Logic Gates Simulator

1 Introduzione 3

1.1 Informazioni sul progetto 3

1.2 Abstract 3

1.3 Scopo 3

1.4 Pianificazione 4

2 Analisi 5

2.1 Analisi del dominio 5

2.2 Definizione dei requisiti 5

2.2.1 Analisi dei requisiti 5

2.2.2 Specifica dei requisiti 6

2.3 Analisi dei mezzi 7

2.3.1 JavaFX 7

2.3.2 MVC 9

2.4 Software 9

2.4.1 Netbeans 9

2.4.2 Scene Builder 9

2.5 Hardware 10

3 Progettazione 11

3.1 Architettura del sistema 11

3.1.1 Design dei dati 11

3.1.2 Design dell’architettura del sistema 11

3.1.3 Design delle interfacce 13

3.1.4 Design procedurale 13

3.2 Progettazione di dettaglio 13

3.2.1 Interfaccia utente 13

4 Implementazione 13

4.1 Logica 13

5 Test 15

5.1 Protocollo di test 15

5.2 Risultati test 15

5.3 Errori/limitazioni conosciute 15

6 Conclusioni 15

6.1 Sviluppi futuri 15

6.2 Considerazioni personali 15

7 Bibliografia 16

7.1 Bibliografia per articoli di riviste: 16

7.2 Bibliografia per libri 16

7.3 Bibliografia per siti web 16

8 Allegati 17

# Introduzione

## Informazioni sul progetto

**Allievi coinvolti**: Samuel Dias Da Silva, Cristiano Colangelo

**Docente responsabile**: Francesco Mussi

**Scuola**: Scuola Arti e Mestieri di Trevano, Sezione Informatica, Modulo 306

**Data di inizio e fine**: 18.11.2016 – 27.01.2017

## Abstract

Logic gates simulator is a desktop app created in Java / JavaFX that simulates the behavior of a logic circuit composed by logic gates and integrated circuits. The user has the ability to drag & drop the component in a designed area which holds the full circuitery and to connect the I/O pins between the logic gates through a connection (wire).

## Scopo

Lo scopo del progetto è quello di creare un simulatore di porte logiche a fini didattici.   
L’interfaccia grafica dell’applicativo è simile a quella di un generico editor nel quale è possibile comporre un circuito formato da porte logiche. L’utente, una volta composto il suo circuito, potrà inserire degli input che si tradurranno in un risultato finale. L’utente potrà poi concatenare i suoi componenti in modo da costruire circuiti anche complessi.

## Pianificazione

|  |
| --- |
| Figura 1: Diagramma di Gantt. |

# Analisi

## Analisi del dominio

Il contesto in cui l’utente dovrà lavorare sarà quello di una semplice applicazione desktop.   
Gli utenti dovranno avere delle conoscenze di base di elettronica, più specificatamente dovranno conoscere il funzionamento pratico di utilizzo della logica booleana in relazione a un circuito di porte logiche.

Gli utilizzatori di questo applicativo potranno essere sia appassionati che professionisti o studenti che vogliono capire il comportamento di alcuni circuiti ma anche il funzionamento della porte logiche. Rispettivamente, questi utilizzeranno il software sul proprio computer personale o aziendale.

Sul mercato vi sono già una vasta gamma di prodotti simili che permettono all’utente di simulare il funzionamento dei circuiti, sia gratuiti (spesso open source) che a pagamento.

Questo capitolo dovrebbe descrivere il contesto in cui il prodotto verrà utilizzato, da questa analisi dovrebbero scaturire le risposte a quesiti quali ad esempio:

* Quale è e come è organizzato il contesto in cui il prodotto dovrà funzionare? **OK**
* Come viene risolto attualmente il problema? **APPROFONDIRE**
* Chi sono gli utenti? Che bisogni hanno? Come e dove lavorano? **OK**
* Che competenze/conoscenze/cultura posseggono gli utenti in relazione con il problema? **FORSE OK**
* Esistono convenzioni/standard applicati nel dominio? **APPROFONDIRE**
* Che conoscenze teoriche bisogna avere/acquisire per poter operare efficacemente nel dominio? **COSA INTENDE? NOI PROGRAMMATORI CHE CONOSCENZE NECESSITIAMO?**

## Definizione dei requisiti

### Analisi dei requisiti

|  |  |
| --- | --- |
| **ID: REQ-001** | |
| **Nome** | Creazione di un simulatore di porte logiche |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |
| **Sottorequisiti** | |
| **001** | Il software deve permettere la simulazione del comportamento di un circuito composto da porte logiche |
| **002** | In funzione di uno o più input il circuito produce un output |
| **003** | Il software verrà prodotto sottoforma di applicazione desktop |

|  |  |
| --- | --- |
| **ID: REQ-002** | |
| **Nome** | Interazioni con le porte logiche |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |
| **Sottorequisiti** | |
| **001** | L’utente dovrà poter interagire solo con i pin di ingresso e mai con quelli d’uscita |
| **002** | I valori di ingresso vengono attivati/disattivati con un click del mouse sui pin delle porte logiche |
| **003** | È possibile effettuare un drag & drop di una porta logica fra una piccola finestra (denominata “toolbox”) e l’area di disegno |
| **004** | È possibile eliminare le porte logiche dal circuito |
| **005** | È possibile spostare liberamente le porte logiche nell’area di disegno |
| **006** | Le porte logiche potranno essere collegate insieme tramite dei fili, e i rispettivi valori dei pin di output verranno trasmessi al pin di input collegato |

|  |  |
| --- | --- |
| **ID: REQ-003** | |
| **Nome** | Componenti del circuito |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** |  |
| **Sottorequisiti** | |
| **001** | Sono presenti le porte logiche AND |
| **002** | Sono presenti le porte logiche OR |
| **003** | Sono presenti le porte logiche NOT |
| **004** | Sono presenti le porte logiche NAND |
| **005** | Sono presenti le porte logiche NOR |
| **006** | Sono presenti le porte logiche XOR |
| **007** | Sono presenti le porte logiche XNOR |

### Specifica dei requisiti

## Analisi dei mezzi

### JavaFX

**JavaFX** è una piattaforma software (integrata dalla versione 2.0 come libreria nativa di Java) per creare e distribuire applicazioni desktop e rich internet applications che possano funzionare su una gran varietà di devices. JavaFX è stato creato per rimpiazzare Swing come la standard GUI library per Java SE. JavaFX ha il supporto per desktop computer e web browsers operanti su Microsoft Windows, Linux e Mac OSX.

Inoltre, questo potente framework permette di modificare la grafica dei suoi componenti grazie al CSS.

FXML è un linguaggio di markup basato su XML che permette agli sviluppatori di creare delle interfacce grafiche in JavaFX separatamente dall’implementazione della logica del programma (vedi C#/WPF).   
Swing invece non ha mai offerto un approccio dichiarativo per creare delle interfacce grafiche.   
Il paradigma dichiarativo per la creazione delle UI è particolarmente comodo da utilizzare in relazione alla scene graph (rappresentazione di JavaFX della struttura gerarchica degli oggetti grafici), in quanto la scene graph appare meno astratta grazie a FXML.

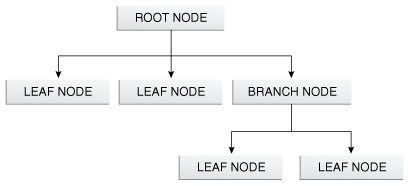
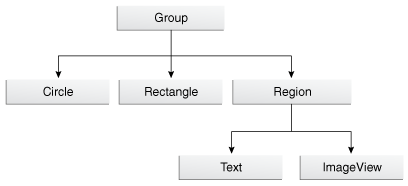
 

Figura 2 Scene Graph: Root, Branch, and Leaf Nodes Figura 3 Scene Graph: Specific Root, Branch, and Leaf Classes

FXML permette agli sviluppatori di creare e modificare UI anche molto complesse.

In conclusione, JavaFX è un framework davvero potente per creare applicazioni veloci e ricche di funzionalità. Ma non solo: JavaFX è più facile da comprendere di Swing e il codice da scrivere oltre che essere ben compensibile, è anche meno!



Figura 4 Esempio di file .FXML

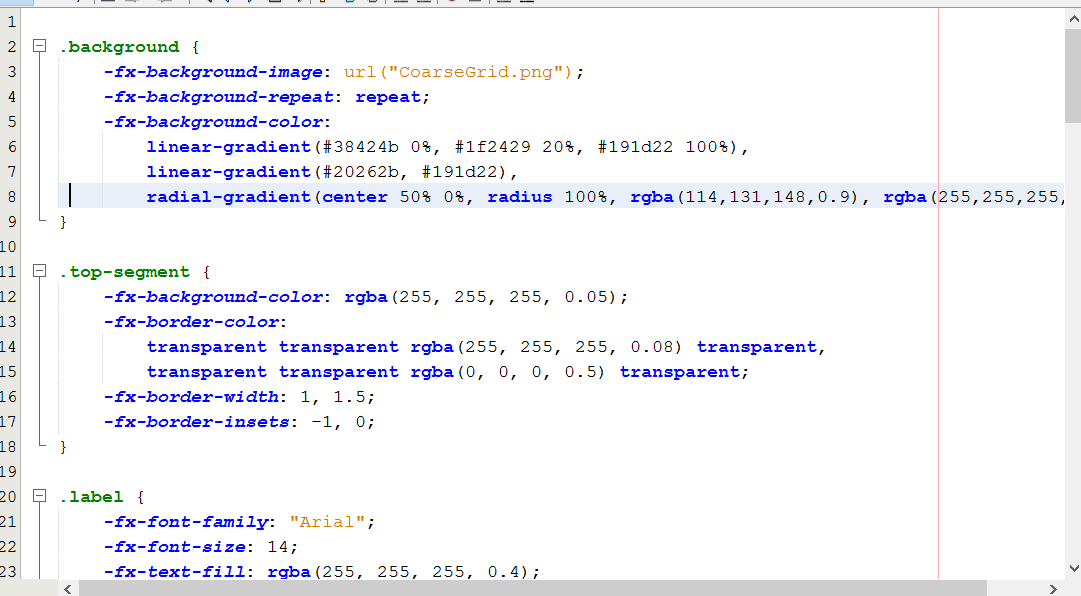


Figura 5 Esempio di file . CSS in JavaFX

**Approfondimenti**:

<https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-architecture.htm>

<http://www.dummies.com/programming/java/10-differences-between-javafx-and-swing/>

## Software

### Netbeans

Netbeans è un IDE scritto in Java. La piattaforma di NetBeans permette alle applicazioni di essere sviluppate partendo da un set di componenti software modulari chiamati moduli. Le applicazioni basate su NetBeans possono venire ampliate da numerosi plug-ins 3d party. Questo IDE è stato creato principalmente per lo sviluppo in Java, ma supporta un gran numero di altri linguaggi, in particolare PHP, C/C++ e HTML5.   
NetBeans è cross-platform e gira su Microsoft Windows, Mac OS X, Linux, Solaris e altre piattaforme compatibili con la JVM. Il team di NetBeans supporta attivamente il prodotto e mira ad aggiungere features su suggerimento dell’ampia community. Ogni release è preceduta da un tempo per il testing e il feedback ad opera della Community.

### Scene Builder

JavaFX Scene è un tool visuale per la creazione di layout che permette all’utente di disegnare rapidamente interfacce grafiche funzionanti su JavaFX (vedi C#/WPF) senza scrivere codice.   
L’utente può fare drag & drop dei componenti nell’area di lavoro, modificare le sue proprietà, aggiungere dello stile, aggiungere degli effetti grafici eccetera. Il codice FXML verrà generato programmaticamente.

Il risultato è un file FXML che può essere combinato con un progetto Java facendo un binding della UI alla logica dell’applicazione.   
Noi abbiamo integrato direttamente il tool in NetBeans (vedi allegato scenebuilder\_howto.md).

## Hardware

* CPU: Intel Core i7-6700HQ 2.6 GHz
* 8GB DDR4 Dual-Channel Memory

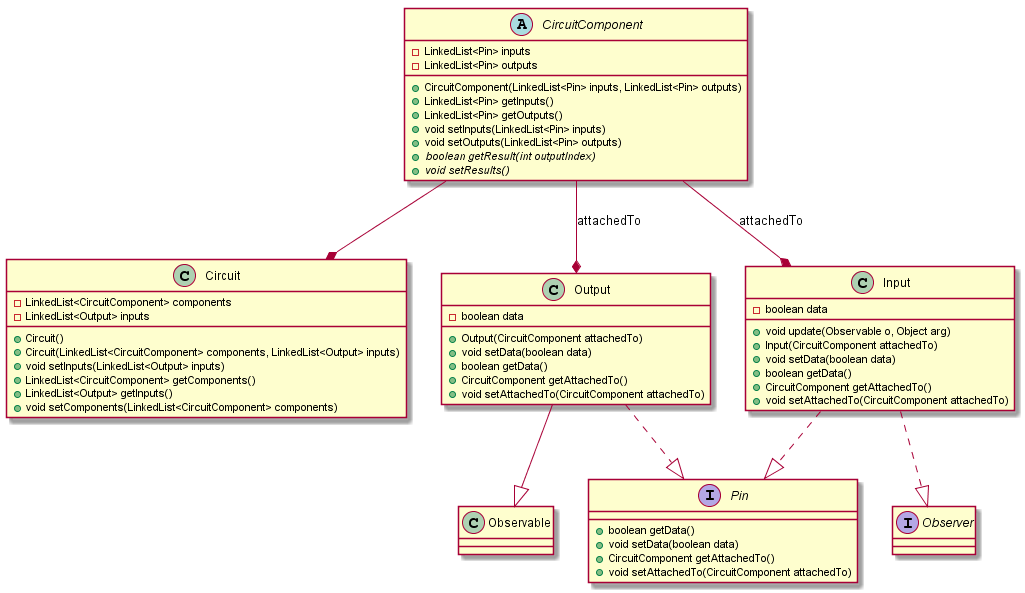
# Progettazione

## Architettura del sistema

### Design dei dati

Oltre ai dati di tipi primitivo, abbiamo utilizzato le LinkedList, sfruttando la sua dimensione dinamica per rappresentare i Pin di ingresso ed i Pin di uscita.

### Design dell’architettura del sistema



Abbiamo cercato di creare un architettura che potesse supportare il maggior numero di componenti, in modo tale che questa non si fermi solamente alle porte logiche, ma anche a componenti più avanzati, come ad esempio dei *FullAdder.*

Questa è l’architettura di base. In pratica, ogni circuito avrà un certo numero di componenti, che a loro volta possono avere due tipi di *Pin*: *Input* o *Output*.

L’interfaccia *Pin* descrive cosa dovrà avere un qualsiasi tipo di *Pin*, inoltre fa da *generalizzazione*.

La classe Input è quella che si occupa di gestire l’ingresso del componente. Questa conterrà due attributi, il primo di nome data e di tipo boolean, che sarà lo stato corrente del Pin (true o false), il secondo sarà il riferimento a quale circuito appartiene.

*Input* andrà ad implementare l’interfaccia *Observer*. Questo gli permette di dire alla porta logica a cui fa riferimento di andare a ricalcolare lo stato dell’uscita.

La classe *Output* è quella che si occupa di gestire l’uscita del *componente*. Questa conterrà, come la classe *Input*, due attributi, il primo di nome data e di tipo *boolean*, che sarà lo stato corrente del Pin (true o false), il secondo sarà il riferimento a quale circuito appartiene.

*Output* andrà ad estendere *Observable*. Questo gli permette di andare ad avvisare tutti gli *Observer* che lui è cambiato, come un messaggio *broadcast*.

*CircuitComponent* è la superclasse che dovrà essere estesa per creare un qualsiasi tipo di componente elettronico, che potrà avere un numero variabile di ingressi e di uscite. Questo viene fatto tramite due attributi, il primo si chiama input ed è di tipo *LinkedList* di genere Pin, il secondo si chiama output ed anch’esso è di tipo *LinkedList* di genere *Pin*.

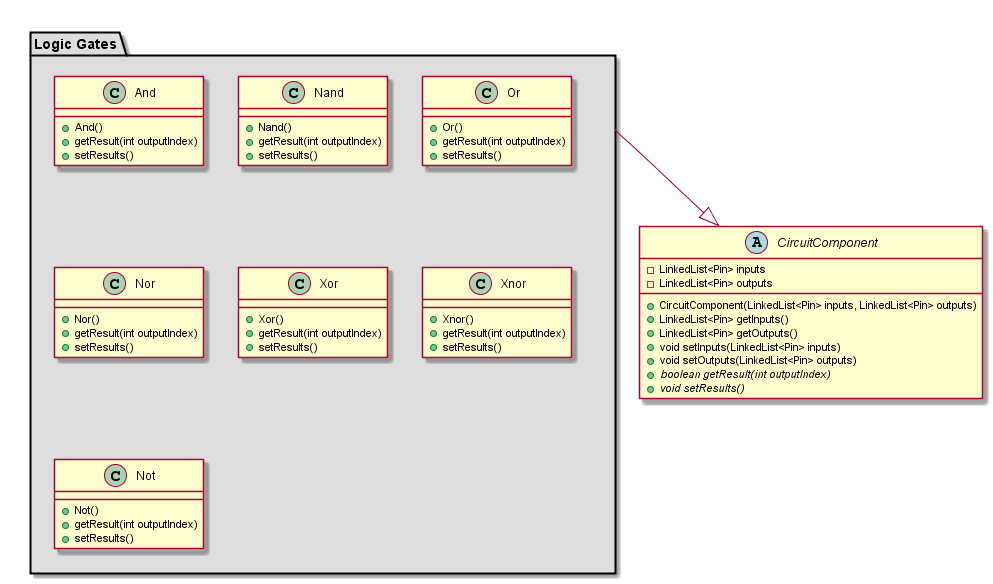
Vi sono due metodi *astratti* che sono *getResult,* il quale, tramite un parametro che fa riferimento all’indice degli *output*, permette di scrivere la funziona booleana del *Pin* di tipo *Output*, il secondo è *setResults* il quale va a ricalcolare, tramite il metodo *getResult,* e settare tutti i *Pin* di tipo *Output*.

Questi due metodi dovranno essere riscritti all’interno del componente che si vuole creare.

*Circuit* è la classe che permette di gestire e simulare un circuito tramite una *LinkedList* di genere *CircuitComponent*.

#### Esempio di estendibilità con le porte logiche:

Da come ho descritto in precedenza, per creare degli altri componenti bisogna andare a creare una classe che vada ad estendere la superclasse astratta *CircuitComponent.* Di conseguenza bisognerà fare qualcosa di simile all’UML seguente:

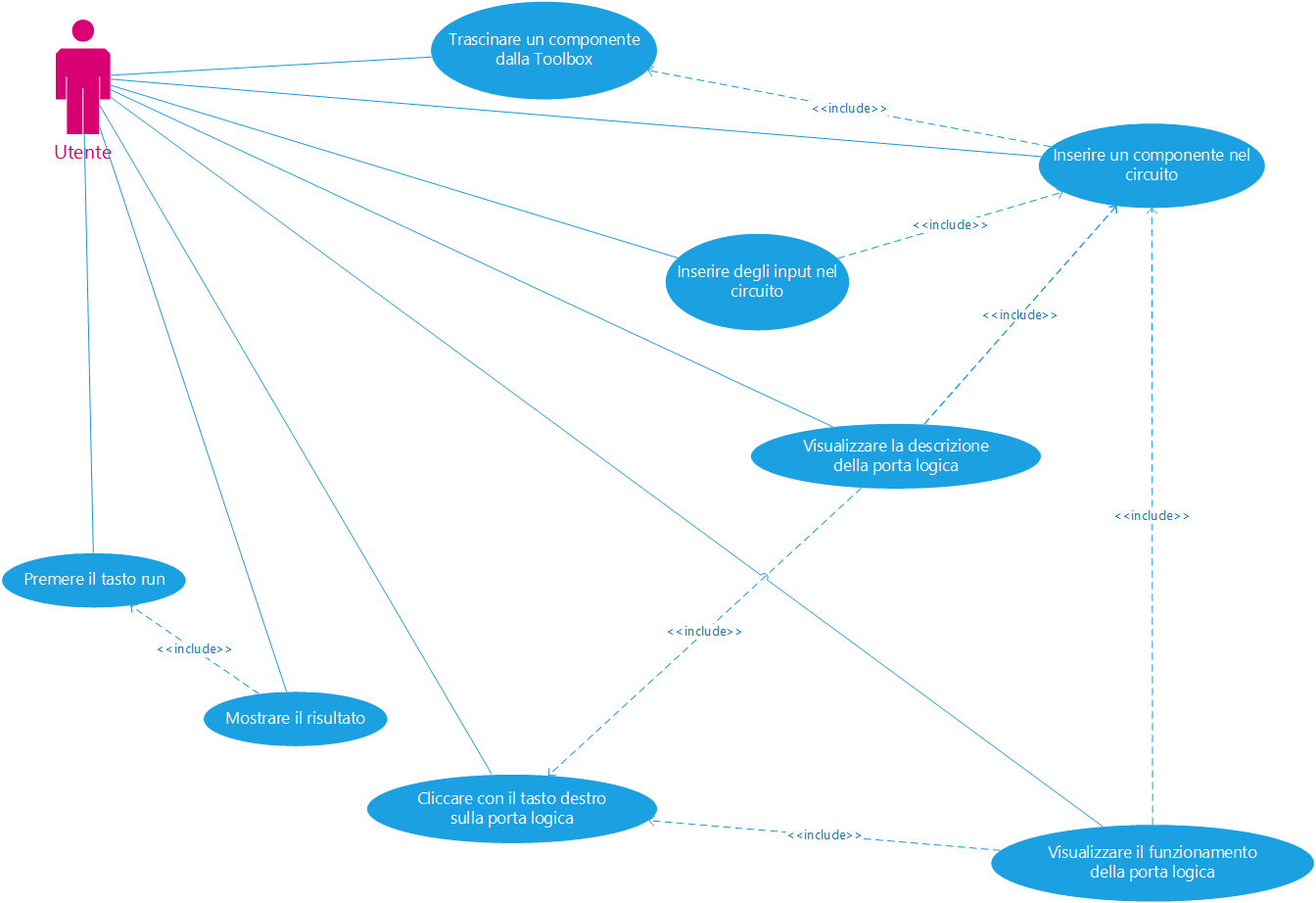


Come si può vedere, ogni porta logica avrà il proprio costruttore per essere istanziato, il metodo sovrascritto *getResult* che tramite un indice indica il funzionamento del relativo *Pin* di tipo *Output*, e il metodo *setResults* che andrà a settare tutte le uscite a dipendenza della logica di esse, facendo quindi riferimento al metodo *getResult.*

### Design delle interfacce

Descrizione delle interfacce interne ed esterne del sistema e dell’interfaccia utente. La progettazione delle interfacce è basata sulle informazioni ricavate durante la fase di analisi.

### Design procedurale



## Progettazione di dettaglio

### Interfaccia utente

# Implementazione

## Logica

Per iniziare, ho implementato la interfaccia *Pin*, definendo i suoi *metodi* ed i relativi *parametri*. In seguito ho scritto le due classi che implementano la interfaccia *Pin*: *Input* e *Output.*

Queste due classi implementano quello che viene definito *Observer Pattern.* Questo permette di avere degli oggetti *Osservatori* (in inglese *Observer*) e degli oggetti *Osservabili* (in inglese *Observable*). Quando gli oggetti *Osservabili* cambiano il proprio stato, questi avviseranno tutti i loro *Osservatori*, come se fosse un messaggio in *broadcast*. Starà agli oggetti *Osservabili* definire il loro comportamento ad un cambiamento del relativo *Osservabile*.

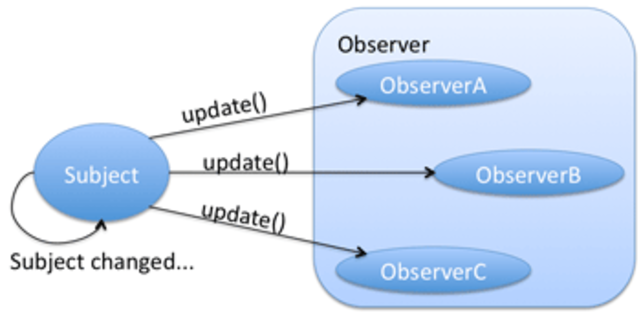
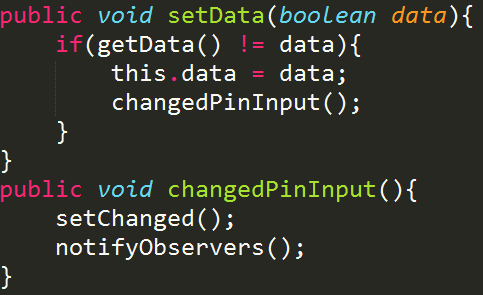


Figura 6: Observer pattern

Da questo schema si può capire un po’ meglio il funzionamento dell’*Observer pattern*.

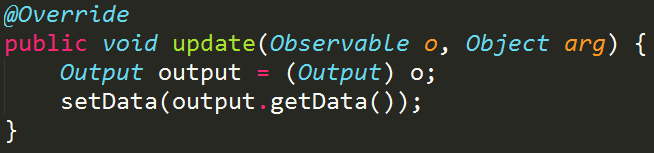
In questo caso gli *Observer* saranno i *Pin* di tipo *Input*, mentre gli *Observable* saranno i *Pin* di tipo *Output*.

Il messaggio di broadcast verso i *Pin* di tipo *Input* avverrà quando lo stato dell’output della porta logica a cui è assegnato cambia. Per fare ciò ho scritto le seguenti righe di codice



In pratica, quando si setta il dato del Pin, questo richiamerà il metodo *changedPinInput*, il quale metterà un flag per dire che è cambiato. In seguito avviserà tutti gli *Input* osservatori tramite il metodo *notifyObservers*.

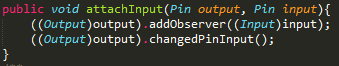
La classe *Input,* dovrà implementare il metodo *update(Observable o, Object arg)* dove il primo parametrò sarà l’oggetto che è andato a notificare *l’Observer*, il secondo sarà un argomento facoltativo*.*



L’oggetto Observable verrà castato in *Output,* ed inseguito si andrà ad aggiornare il dato dell’input tramite il *setter* di *data,* con il *data* del Pin di *output*.

In seguito ho implementato la superclasse astratta *CircuitComponent*. Questa ha due attributi di tipo *LinkedList* di genere *Pin*, che saranno i corrispettivi *input* e *output,* sono stati implementati inoltre i due relativi getters e setters ed il costruttore per la inizializzazione di un componente. Oltre a questo non vi è molto in questa classe.

La classe *Circuit* ha due attributi di tipo *LinkedList*; la prima del genere *CircuitComponent* e la seconda del genere *Output.* Questa classe fornisce due metodi importanti: *attachInput* e *deattachInput.* Questi servono a collegare o scollegare quelli che sono dei cavi invisibili tra *Output* ed *Input,* in pratica aggiungono/tolgono l’*observer* dall’output.



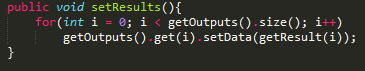
C:\Users\Dias\AppData\Local\Microsoft\Windows\INetCache\Content.Word\deattachInput.png

Quando si vanno a collegare i due Pin, quello di output andrà a richiamare il metodo *changedPinInput,* in modo tale da cambiare lo stato del componente a cui appartiene il *Pin* di *Input.*

Per quanto concerne le porte logiche, seguono tutte lo stesso schema, per questo motivo spiegherò il tutto utilizzando come esempio la porta logica AND.

Ho creato il costruttore, che andrà ad aggiungere tutti gli input e gli output di cui ha bisogno, (ad esempio per la porta logica AND, gli input saranno due mentre gli output sarà solo uno). Dopodiché ho sovrascritto il metodo *getResult,* e siccome ha un solo *output,* ho descritto il funzionamento di solamente un solo *Pin* tipo di *output.*

Il metodo setResults rimane sempre uguale, in pratica, tramite un ciclo for, richiama il metodo *getResult* della porta logica, passandogli come parametro l’indice che incrementa.



# Test

## Protocollo di test

|  |  |
| --- | --- |
| Test case | TC-01 |
| Nome | REQ-003 |
| Descrizione | Testare le porte logiche |
| Prerequisiti | JUnit |
| Procedura | 1. Istanziare una porta logica 2. Testare che tutte le combinazioni della porta logica tramite il metodo testCombinations(portaLogica) 3. Testare che la prima combinazione funzioni, richiamare il metodo setFirstCombination(portaLogica) e fare assertEquals(false, g.getResult(0)) 4. Testare che tutte le combinazioni della porta logica tramite il metodo testCombinations(portaLogica) 5. Testare che la seconda combinazione funzioni, richiamare il metodo setSecondCombination(portaLogica) e fare assertEquals(false, g.getResult(0)) 6. Testare che tutte le combinazioni della porta logica tramite il metodo testCombinations(portaLogica) 7. Testare che la terza combinazione funzioni, richiamare il metodo setThirdCombination(portaLogica) e fare assertEquals(true, g.getResult(0)) 8. Testare che tutte le combinazioni della porta logica tramite il metodo testCombinations(portaLogica) 9. Testare che la quarta combinazione funzioni, richiamare il metodo setFourthCombination(portaLogica) e fare assertEquals(false, g.getResult(0)) |
| Risultati attesi | Passed |

|  |  |
| --- | --- |
| Test case | TC-02 |
| Nome | REQ-001 |
| Descrizione | Test di un circuito composto da tre porte logiche |
| Prerequisiti | JUnit |
| Procedura | 1. Istanziare un Circuit tramite il suo costruttore 2. Istanziare due CircuitComponent ed istanziarli tramite il costruttore And. Questi verranno chiamati and1 e and 2. 3. Istanziare un CircuitComponent ed istanziarlo tramite il costruttore Xor, questa verrà chiamata xor1. 4. Aggiungere i tre CircuitComponent creati in precedenza a Circuit tramite il suo metodo addCircuitComponent e come parametro mettere il componente. 5. Collegare le tre porte logiche nel Circuit tramite il suo metodo attachInput, come argomenti inserire and1.getOutputs().get(0) e xor1.getInputs().get(0) la seconda volta inserire and2.getOutputs().get(0) e xor1.getInputs().get(1) . 6. Utilizzare assertEquals(false, xor1.getOutputs().get(0).getData()) |
| Risultati attesi | Passed |

|  |  |
| --- | --- |
| Test case | TC-03 |
| Nome | REQ-001 |
| Descrizione | Esempio del test di una porta logica, in questo caso la porta AND. |
| Prerequisiti | JUnit |
| Procedura | 1. Istanziare un Circuit tramite il suo costruttore 2. Istanziare due CircuitComponent ed istanziarli tramite il costruttore And. Questi verranno chiamati and1 e and 2. 3. Istanziare un CircuitComponent ed istanziarlo tramite il costruttore Xor, questa verrà chiamata xor1. 4. Aggiungere i tre CircuitComponent creati in precedenza a Circuit tramite il suo metodo addCircuitComponent e come parametro mettere il componente. 5. Collegare le tre porte logiche nel Circuit tramite il suo metodo attachInput, come argomenti inserire and1.getOutputs().get(0) e xor1.getInputs().get(0) la seconda volta inserire and2.getOutputs().get(0) e xor1.getInputs().get(1) . 6. Cambiare lo stato degli input della and1 con and1.getInputs().get(0).setData(false) e and1.getInputs().get(1).setData(true) 7. Utilizzare assertEquals(false, xor1.getOutputs().get(0).getData()) |
| Risultati attesi | Passed |

|  |  |
| --- | --- |
| Test case | TC-03 |
| Nome | REQ-001 |
| Descrizione | Esempio del test di una porta logica, in questo caso la porta AND. |
| Prerequisiti | JUnit |
| Procedura | 1. Istanziare un Circuit tramite il suo costruttore 2. Istanziare due CircuitComponent ed istanziarli tramite il costruttore And. Questi verranno chiamati and1 e and 2. 3. Istanziare un CircuitComponent ed istanziarlo tramite il costruttore Xor, questa verrà chiamata xor1. 4. Aggiungere i tre CircuitComponent creati in precedenza a Circuit tramite il suo metodo addCircuitComponent e come parametro mettere il componente. 5. Collegare le tre porte logiche nel Circuit tramite il suo metodo attachInput, come argomenti inserire and1.getOutputs().get(0) e xor1.getInputs().get(0) la seconda volta inserire and2.getOutputs().get(0) e xor1.getInputs().get(1) . 6. Cambiare lo stato degli input della and1 con and1.getInputs().get(0).setData(true) e and1.getInputs().get(1).setData(true) 7. Utilizzare assertEquals(true, xor1.getOutputs().get(0).getData()) |
| Risultati attesi | Passed |

## Risultati test

|  |  |
| --- | --- |
| Test Case | Esito |
| TC-01 | Passato |
| TC-02 | Passato |
| TC-03 | Passato |

## Errori/limitazioni conosciute

La logica del programma non permette di avere dei collegamenti che vanno dall’output di un circuito all’input dello stesso. Questo causa una *StackOverflowException,* per il semplice fatto che i due continuano a cambiare, incrementando ed esaurendo la memoria della *callStack.*

Questo problema si pone anche quando vi sono molteplici (sulle centinaia) porte logiche che continuano a cambiare il loro stato.

# Conclusioni

## Sviluppi futuri

In futuro sarebbe interessante continuare questo progetto. Siccome abbiamo creato quella che è un’architettura che può essere sviluppata ulteriormente, sarebbe interessante mettere altri tipi di circuiti logici, circuiti integrati ed aggiungere inoltre altri tipi di funzionalità, come ad esempio il salvataggio di un circuito.

## Considerazioni personali

# Bibliografia

## Bibliografia per articoli di riviste:

1. Cognome e nome (o iniziali) dell’autore o degli autori, o nome dell’organizzazione,
2. Titolo dell’articolo (tra virgolette),
3. Titolo del periodico (in italico),
4. Volume o tomo, Numero ed. eventualmente parte,
5. Prima ed ultima pagine dell’articolo,
6. Data di pubblicazione.

**Esempio:**

* …

## Bibliografia per libri

1. Cognome e nome (o iniziali) dell’autore o degli autori, o nome dell’organizzazione,
2. Titolo del libro (in italico),
3. ev. Numero di edizione,
4. Luogo di pubblicazione (città),
5. Nome dell’editore,
6. Anno di pubblicazione,
7. ev. Prima ed ultima pagina del riferimento.

**Esempio:**

* Kernighan Brian W. e Ritchie Dennis M., *Linguaggio C*, seconda edizione, Milano: Jackson Libri, 1989.

## Bibliografia per siti web

* <https://en.wikipedia.org/wiki/JavaFX>, *JavaFX* , 09/12/2016
* <http://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html>,  
  *Java FX SceneBuilder*, 09/12/2016
* <http://docs.oracle.com/javafx/2/scenegraph/jfxpub-scenegraph.htm>, *Working with the JavaFX Scene Graph*, 09/12/2016
* <http://gluonhq.com/wp-content/uploads/2015/02/SceneBuilderLogo.png>, *SceneBuilderLogo.png*, 09/12/2016
* <http://mikecann.co.uk/wp-content/uploads/2009/12/javafx_logo_color_1.jpg>, *javafx\_logo\_color\_1.jpg,* 09/12/2016
* <http://www.zwodnik.com/media/cache/7d/ac/7dac82fd78702aeca1a98b1d1feee42e.png>, netbeansLogo.png, 09/12/2016
* <https://en.wikipedia.org/wiki/NetBeans>, *Netbeans,* 09/12/2016
* <http://tidalwave.it/media/diagrams/20120808/MVC-class-diagram.png>, *MVC-class-diagram.png,* 09/12/2016

# Allegati

Elenco degli allegati, esempio:

* Allegato A: Codici sorgente
* Allegato B: Istruzioni di installazione
* Allegato C: Manuali di utilizzo
* Documentazione di prodotti di terzi
* …