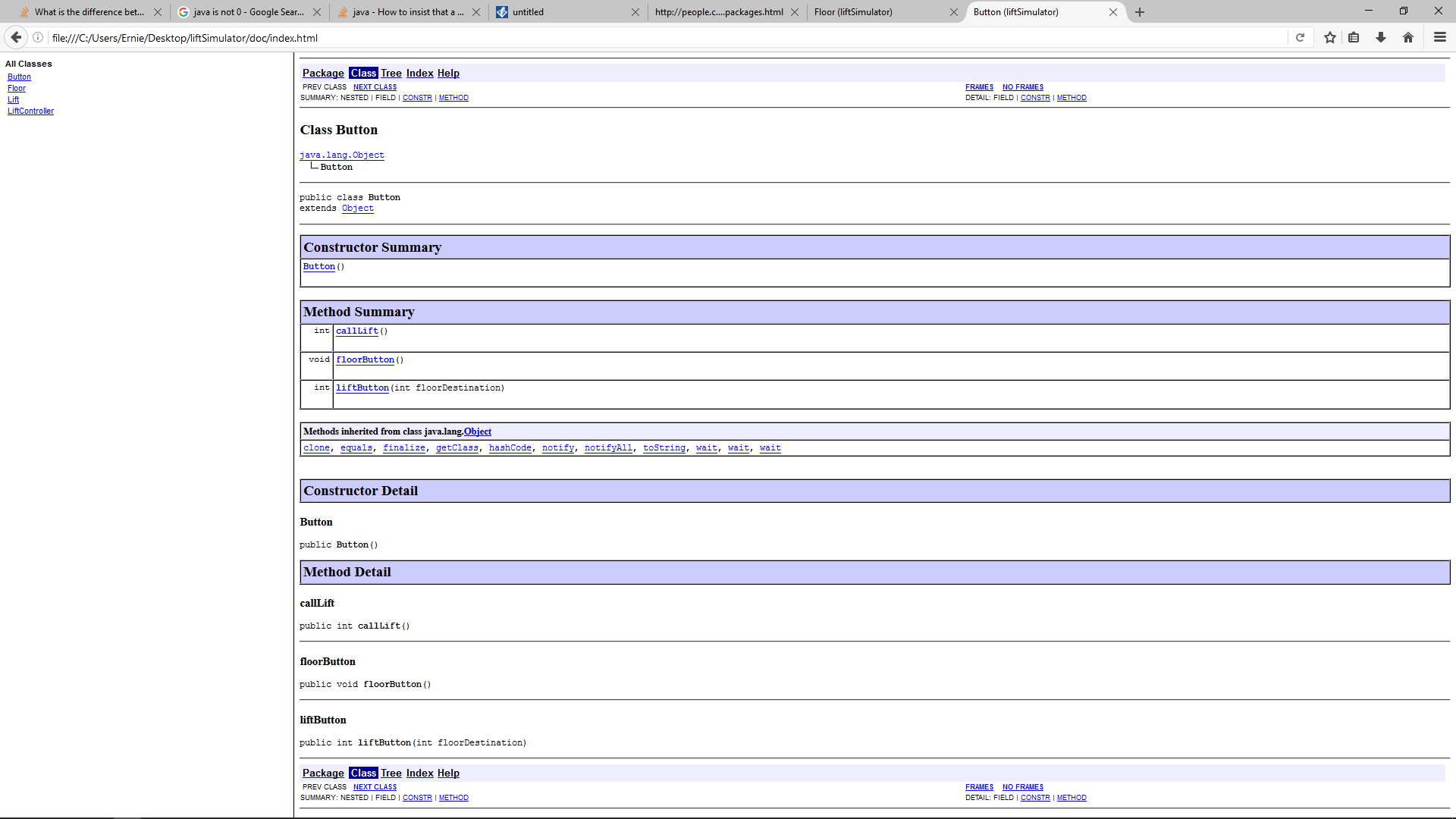


Figure 5 – Class button javdoc

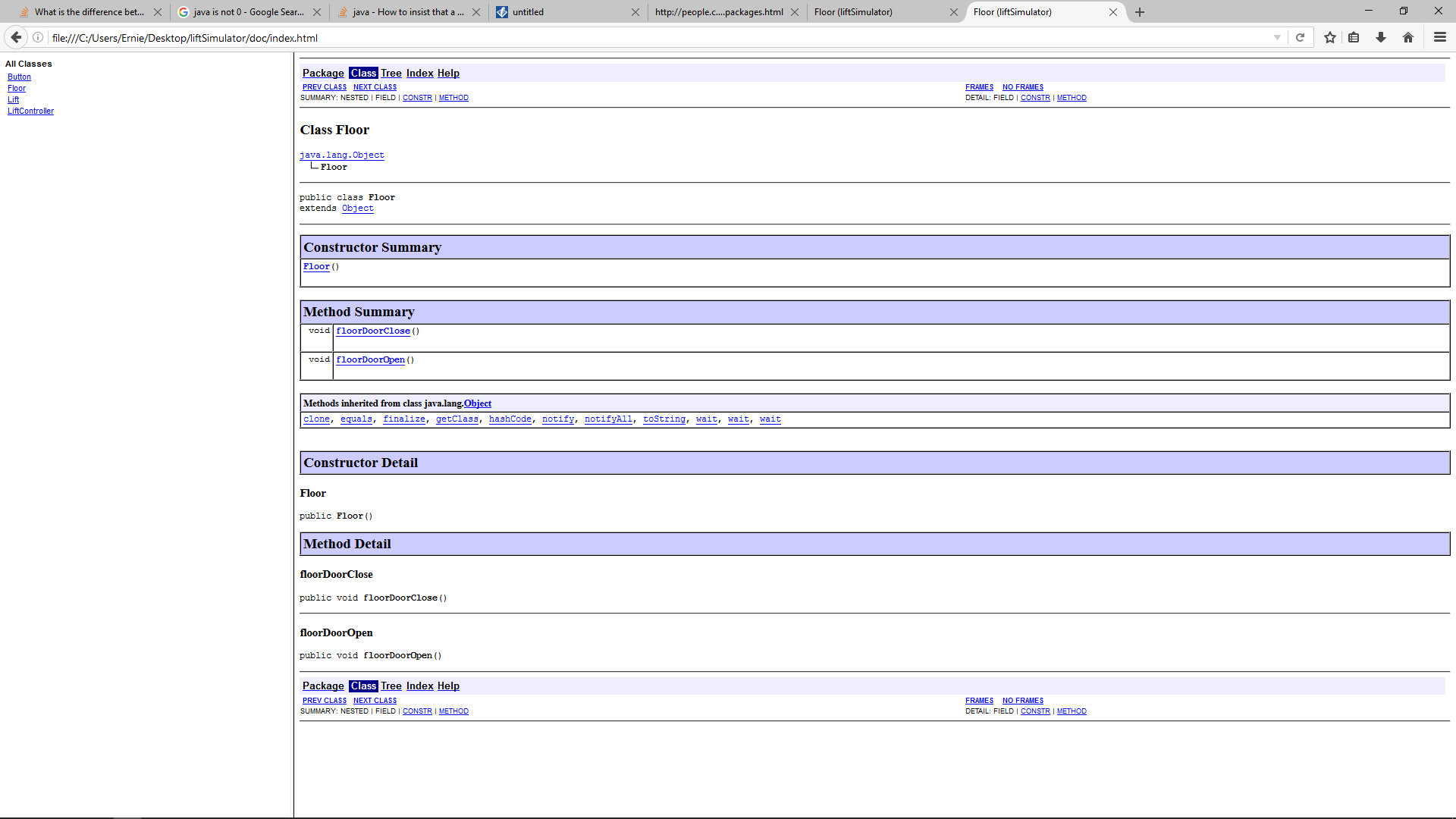


Figure 6 – Class Floor javdoc

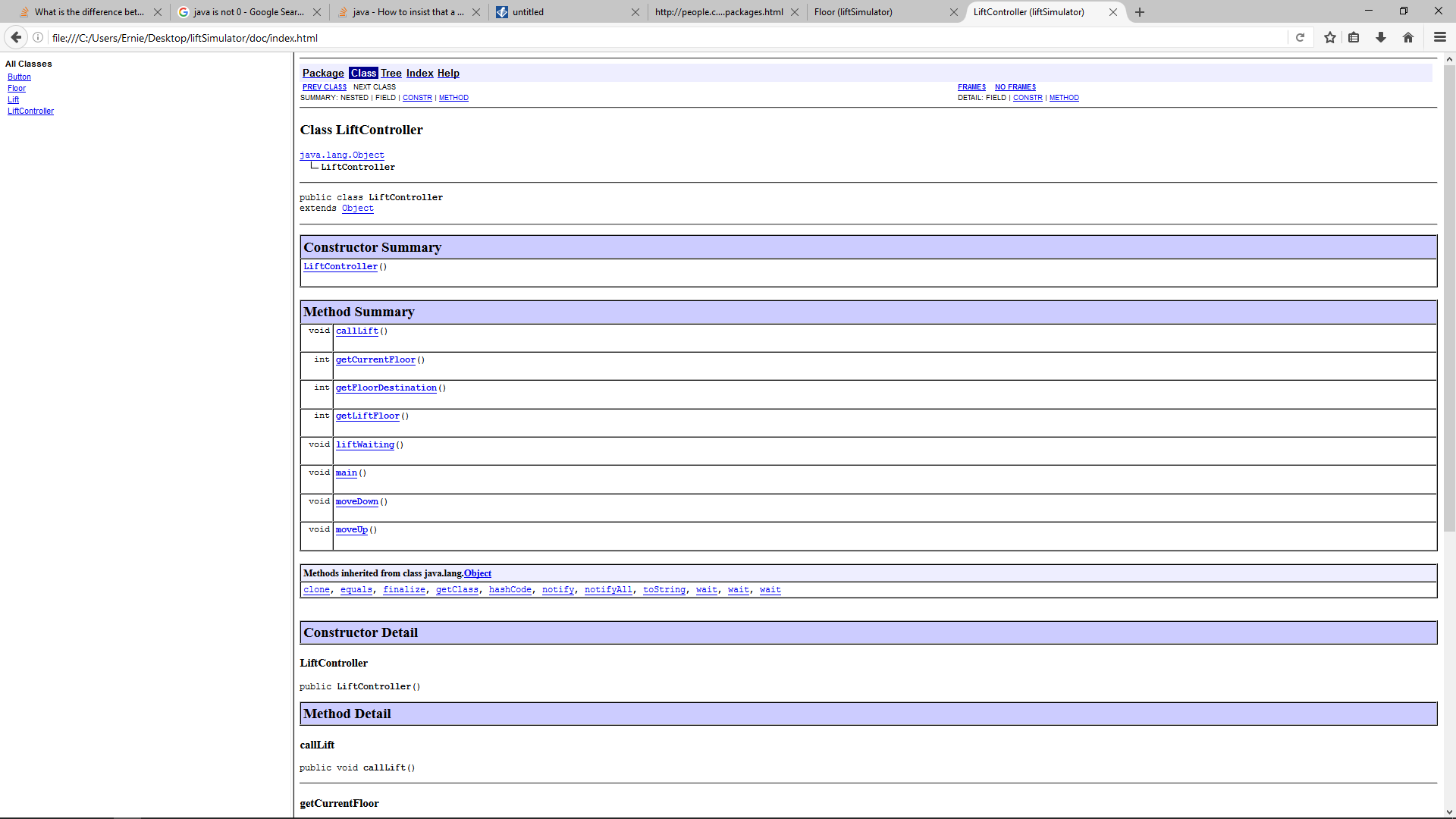


Figure 7 – Class LiftController javdoc

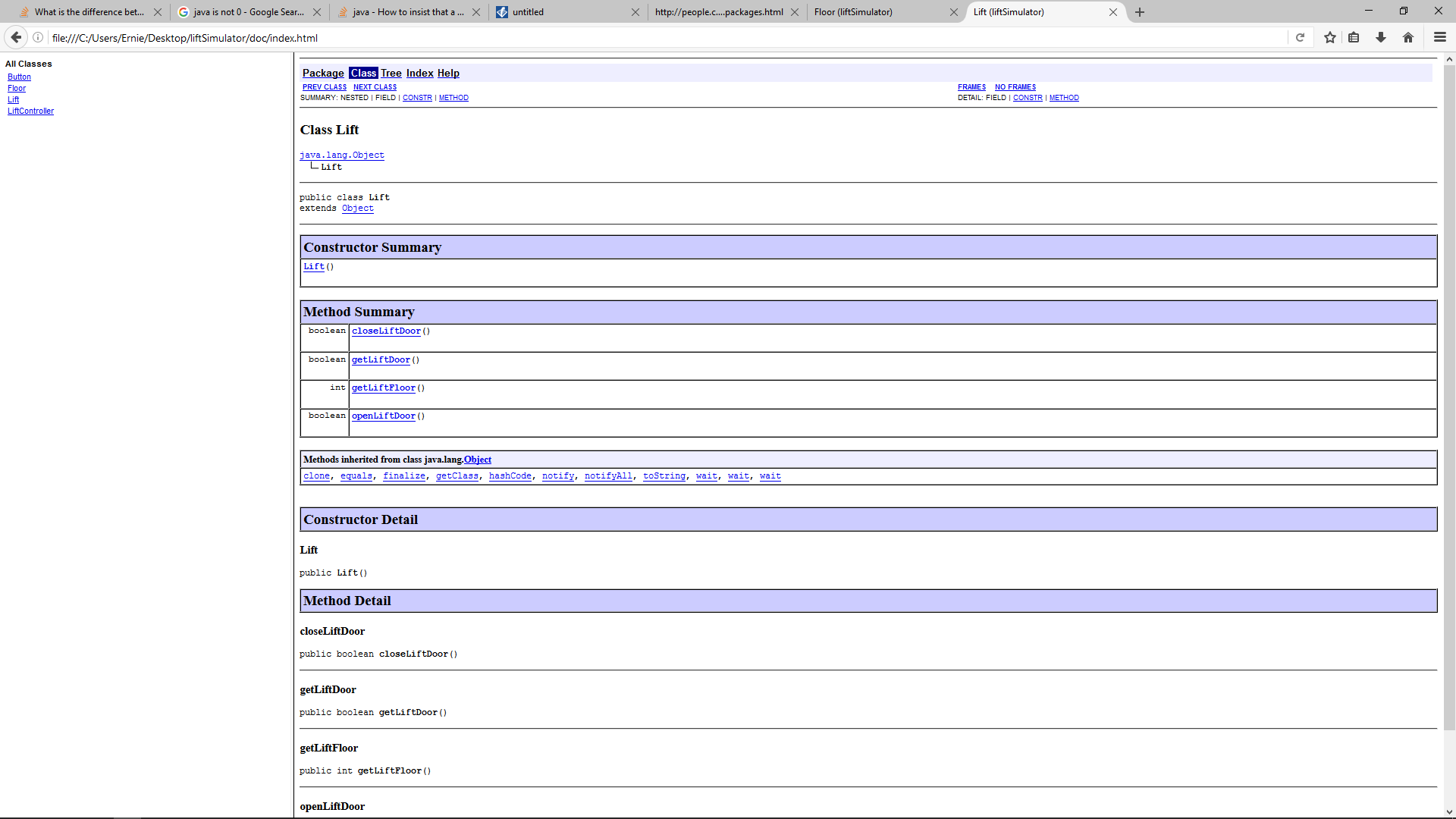


Figure 8 – Class Lift javdoc

Codelisting:

* Button.java
* Floor.java
* Lift.java
* liftController.java

**Source code – Lift Controller**

/\*imports the java utility Scanner. It is used to get inputs from the user\*/

import java.util.Scanner;

public class LiftController

{

private int currentFloor;

private int liftFloor;

private int floorDestination;

/\* The main method of the program \*/

public void main(){

LiftController newLiftController = new LiftController();

Lift newLift = new Lift();

Floor newFloor = new Floor();

liftFloor = newLift.getLiftFloor();

Button newButton = new Button();

Scanner whichFloor = new Scanner (System.in);

System.out.println("which floor are you in? 1,2,3,4?");

while(!whichFloor.hasNextInt()) {

whichFloor.next();

System.out.println("please input integer");

}

currentFloor = whichFloor.nextInt();

switch(currentFloor){

case 0: currentFloor = 0;

System.out.println("invalid floor number");

break;

case 1: currentFloor = 1;

System.out.println("you are in the 1st floor");

break;

case 2: currentFloor = 2;

System.out.println("you are in the 2nd floor");

break;

case 3: currentFloor = 3;

System.out.println("you are in the 3rd floor");

break;

case 4: currentFloor = 4;

System.out.println("you are in the 4th floor");

break;

case 5: if (currentFloor > 4){

System.out.println("invalid floor number");

}

}

if(liftFloor == currentFloor){

liftWaiting();

} else {

callLift();

}

Scanner whichDestination = new Scanner (System.in);

System.out.println("which floor do you want to go?");

floorDestination = whichDestination.nextInt();

if(floorDestination > currentFloor){

moveUp();

} else{

moveDown();

}

}

/\* Moves the Lift up the floors \*/

public void moveUp(){

Lift lft1 = new Lift();

liftFloor = lft1.getLiftFloor();

Floor flr1 = new Floor();

lft1.closeLiftDoor();

while (liftFloor != floorDestination){

liftFloor++;

System.out.println("Lift is going up");

System.out.println("Current Floor of the Lift is: " + liftFloor);

}

lft1.openLiftDoor();

}

/\*Moves the lift down the floor\*/

public void moveDown(){

Lift lft2 = new Lift();

liftFloor = lft2.getLiftFloor();

Floor flr2 = new Floor();

while (liftFloor != floorDestination || liftFloor < 1){

liftFloor--;

System.out.println("Lift is going down");

System.out.println("Current Floor of the Lift is: " + liftFloor);

}

lft2.openLiftDoor();

}

/\* Checks if the lift is at the same floor as the user, if so it opens the doors to the lift \*/

public void liftWaiting(){

Lift lft3 = new Lift();

if (currentFloor == liftFloor){

System.out.println("The lift is currently waiting");

lft3.openLiftDoor();

}

}

/\*Calls the lift if the lift isnt in the same floor as the user\*/

public void callLift(){

if(liftFloor != currentFloor){

moveUp();

} else{

moveDown();

}

}

/\*a method that returns currentFloor variable. It is created so it is easier to access if the variable is used in another class\*/

public int getCurrentFloor(){

return currentFloor;

}

/\*a method that returns floorDestination variable. It is created so it is easier to access if the variable is used in another class\*/

public int getFloorDestination(){

return floorDestination;

}

/\*a method that returns liftFloor variable. It is created so it is easier to access if the variable is used in another class\*/

public int getLiftFloor(){

return liftFloor;

}

}

**Source Code – Lift**

/\*imports the java utility Scanner. It is used to get inputs from the user\*/

import java.util.Scanner;

public class Lift

{

private int liftFloor;

private boolean liftDoor;

/\*The constructor for the Lift Class\*/

public Lift(){

liftFloor = 1;

liftDoor = false;

}

/\*the method for opening the door of the lift. It is connected to the floorDoorOpen method so both door open at the same time when called\*/

public boolean openLiftDoor(){

Floor f1 = new Floor();

liftDoor = true;

System.out.println("Lift door is opened");

f1.floorDoorOpen();

return true;

}

/\*the method for closing the door of the lift. It is connected to the floorDoorClosed method so both door open at the same time when called\*/

public boolean closeLiftDoor(){

Floor f2 = new Floor();

liftDoor = true;

System.out.println("Lift door is closed");

f2.floorDoorClose();

return true;

}

/\*a method that returns liftFloor variable. It is created so it is easier to access if the variable is used in another class\*/

public int getLiftFloor(){

return liftFloor;

}

/\*a method that returns liftDoor variable. It is created so it is easier to access if the variable is used in another class\*/

public boolean getLiftDoor(){

return liftDoor;

}

**}**

**Source Code – Floor**

/\*imports the java utility Scanner. It is used to get inputs from the user\*/

import java.util.Scanner;

public class Floor

{

private boolean floorDoor;

private int floorDestination;

/\*The constructor for the Floor Class\*/

public Floor(){

floorDestination = 1;

}

/\*the method for opening the door of the lift in a floor of a building. It is connected to the openLiftDoor method so both door open at the same time when called\*/

public void floorDoorOpen(){

floorDoor=true;

System.out.println("floor door is opened");

}

/\*the method for closing the door of the lift in a floor of a building. It is connected to the closeLiftDoor method so both door open at the same time when called\*/

public void floorDoorClose(){

floorDoor = false;

System.out.println("floor door is closed");

}

} }

**public class Button**

**{**

private int liftFloor;

private int currentFloor;

/\*Constructor for Button\*/

public Button(){

}

/\*method for calling the Lift\*/

public int callLift(){

LiftController lc = new LiftController();

currentFloor = lc.getCurrentFloor();

Lift lf1 = new Lift();

liftFloor = lf1.getLiftFloor();

if(currentFloor < liftFloor){

lc.moveUp();

}

else if(currentFloor > liftFloor){

lc.moveDown();

}

System.out.println("Lift has arrived in current floor");

return liftFloor;

}

/\*method for the floor buttons\*/

public void floorButton(){

Floor fl = new Floor();

if (currentFloor > 0 && currentFloor == 4){

System.out.println("");

callLift();

fl.floorDoorOpen();

}

else if (currentFloor == 4){

System.out.println("");

callLift();

fl.floorDoorOpen();

}

}

/\*method for the buttons on the lift\*/

public int liftButton(int floorDestination){

if(floorDestination != 0 && floorDestination > 5){

while (liftFloor != floorDestination){

System.out.println("Current Floor of lift is: " + liftFloor);

liftFloor++;

}

}

return floorDestination;

}

}

**Testing**

I have devised a test plan to test if the function of the lift system. These set of test ensures that the code I is up to the standard that the scenario wants on the system. It will also test for any error conditions such as inputting the wrong answer.

|  |  |  |  |
| --- | --- | --- | --- |
| Test | Expected results | Results | See Figure |
| Running the program | A command line should open and ask the user which floor to start with | A command line opened and ask the user which floor to start with | 9 |
| Lift and the user on the same floor | The Lift and the floor door should be open and allows the user to pick which floor to go | The Lift and the floor door is open and allows the user to pick which floor to go | 10 |
| Testing the Lift Going up | The Lift and the floor door should close and the lift should go the user’s destination floor. It must also tell which floor the lift is passing and which direction its going | The Lift and the floor door should close and the lift should go the user’s destination floor. It tells the floor which door its passing and its telling the user which direction its going | 11 |
| Test the lift going down | The Lift and the floor door should close and the lift should go the user’s destination floor | The Lift and the floor door should close but the floor isn’t stopping to the correct floor | 14 |
| Putting an invalid floor when asked | It should show a message that says “invalid floor” and should allow the user to input again | It shows a message that says “invalid floor” and allowed the user to input again | 12 |
| Putting an invalid floor when asked (imputing a character or string rather than an integer) | It should show a message that says “please put an integer” and should allow the user to input again | It shows a message that says “please put an integer” and allowed the user to input again | 13 |
| Call the lift on another Floor | The Lift should close the door and move to the same floor as the user | Same result with the test with the lift going down | 14 |

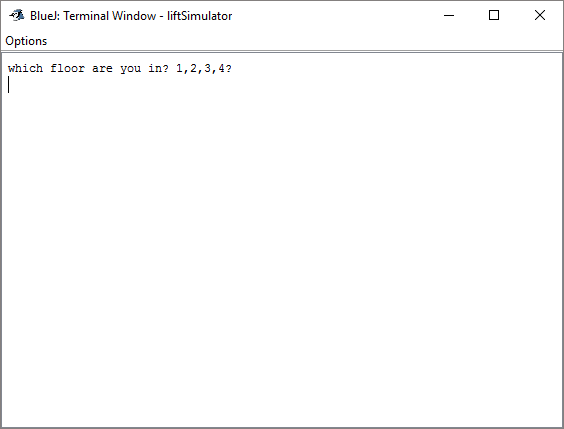


Figure 9 – Start of the Program

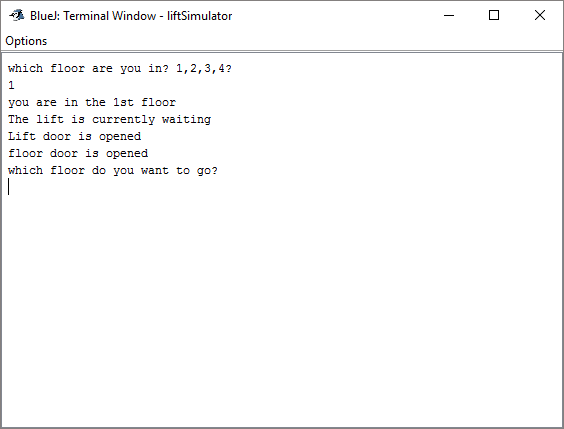
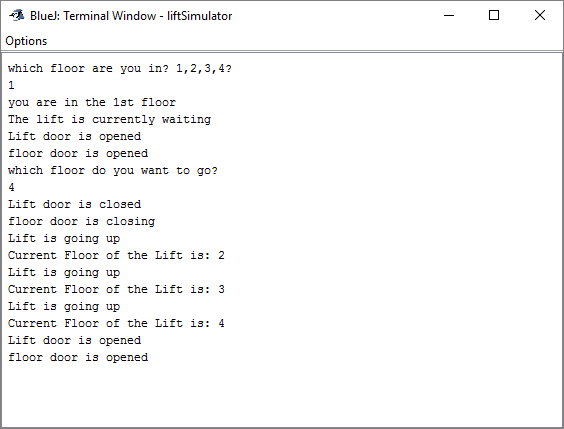


Figure 10 (above) – Lift and the User are on the same floor

Figure 11 (Bellow) – Lift going up to the floor that the user wants



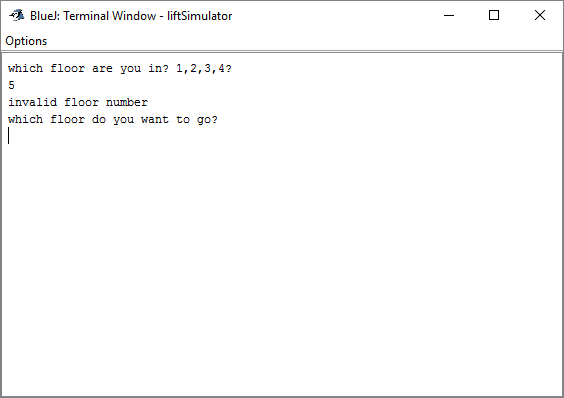
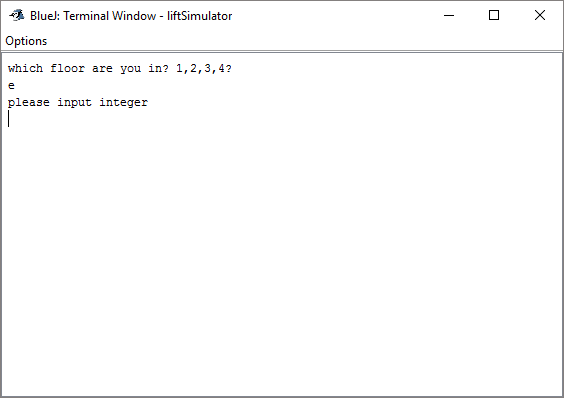


Figure 12 (above) – testing for invalid floor number

Figure 13 (below) – testing for inputting a character instead of an integer



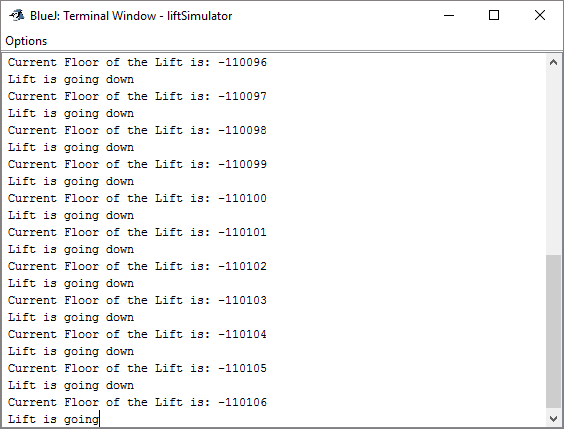


Figure 14 (above) – testing the lift going down

**Appraisal**

Looking at the project overall, the methods that I have used for this project are adequate enough to do what I was aiming for. I do admit that some of the methods does not work as I intended it to be, but there is always room for improvement in the future if so given the task again. I am quite happy that the methods that open and closes the door works, especially when it simultaneously closes and opens together when necessary. The mechanism behind the door lift and the floor lift that does not open when the lift is moving also is another one of the highlights of the that method.

The method lift waiting also worked well as I intended it to be as it opens the door and the floor door when the user is on the same floor as the lift. Of course, there are a lot of methods that I used that didn’t work, for example the lift calling mechanism needs a bit of work if given more time.

The method that allows the lift to move up the user is also spot on, unfortunately as I cannot replicate the results for the moving down method, the lift doesn’t work as I intended it to be. But hopefully if given more time, the method will be fixed and therefore fully simulate an Elevator.

The method for calling the lift also have the same problems as the method for moving down the elevator. Same with the moving down method, given more time I can confidently say that it can be fixed and also improved on.

I will also add in the future more features such as even listeners, door delays when closing and opening the doors, a working GUI (Graphical User Interface, add another Lift, allows the user to add more floors and also add if possible add passengers to the lift that also randomly goes in and out of the lift.