Abstract

Starting from a hand powered elevator till reaching to what we are dealing with today of systematically complex one's, clarify to us that technological development doesn't stop, and whenever time is passing, new developments will be available to us helping to solve every problem we face. Simulation is a way of developing application or design simulating real world, the way which help in clarifying the real idea. The reasons upon using simulations may be for literature, cost, time, or safety reasons. The elevator challenge was born out of a long time curiosity in the control of elevators. Elevator simulation is becoming increasingly more

flexible and powerful. Elevator simulation models of varying sophistication have been written and applied for many years. The continuing improvements in computer technology and software development tools make increasing complex and comprehensive simulation models feasible.

Table of contents

Contents

Elevator Simulation

Introduction:

This documentation contains a specification of the Elevator simulation which is implemented in Java languages. In the first part of this document, All the CRC (Class Responsibility Collaboration) are described to fulfil the OOP concepts and other brainstorming purposes. In the second part we can see the UML (Unified modeling language) Class diagrams of the whole Elevator in which we describe, how the classes interacted with other class and perform the tasks according to the scenario which is given at the start. All the Elevator simulation are described in UML class diagrams. In class we describe all public and private methods and data members which are used to perform the tasks.

Methodology:

All the data which is mentioned in this report is according to the requirements that are given. We write all the code in java language by using Eclipse IDE and the CRC cards are generating for brainstorming and the UML class diagrams are designed in Online UML tool.

Discussion:

For the sake of this report, we do research on the scheduling of Elevator and finally we done it after a lot of efforts. Following below are the CRC (Class Responsibilities Collaboration) and UML diagrams.

CRC (Class Responsibilities Collaboration) cards:

A Class Responsibility Collaborator (CRC) model (Beck & Cunningham 1989; Wilkinson 1995; Ambler 1995) is a collection of standard index cards that have been divided into three sections, as depicted in Figure 1. A class represents a collection of similar objects, a responsibility is something that a class knows or does, and a collaborator is another class that a class interacts with to fulfill its responsibilities.

Representation:

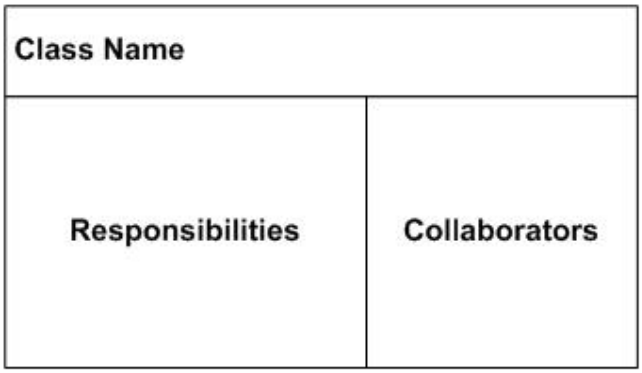

Figure 1: CRC layout

Elevator Simulation CRC cards:

These are the CRC cards of the Elevator Simulation.

Login:

class name:	Login
Responsibilities:	Collaborations:
Elevator counter	Elevator
Persons Limits	Person, Enter and exit
Floor counter	Floors, Start from 1st to last

Table 1: Login, CRC card

Main:

class name:	Main
Responsibilities:	Collaborations:
Elevator Count	Elevator
Floor Count	Person, Enter and exit
Person Limit	Floors, Start from 1st to last

Table 2: Main, CRC card

Elevator Inside Panel:

class name:	ElevatorInsidPanel
Responsibilities:	Collaborations:
Number of Digits	Elevator car
Phone Button	Elevator car
Alarm Button	Elevator car
Emergency Button	Elevator car
Stop Button	Elevator car
Floor Count	Floors, Start from 1st to last

Table 3: Elevator Inside Panel, CRC card

Floors:

class name:	Floors
Responsibilities:	Collaborations:
Floor Numbers	Building
Add Person Button	Elevator outside
Remove Person Button	Elevator Inside
Elevator Up Button	Elevator Inside
Elevator Down Button	Elevator Inside
Alarm Button	Elevator car
Emergency call button	Elevator car
Floor Count	Floors, Start from 1st to last

Table 4: Floors, CRC card

Elevator Outside:

class name:	ElevatorOutside
Responsibilities: Floor check Floor count Elevator Number Person limit	Collaborations: Building Floors, Start from 1st to last Building Person, enter and exit

Table 5: Elevator outside, CRC card

Simulator:

class name:	Simulator
Responsibilities: Floor Count Elevator Number	Collaborations: Floors, start from 1st to last Building

Table 6: Simulator, CRC card

Person:

class name:	Person
Responsibilities: Present Floor Target Floor	Collaborations: Simulator, Building, Floors Simulator, Building, Floors

Table 7: Person, CRC card

File Operation:

class name:	File Operation
Responsibilities: Files Directory	Collaborations: Directory Storage

Table 8: File Operation

Class Diagrams:

Class diagram, one of the most commonly used diagrams in object-oriented system, models the static design view for a system. The static view mainly supports the functional requirements of a system; the services of the system should provide to the end users. We will see from our practical experience that lots of fun comes out when modeling out system with class diagrams. The discussion on different views of class diagrams for the system will be put into emphasis later in this paper.

A class diagram shows a set of classes, interfaces, and collaborations and their relationships. Class diagrams involve global system description, such as the system architecture, and detail aspects such as the attributes and operations within a class as well. The most common contents of a class diagram are:

- Class
- Interfaces
- Collaborations
- Dependency, generalization, and association

Representation:

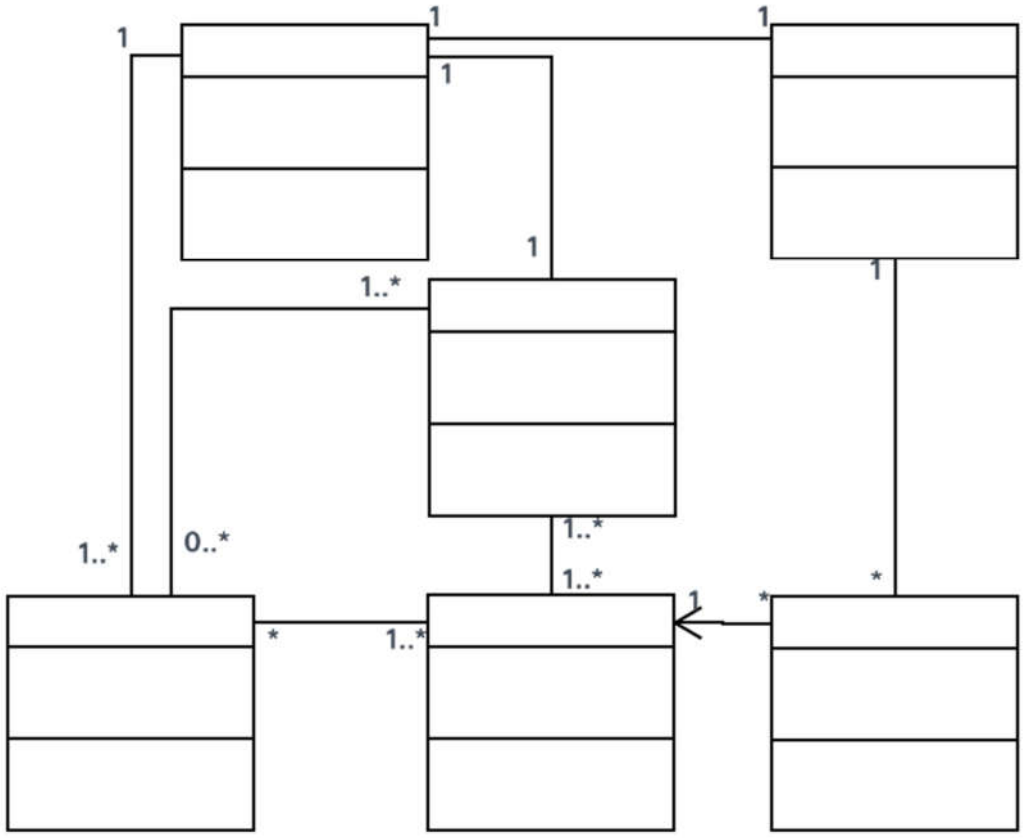

Figure 2: Class diagram representation

Elevator Simulation (Class Diagram):

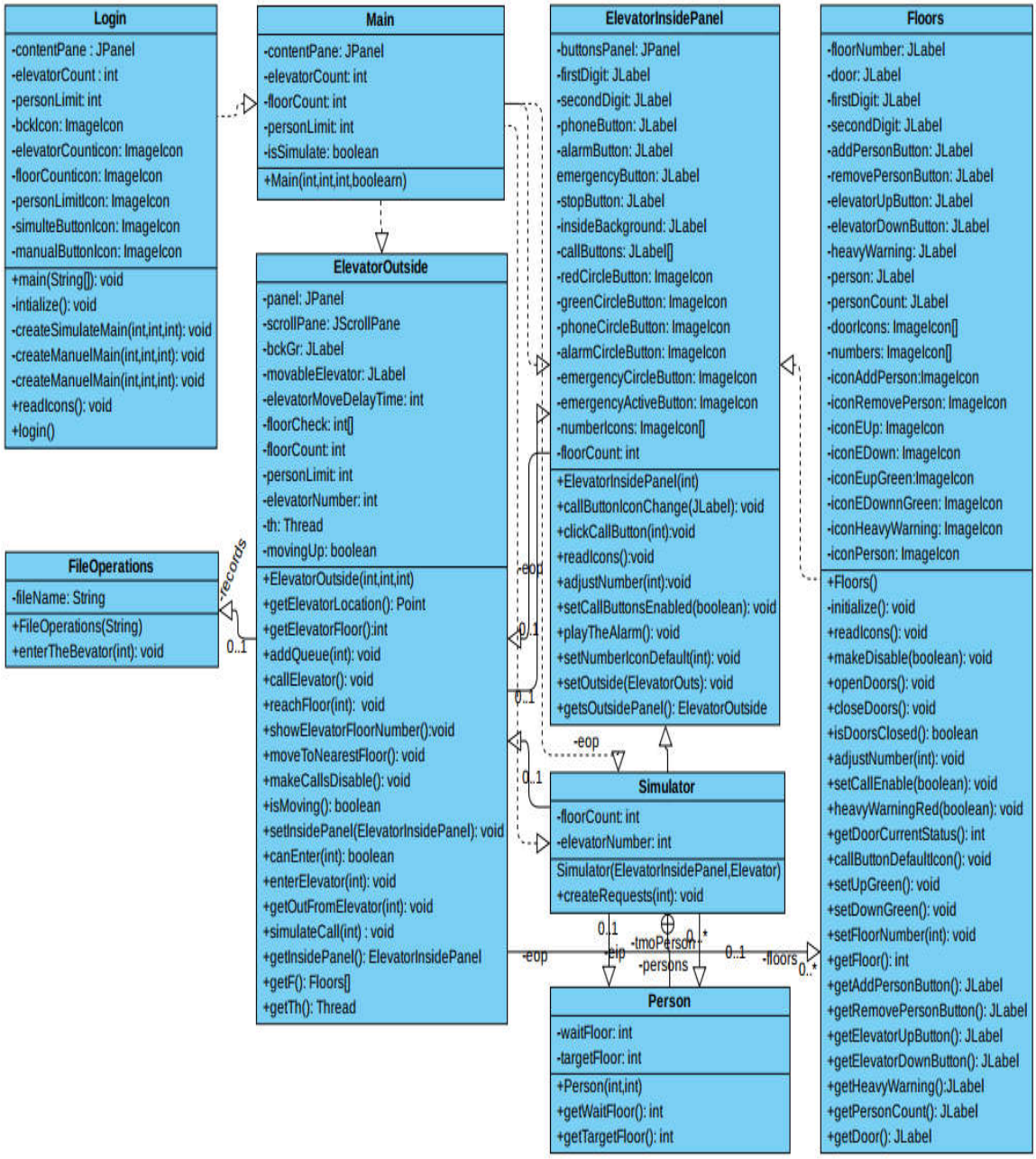

Figure 3: Class Diagram, 'Elevator Simulation'

Description:

For the sake of design, the UML class diagram of Elevator Simulation, I follow the CRC cards and done it in online designing tool which name is 'Visual paradigm'.

Following below are the description of UML classes:

Login:

This class will have done some functions like count the numbers of elevators and also count the numbers of person that are inside the elevator. This is linked with main class with realization.

Main:

This class performs the tasks like count the number of floors and count the number of persons that are inside the elevator a performs other GUI based tasks. This class is attached to diffierent classes those classes are 'ElevatorInsidePanel', 'ElevatorOutside', 'Simulator' with realization.

ElevatorInsidePanel:

This class perform many GUI based task which belongs to interface, security and safty.

Floors:

This class counts the number of floors and give this information to other class. There are many other methods that are performed in this class mostly methods are public. These are 'Floors', 'intiallize', 'readIcons', 'Open Doors', 'Close Doors' and 'adjust numbers'.

Elevator Outside:

This class checks the floors number and also perform the task to calculate the limit of persons an elevator numbers. This class performs many functions that are inside the class.

File Operations:

This class is used to read the names of file and perform operation like reading of data.

Simulator:

This class linked with different class through realization, generalization and composition. This class also count the Index of elevator.

Person:

This class count the numbers of person that are in queue and also performs the task in which target the floor.

Design Language:

Java is a programming language originally developed by James Gosling at Sun Microsystems and released in 1995 as a core component of Sun Microsystems. The language derives much of its syntax from C and C++ but has a simpler object model and fewer low-level facilities. Java applications are typically compiled to bytecode (class file) that can run on any Java Virtual Machine (JVM) regardless of computer architecture. Java is general-purpose, concurrent, class-based, and object-oriented, and is specifically designed to have as

few implementations dependencies as possible. It is intended to let application developers "write once, run anywhere".

There were five primary goals in the creation of the Java language:

1. It should be "simple, object oriented, and familiar".
2. It should be "robust and secure".
3. It should be "architecture neutral and portable".
4. It should execute with "high performance".
5. It should be "interpreted, threaded, and dynamic".

There were five primary goals in the creation of the Java language:

1. It should be "simple, object oriented, and familiar".

2. It should be "robust and secure".

3. It should be "architecture neutral and portable".

4. It should execute with "high performance".

5. It should be "interpreted, threaded, and dynamic".

Implementation:

Iterative and incremental processes represent best practice that has been identified from the limitations of the waterfall process. The term ‘iterative’ indicates the repetition of one or more activities; the term ‘incremental’ indicates that development proceeds from an initial subset of the requirements to more and more complete subsets, until the whole system is addressed. Incremental development involves an initial partition of the intended functionality; some or all of the subsequent development activities can be carried out independently and in parallel. The final product results from the total integration of the partitions.

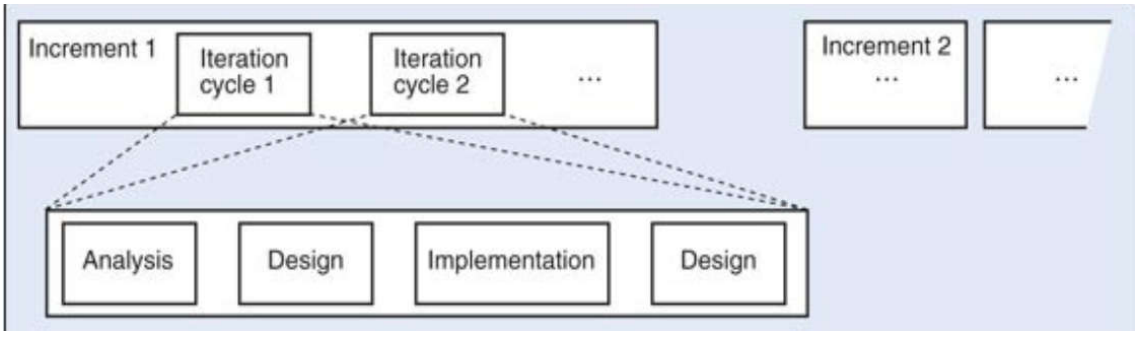

Figure 4: Implementation with iterative model

In iterative and incremental processes there is still a need for analysis, design, implementation and testing activities, but these activities are carried out in a more flexible way than in the waterfall process, since not all requirements are received at once, the requirements from customer goes on getting added to the list even after the end of "Requirement Gathering and

Analysis" phase, this affects the system development process. Although the problems with one phase are never solved completely during that phase and in fact many problems regarding a particular phase arise after the phase is signed off, that's why iterative process considered better than Waterfall process. An iterative and incremental process consists of several cycles of analysis, design, implementation and testing. Each cycle is short and provides feedback for the next cycle, in which a more refined and enhanced development is achieved. With an incremental model, development starts from small subsets of the requirements, reducing the complexity and scope of each analysis, design and coding cycle. Each increment is carried through the development activities to produce a working subset of the system, and is developed through several iterations. The integration of the increments results in the final system. However, this integration can be progressively achieved by successive releases of the software, each release achieving more functionality.

Testing:

Here the code of Junit testing in Eclipse and the screen short which indicates the code work correctly and performs all the operation correctly.

Code:

```
import static org.junit.Assert.*;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

/**
 *
 */

/**
 * @author HP
 *
 */
public class Elevator_SimulaitonTest_JUnitTest extends Elevator_Simulaiton {

	/**
	 * @throws java.lang.Exception
	 */
	@BeforeClass
	public static void setUpBeforeClass() throws Exception {
	}

	/**
	 * @throws java.lang.Exception
	 */
	@AfterClass
```

```
	public static void tearDownAfterClass() throws Exception {
	}

	/**
	 * @throws java.lang.Exception
	 */
	@Before
	public void setUp() throws Exception {
	}

	/**
	 * @throws java.lang.Exception
	 */
	@After
	public void tearDown() throws Exception {
	}

	@Test
	public void test() {
		fail("Not yet implemented");
	}

}
}
```

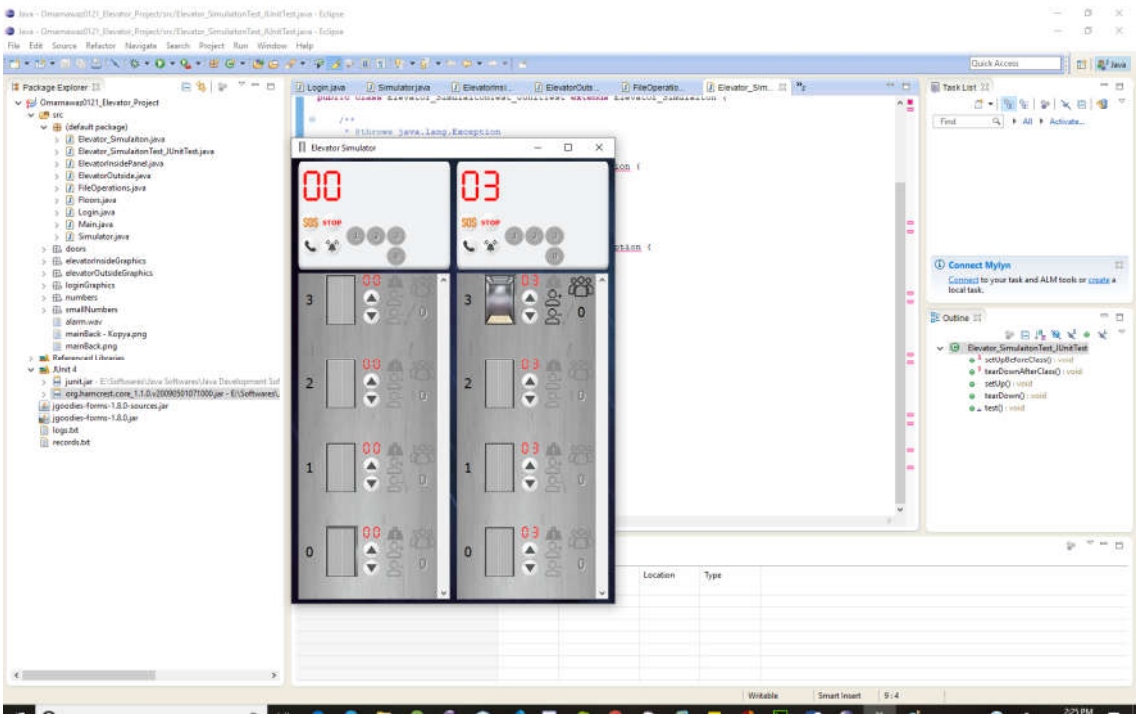

Figure 5: Layout with JUNIT Testing

Summary:

Simulation is a way of developing application or design simulating real world, the way which help in clarifying the real idea. The reasons upon using simulations may be for literature, cost, time, or safety reasons. Company problem appears when it is intended to equip its building with an elevator and want to know according to its needs the way to settle the elevator to best satisfy this needs. The way where we settle an Object-Oriented Software Simulator, concerning the logic and algorithm required to move the elevator between floors, giving the company a clear idea and leading it to choose the best design which satisfies this needs.