Abhandlung MVC- und Observer-Pattern

Christian Feier

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Inhaltsverzeichnis

1 Einleitung

In der Programmierung ist es gerade wichtig das der Sourcecode gut erweiterbar und wiederverwendbar ist, dazu ist es notwendig sauber und struktuiert zu programmieren.
Um dies zu erreichen werden im folgenden 2 wichtige Werkzeuge vorgestellt, zum einen das MVC-Pattern, und zum anderen das Observer-Pattern. Um diese beiden Werkzeuge noch etwas näherzubringen wird es am Ende auch noch ein kleines Beispiel in Form eines Addierers zu geben. Anhand des Beispiels werden wir auch sehen, dass das Observer-Pattern zum Beispiel dazu genutzt werden kann um das MVC-Pattern zu realisieren.

2 Das MVC-Pattern

2.1 Was ist das MVC-Pattern eigentlich?

MVC steht für **M**odel-**V**iew-**C**ontroller, wobei jede dieser 3 Komponenten individuelle Aufgaben übernimmt. Grafisch kann man sich das MVC folgendermaßen vorstellen:

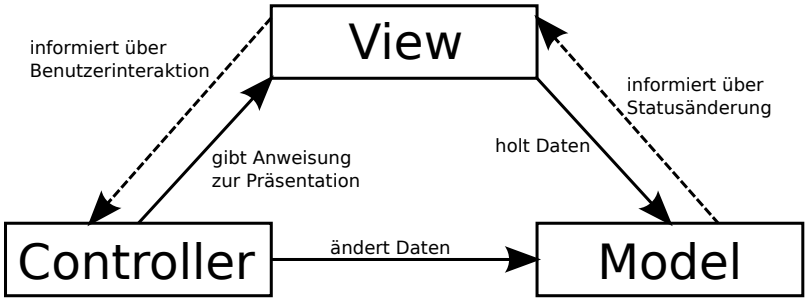

Abbildung 2.1: MVC-Aufbau

Eine genauere Erklärung zur Grafik kommt später in Abschnitt 2.2.

2.1.1 Bedeutung der Komponenten

Model:

Das Model enthält zum einen die darzustellenden Daten, und dient zum anderen dazu benötigte Berechnungen bzw. Verarbeitungen der Daten durchzuführen. Beispielsweise kann das Model Datenbankzugriff haben und benötigte Daten daraus verwalten und zur Verfügung stellen. Ebenfalls können im Model Operationen auf diese Daten durchgeführt werden.

View:

Die View dient zur Präsentation der im Model abgelegten Daten und ist zur Entgegennahme von Benutzerinteraktionen zuständig, die Weiterverarbeitung der Benutzerinteraktion gehört aber nicht zum Aufgabenbereich der View, sondern zu dem des Controllers.

Controller:

Der Controller dient zur Steuerung des Programms, er reagiert auf Benutzerinteraktionen und ruft gegebenenfalls Methoden des Models oder der View auf. Der Controller selbst entscheidet also nur anhand der Benutzerinteraktion welche Daten manipuliert werden sollen, tut dies aber nicht selber, sondern gibt den Befehl ans Model weiter, oder sagt der View welche Daten wo angezeigt werden sollen.
Die Reaktion auf Benutzerinteraktionen geschieht hier meist über Handler oder Callbackfunktionen. Wie man diese indirekte Verbindung beispielsweise in Java umsetzen könnte wird später in Abschnitt 2.3 gezeigt.

2.2 Wer kennt wen?

Betrachten wir nochmal die Abbildung 2.1 weiter oben. Nun eine kurze Erklärung.
Die durchgezogenen Verbindungen drücken eine direkte Verbindung zur jeweiligen Komponente aus. Eine gestrichelte Linie drückt keine direkte Verbindung aus, sondern zeigt lediglich eine indirekte Verbindung, beispielsweise durch das noch später in Kapitel 3 besprochene Observer-Pattern.
Man könnte sich nun Fragen, wieso eigentlich indirekte Verbindungen? Wieso keine direkten? Die Antwort darauf ist eigentlich recht simpel, eine direkte Verbindung würde lediglich zu stande kommen, wenn Objekt A ein anderes Objekt B direkt ansprechen kann, also eine Instanz vom Objekt B in Objekt A als beispielsweise privates Attribut vorhanden ist. Nun würde das eine weitere Abhängigkeit zwischen den Objekten bedeuten, und das wiederrum senkt die Wiederverwertbarkeit. Daher versucht man sofern es effizient machbar ist direkte Verbindungen zu vermeiden, und zieht indirekte vor.

Wie man sieht kennt einzig allein das Model niemanden direkt, es gibt also im Model keine Instanz einer andere Komponente bzw. Möglichkeit direkt mit einer anderen Komponente zu sprechen. In einer guten MVC Implementierung kann man das Model direkt aus dem Programm auslagern, und es muss direkt funktionieren. Hätte man direkte Verbindungen zu anderen Komponenten, so wäre dies nicht möglich. Es existiert daher lediglich eine indirekte Verbindung zur View, hierüber kann das Model einer View mitteilen, das ein bestimmtes Ereignis ausgelöst wurde, ohne die View explizit zu kennen. Nehmen wir als Beispiel mal an wir haben eine View mit einer Tabelle mit Einträgen. Nun kann der User einen dieser Einträge über die Oberfläche durch Betätigung eines Buttons löschen. Wird dieser Button gedrückt, würde der Controller den Befehl delete ans Model senden, das Model würde nun den Eintrag löschen und könnte beispielsweise eine Notification an die View senden, das der Eintrag gelöscht wurde.

Beim Controller sieht das anders aus. Der Controller kennt sowohl die View, als auch das Model. Um Daten im Model zu manipulieren benötigt der Controller eine direkte Verbindung zum Model, dies geschieht normal einfach durch eine Instanz des Models die im Controller als beispielsweise privates Attribut vorhanden ist. Analog speichert man auch eine Instanz der View im Controller, diese wird beispielsweise dazu benötigt, um zusätzlich benötige Informationen aus der View zu bekommen, wenn eine bestimmte Benutzerinteraktion wie das drücken eines Buttons ausgeführt wurde.

Und nun zur View. Die View kennt das Model direkt, und indirekt den Controller. Eine direkte Verbindung zum Model ist notwendig, um Daten in der View aktualisieren zu können. Da das Model die View nicht kennt, muss sich die View die benötigten Daten direkt aus dem Model selber holen können müssen.
Die indirekte Verbindung zum Controller dient dazu, um Benutzerinteraktion dem Controller mitzuteilen. Wird beispielsweise ein Event wie das drücken eines Buttons in der View ausgelöst, so wird der Controller über diese indirekte Verbindung über die Benutzerinteraktion informiert. Eine mögliche Umsetzung dieser indirekten Verbindung zwischen View und Controller in Java wird in Abschnitt 2.3 besprochen.

Ein Programm kann durchaus mehr als nur ein Model, Controller oder View haben. Beispielsweise ist jedes Fenster eines Programms eine neue View, und jede View hat ihren eigenen Controller, und im Normalfall hat diese View auch noch ein Model dazu, um eventuell anfallende Eingabeverarbeitungen durchzuführen bzw. Informationen zwischenzuspeichern. Hierbei muss aber nicht jeder Controller bzw. jede View ihr eigenes Model haben, es ist durchaus normal das sich mehrere Views ein Model teilen.
Auf diese Weise erreichen wir einen modularen Aufbau, es ist also ohne viel Aufwand möglich eine Komponente wie beispielsweise ein Model oder eine View aus einem Projekt zu nehmen und in ein anderes Projekt einzubauen, da lediglich der Controller und eventuell die Komponenten mit einer direkten Verbindung angepasst werden müssen.

2.3 Mögliche Umsetzung der indirekten Verbindung View zu Controller

Eine Möglichkeit die indirekte Verbindung zwischen der View und dem Controller ist die, das der Controller direkt als Listener genutzt wird, also das Listener Interface implementiert und sich selber bei der Initialisierung der View als Listener der einzelnen Komponenten einsetzt.
Um diese Möglichkeit etwas näherzubringen betrachten wir kurz das folgende Codebeispiel:

```
1  import java.awt.event.ActionListener;
2  import javax.swing.JButton;
3  import javax.swing.JFrame;
4
5  public class myView extends JFrame{
6
7      // gui components
8      private JButton myButton;
9      // ....
10
11     /**
12      * init gui components
13      */
14     public void init()
15     {
16         /* do something, like init myButton */
17     }
18
19     /* ******************
20      *  Getter & Setter *
21      * ******************
22      */
23     /**
24      * set ActionListener
25      */
26     public void setActionListener(ActionListener l)
27     {
28         myButton.addActionListener(l);
29     }
30
31     /**
32      * gets myButton
33      */
```

```
34      public JButton getMyButton()
35      {
36          return myButton;
37      }
38  }
```

Listing 1: Beispiel View

```
1   import java.awt.event.ActionEvent;
2   import java.awt.event.ActionListener;
3
4   public class myController implements ActionListener
5   {
6       // myView instance
7       private myView window;
8
9       /**
10       * Constructor
11       */
12      public myController()
13      {
14          /* do something, like init window */
15      }
16
17      /*
18       * (non-Javadoc)
19       * @see java.awt.event.ActionListener#actionPerformed
20         (java.awt.event.ActionEvent)
21       */
22      @Override
23      public void actionPerformed(ActionEvent e)
24      {
25          // myButton pressed
26          if(e.getSource() == window.getMyButton())
27              myButtonPressed();
28      }
29
30      /**
31       * called if myButton got pressed
32       */
33      private void myButtonPressed()
34      {
35          /* do something */
36      }
37  }
```

Listing 2: Beispiel Controller

Zuerst ein paar Worte zu der View in Listing 1. Als Beispiel dient uns hier der JButton myButton aus Zeile 8, welcher durch `init()` initialisiert wird. Wichtig für diese Variante sind nur die `setActionListener(ActionListener l)` und `getMyButton()` Methoden. Durch `setActionListener(ActionListener l)` können wir den GUI Elementen einen ActionListener zuordnen. Und durch die Getter Methoden kommen wir an die aktuelle Instanz der Kom-

ponenten.

Betrachten wir nun den Controller in Listing 2 so sieht man das dieser in Zeile 4 den ActionListener implementiert und dadurch die Methode `acitonPerformed(ActionEvent e)` in Zeile 23 implementiert werden muss. Da der Controller nun sozusagen selber ein ActionListener ist, kann er den GUI Elementen der View als ActionListener zugeordnet werden.
So nun werden schonmal alle Benutzerinteraktionen der View an den Controller weitergeleitet, wird beispielsweise auf den JButton myButton gedrückt, so wird die Methode `actionPerformed(ActionEvent e)` des zugeordneten ActionListeners, also unserem Controller ausgelöst. Nun muss noch herausgefunden werden welcher Button nun gedrückt wurde, da alle GUI Komponenten beim auslösen eines ActionEvents die Methode `actionPerformed(ActionEvent e)` des Controller auslösen würden. Hierzu benötigen wir erstmal die Source des Events, diese erhalten wir mit `e.getSource()`, wobei `e` das ActionEvent sei, siehe dazu Zeile 26 in Listing 2. Nun können wir die Source des Events mit den einzelnen Komponenten abgleichen, daher wurden auch die Getter in der View benötigt. Hier dient als Beispiel wieder Zeile 26. Wird nun der Button myButton gedrückt, so würde das if Statement in Zeile 26 zu true ausgewertet werden, und die Methode `myButtonPressed()` ausgeführt.

Auf diese Weise haben wir nun die View mit dem Controller indirekt verbunden, so das der Controller sämtliche Benutzerinteraktionen mit der View direkt weitergeleitet bekommt und verarbeiten kann. Natürlich funktioniert diese Variante auch mit anderen Listenern wie beispielsweise dem `ListSelectionListener(ListSelectionEvent e)`. Ein Controller kann durch Implementierung verschiedener Interfaces auch als mehrere verschiedene Listener agieren.

Direkte Verbindungen kann man ermöglichen, indem man einfach eine Instanz des jeweiligen Objekts im anderen Objekt ablegt, beispielsweise hat der Controller eine direkte Verbindung zur View, da im Controller eine Instanz der View als privates Attribut vorhanden ist, dieses findet man in Listing 2 in Zeile 7.

3 Das Observer-Pattern

3.1 Was ist das Observer-Pattern eigentlich?

Das Observer Pattern ist ein Design Pattern um verschiedene Klassen durch indirekte Verbindungen zu verbinden. Der große Vorteil hieran liegt, das keine direkten Verbindungen benötigt werden, und somit die Kopplung, also die Abhängigkeit der Klassen untereinander möglichst gering gehalten wird, und somit die Wiederverwertbarkeit deutlich steigt.
Wie man das Observer-Pattern umsetzt soll das nachfolgende UML-Diagramm verdeutlichen. Vorerst erscheint dies eventuell etwas verwirrend, aber im späteren Verlauf wird diese Verwirrung hoffentlich beseitigt:

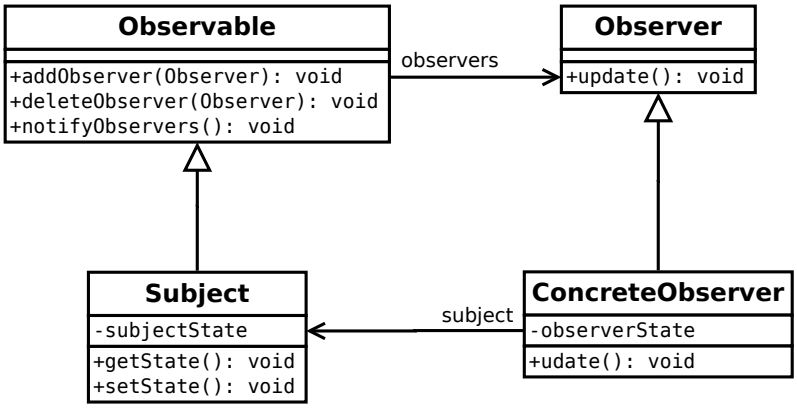

Abbildung 3.2: UML Diagramm zum Observer-Pattern

Salopp gesagt können sich diverse Klassen als Observer bei einer anderen Klasse S registrieren. Tritt in S eine Änderung auf, informiert S alle Observer über die Änderung, quasi ein Broadcast an alle Observer. Dies ist vergleichbar mit einem RSS-Feed. Alle Nutzer registrieren bzw. abonnieren einen RSS-Feed und bekommen bei einer Änderung der Seite über den Feed eine Benachrichtigung. So verhält es sich hier auch, wobei die Observer die Abonnenten, und das Subject die Seite ist.

3.1.1 Was bedeuten die einzelnen Komponenten nun?

Im folgenden nehme ich bezug auf Java, das heisst ich werde Komponenten die in Java als Interface gehandhabt würden, auch hier so verwenden als wären sie eins. In anderen Sprachen die Multivererbung unterstützen wie beispielsweise Python wären das eher abstrakte Klassen.

Subject:
Das Subject ist die Klasse die beobachtet werden soll. Das Subject informiert bei Bedarf alle Observer.
Erinnern wir uns nochmal ans MVC-Pattern, an die Klassen View und Controller. Will man beispielsweise eine indirekte Verbindung von View zu Controller, so wäre in diesem Beispiel die View das Subject, welches beobachtet wird, und der Controller der Observer, welcher beobachtet.

Observable:
Das Subject erbt von Observable. In der Observable Klasse werden alle Observer gespeichert. Zusätzlich befindet sich in der Observable Klasse auch die Methode `notifyObservers()`, welche alle registrierten Observer informiert.

ConcreteObserver:
Der ConcreteObserver ist ein beim Subject registrierter Observer, informiert das Subject seine Observer, so wird im ConcreteObserver die Methode `update()` aufgerufen.

Observer:
Der Observer ist ein Interface das vom ConcreteObserver implementiert wird, es stellt ihm die Methode `update()` bereit.

3.2 Wie funktioniert das Pattern intern?

Java bietet bereits eine Klasse Observable, zu finden in `java.util.Observable`, und ebenfalls auch ein Observer Interface, zu finden in `java.util.Observer`.
Zum besseren Verständnis wie das Observer Pattern funktioniert, besprechen wir in diesem Abschnitt aber noch kurz eine kleine vereinfachte Implementierung der 4 Komponenten. Hierzu betrachten wir zuerst das Interface `MyObserver`, welcher im UML-Diagramm den Observer repräsentiert:

```
1  public interface MyObserver
2  {
3      /**
4       * called whenever an Observer got a notification
5       */
6      public void update(Object arg);
7  }
```

Listing 3: Beispiel eines Observer Interfaces

Im Prinzip stellt dieses Interface die `update()` Methode zur Verfügung, und sorgt dafür das wir den ConcreteObserver als Observer beim Subject registrieren können.

Weiter gehts mit der Klasse `MyObservable` die im UML Diagramm die Komponente Observable repräsentiert:

```
1  import java.util.ArrayList;
2
3  public class MyObservable
4  {
5      // list to save all registered Observers
6      private ArrayList<MyObserver> observers = new ArrayList<MyObserver>();
7
8      /**
9       * adds an Observer
10      *
11      * @param newObserver
```

```
12          *                       Observer to add
13          */
14         public void addObserver(MyObserver newObserver)
15         {
16             observers.add(newObserver);
17         }
18
19         /**
20          * deletes an Observer
21          *
22          * @param newObserver
23          *                       Observer to delete
24          */
25         public void deleteObserver(MyObserver newObserver)
26         {
27             observers.remove(newObserver);
28         }
29
30         /**
31          * notifies all registered Observers
32          *
33          * @param arg
34          *              subject specific argument
35          */
36         public void notifyObservers(Object arg)
37         {
38             for(MyObserver obs: observers)
39                 obs.update(arg);
40         }
41 }
```

Listing 4: Beispiel einer Observable Klasse

Im Prinzip werden hier alle Obsever verwaltet, da des Subject davon erbt, hat es zugriff auf all diese Methoden und kann somit die registrierten Register verwalten. Gespeichert werden die Observer hier einfach in einer ArrayList. Will man die Observer informieren, geschieht dies über die Methode `notifyObservers(Object arg)`, wobei arg ein hier hinzugefügter Paramter ist, der eventuell für den Observer benötigte Daten enthält. Wofür man diesen Parameter nutzen können besprechen wir dann später in Abschnitt 3.3.1, wenn es um die verschiedenen Update Möglichkeiten geht.

Nun gehts weiter zur Klasse `Subject`, welche die Komponente Subject repräsentiert.

```
1  public class Subject extends MyObservable
2  {
3      /**
4       * Constructor
5       */
6      public Subject()
7      {
8          super();
9      }
10
```

```
11      /**
12       * notifies all Observers
13       */
14      public void doSomething()
15      {
16          notifyObservers("");
17      }
18  }
```

Listing 5: Beispiel einer Subject Klasse

hier ist erstmal relevant das man die Observable Komponente initialisiert, dies geschieht hier in Zeile 8. Und desweiteren das informieren der Observer, dies geschieht hier durch aufrufen der Methode `notifyObservers()`, es wird nun in jedem Observer die Methode `update(Object arg)` aufgerufen. Als Parameter übergeben wir der einfachheithalber einen leeren String. Mehr dazu wie gesagt dann in Abschnitt 3.3.1.

Und zuletzt noch die Klasse `ConcreteObserver`, welche den ConcreteObserver repräsentiert:

```
1   public class ConcreteObserver implements MyObserver
2   {
3       /*
4        * (non-Javadoc)
5        * @see MyObserver#update(java.lang.Object)
6        */
7       @Override
8       public void update(Object arg)
9       {
10          System.out.println("something changed");
11      }
12  }
```

Listing 6: Beispiel es ConcreteObserver

hier kann noch viel mehr drin stehen, aber in erster Linie gehts uns hier nur um die vom Interface vorgegebene Methode `update()`. Informiert ein Subject seine Observer, so wird in jedem Observer diese Methode aufgerufen. Nun kann man in dieser Methode auf die Notification reagieren. Wie man dies tun könnte, wird später auch noch im Beispiel in Kapitel 4 gezeigt. Hier wird der einfachheithalber lediglich der String "something changed" auf der Konsole ausgegeben.

Will man den vereinfachten Observer mal testen, so kann man die folgende Main Methode nutzen:

```
1   public class Main
2   {
3       public static void main(String[] args)
4       {
5           // instanciate subject and observer
6           Subject sub = new Subject();
7           ConcreteObserver obs = new ConcreteObserver();
8
9           // add observer to subject
```

```
10          sub.addObserver(obs);
11
12          // let subject do something
13          sub.doSomething();
14      }
15  }
```

Listing 7: Main Methode zum testen

Da es nur eine einfache Implementierung ist fehlen hier natürlich noch ein paar Methoden, wie beispielsweise `setChanged()` in der `MyObservable` Klasse, oder der `getState()` Methode in der `MyObservable` Klasse.

3.3 Update Möglichkeiten

So, nun haben wir in Abschnitt 3.1.1 bereits gelernt wie das Observer-Pattern funktioniert, doch wie tauschen wir nun Informationen über die indirekte Verbindung aus? Es treten ja nicht nur Situationen auf, indem wir eine andere Klasse über ein Ereignis informieren wollen, sondern müssen dieser Klasse eventuell auch ein paar Daten zum verarbeiten zukommen lassen. Um dies über diese indirekte Verbindung zu bewerkstelligen gibt es 3 Lösungsansätze. Diese 3 Lösungsansätze werden im folgenden kurz erläutert.

3.3.1 Push-Methode

Die erste hier vorgestellte Methode ist die Push-Methode, bei ihr werden bei jeder Notification vom Subject diverse Informationen an alle Observer gesendet, hierbei ist es dem Subject egal ob die Observer diese Informationen brauchen oder nicht, es kommt also oft vor das Observer Informationen bekommen die sie gar nicht benötigen.
Nachteilhaft bei dieser Methode ist, das die Wiederverwertbarkeit gesenkt wird, da man durch das senden diverser Informationen bei der Notification Annahmen über die Observer macht.
Vorteilhaft dagegen ist, das die Observer alle benötigten Informationen direkt mitgeliefert bekommen, und sie nicht erst zusammentragen müssen. Dies steigt ein wenig die Performance im Vergleich zur gleich besprochenen Pull Methode in 3.3.2.

Hilfreich für die Push-Methode wäre es ein neues Objekt anzulegen und dort alle relevanten Information zu speichern und bei der Notification als Subject spezifisches Argument mitzusenden. Im Beispiel in Kapitel 4 wird so ein `UpdateEvent` Objekt demonstriert.

3.3.2 Pull-Methode

Die Pull-Methode ist das komplette Gegenstück zur Push-Methode. Im Gegensatz zur Push-Methode werden bei der Pull-Methode nur minimal notwendige Informationen bei einer Notification mitgesendet.
Benötigt ein Observer nun zusätzliche Informationen vom Subject so muss der Observer diese benötigten Informationen explizit abfragen. Damit er dazu keine direkte Verbindung benötigt, ist es bei den meisten Observer Implementierungen normal, das in der Observable Klasse eine Referenz zum Subject gespeichert ist, und bei der Notification eine Referenz zur Observable Klasse vorhanden ist. In Java beispielsweise ist dies der Fall, dort wird bei jeder Notification

das Subject spezifische Argument, und eine Referenz zur Observable Klasse mitgesendet.
Vorteil dieser Methode ist, das die Kopplung auf ein Minimum reduziert wird, da das Subject keinerlei Annahmen über seine Observer treffen muss, und die Observer somit selber herausfinden müssen was sich geändert hat, um korrekt zu reagieren.
Nachteil dagegen ist, dass das zusammensuchen, und herausfinden was sich überhaupt geändert hat unter Umständen schon an die Performance gehen kann.

Es sollte hier noch erwähnt werden das man durchaus auch Zwischenformen der beiden Methoden verwenden kann.

3.3.3 Change-Manager

Nun haben wir in Abschnitt 3.3.1 und 3.3.2 zwei extreme Ansätze kennengelernt. Die letzte Lösungsmöglichkeit ist der sogenannte Change-Manager. Wächst ein Programm immer weiter an, so steigt auch die Anzahl der verwendeten Observer an. Mit steigender Anzahl von Observern wird das Observernetzwerk immer komplexer, und genau dann wird es Zeit für eine Komponente die dieses Netzwerk verwaltet, dem Change-Manager.
Der Change-Manager ist in erster Linie dazu da, den Aufwand der Observer auf ein Minimum zu reduzieren. Dies geschieht durch eine Update Strategy. Nehmen wir als Beispiel man an ein Observer muss erst genau dann eine Aktion durchführen, wenn mehrere Subjekte eine bestimmte Aktion durchgeführt haben. Ohne Update Strategy würde jedes Subjekt eine Broadcast Nachricht an seine Observer senden, und die Observer würden jedesmal schauen ob alle benötigten Aktionen durchgeführt wurden, und wenn ja, ihre Aktion durchführen. Die Update Strategy soll dafür sorgen das der Change-Manager den Observer erst genau dann informiert, wenn die Subjekte *alle* ihre notwendigen Aktionen durchgeführt haben, so das der Observer anstatt von jedem Subject eine Notification, nur eine Notification vom Change-Manager erhält.
Sogesehen sollte daher der Change-Manager bei größeren Projekten immer die erste Wahl sein, da er das Projekt später gut erweitert werden lässt, und den Aufwand der Observer minimiert, was Performance technisch Vorteile bringt.
Der größte Nachteil des Change-Managers dagegen ist allerdings das seine Implementierung sehr viel schwerer ist als die der anderen beiden vorgestellen Lösungsmöglichkeiten. Beispielsweise ist ist das Anwenden des *Mediator*- und *Singleton*-Pattern zu empfehlen, was wiederrum etwas mehr Programmierkenntnis erfordert.

So was tut der Change-Manager nun eigentlich? Er hat im Grundegenommen 3 Aufgaben:

i.) er hat beinhaltet die Update Strategy und sorgt dafür das diese reibungslos angewendet wird

ii.) Er verwaltet eine Map die jedes Subject auf ihre zugehörigen Observer abbildet, so wird den Subjects die Aufgabe abgenommen ihre Observer selber zu verwalten

iii.) er updatet alle notwendigen Observer auf Befehl des Subjects

Der Change-Manager ist aber mit Abstand die komplizierteste Update-Methode, weshalb wir sie hier nicht weiter vertiefen werden. Wer mehr über den Change-Manager und die bereits zuvor genannten *Mediator*- und *Singleton*-Pattern wissen will, dem sei unter anderem das Buch

"Design Patterns" von Erich Gamma nahegelegt. Hier werden diverse Patterns und deren Anwendung genauer erläutert.

3.4 Beispiel für das Observer-Pattern

Wie bereits in Abschnitt 3.2 erwähnt, bietet Java bereits die Klassen `java.util.Observable` und `java.util.Observer` an. In diesem Beispiel werden wir auch auf diese zurückgreifen. Hierzu betrachten wir erstmal die folgenden Beispiel Codes `AnObserver` und `ASubject`:

```
1  import java.util.Observable;
2
3  public class ASubject extends Observable
4  {
5      /**
6       * all Observers will be notified if this method is called
7       */
8      public void myAction()
9      {
10         super.setChanged();
11         super.notifyObservers();
12     }
13 }
```

Listing 8: Beispiel Subject mit `java.util.Observable`

```
1  import java.util.Observable;
2  import java.util.Observer;
3
4  public class AnObserver implements Observer
5  {
6      private static int id = 1;
7      // Observer id
8      private int myID;
9
10     /**
11      * Constructor
12      */
13     public AnObserver()
14     {
15         // set this observer id
16         myID = id;
17         // increment static id
18         id++;
19     }
20
21     /*
22      * (non-Javadoc)
23      * @see java.util.Observer#update
24      * (java.util.Observable, java.lang.Object)
25      */
26     @Override
27     public void update(Observable arg0, Object arg1)
28     {
```

```
29          // call myRespond if an notification is received
30          myRespond(arg0);
31      }
32
33      /**
34       * this method is called if a notification is received
35       *
36       * @param obs
37       *              observable who send this notification
38       */
39      private void myRespond(Observable obs)
40      {
41          System.out.println("Observer "+myID+" got a notification from "+obs);
42      }
43  }
```

Listing 9: Beispiel Observer mit `java.util.Observer`

Beim Subject in Listing 8 haben wir die Methode `myAction()` in der die Notification stattfindet, anders als bei der eigenen kleinen Implementierung bietet die von Java bereits bereitgestellte Klasse `java.util.Observable` natürlich viel mehr Methoden. Eine davon findet man bereits im Programmcode, in Zeile 10, die `setChanged()` Methode. Sie setzt einen internen Boolean auf true. Da bei jeder Notification eine Referenz zur Observable Instanz mitgereicht wird, können so alle Observer durch die ebenfalls vorhande Methode `hasChanged()` nachprüfen ob sich etwas geändert hat, und dementsprechend reagieren.
Die registrierten Observer werden also in den Zeilen 10 und 11 benachrichtigt.

Nun zur Observer Klasse in Listing 9. Im Konstruktor bekommt jeder Observer eine neue ID, dies dient hier später zur besseren Übersichtlichkeit. Anders als im Beispiel in Abschnitt 3.2 hat die `update()` Methode 2 Parameter, einmal das schon in Listing 6 gesehene spezifische Object arg1, und neu hinzugekommen ist eine Referenz vom Observable, arg0.
Tritt nun eine Notification ein, so wird die Methode `myRespond(Observable obs)` aufgerufen. In dieser wird uns dann angezeigt welcher Observer von welchem Observable eine Notification erhalten hat, natürlich können hier auch andere Aktionen durchgeführt werden, wie wir dann in Kapitel 4 sehen werden.

Zuletzt testen wir jetzt nun noch die Implementierung, hierzu nutzen wir die folgende Main:

```
1   public class Main
2   {
3       /**
4        * Main method
5        */
6       public static void main(String[] unused)
7       {
8           // instanciate Subjects and Observers
9           ASubject sub1 = new ASubject();
10          ASubject sub2 = new ASubject();
11          AnObserver obs1 = new AnObserver();
12          AnObserver obs2 = new AnObserver();
13
14          // add Observers to the Subjects
```

```
15          sub1.addObserver(obs1);
16          sub1.addObserver(obs2);
17          sub2.addObserver(obs1);
18          sub2.addObserver(obs2);
19
20          // call methods
21          sub1.myAction();
22          sub2.myAction();
23      }
24  }
```

Listing 10: Beispiel Main

Um zu demonstrieren wie die Observer funktionieren haben wir 2 Subjects, und 3 Observer angelegt. Die Observer 1 und 2 registrieren sich an Subject 1, und Observer 1 und 3 an Subject 2. Starten wir nun die Main, erhalten wir folgende Ausgabe:

```
Observer 2 got a notification from ASubject@30c221
Observer 1 got a notification from ASubject@30c221
Observer 3 got a notification from ASubject@119298d
Observer 1 got a notification from ASubject@119298d
```

Wie man sieht wurde nur Observer 1 und 2 von Subject 1, und nur Observer 1 und 3 von Subect 2 benachrichtigt.

4 Beispiel MVC- und Observer-Pattern

4.1 kurze Projekterläuterung

Das folgende Beispiel stellt einen kleinen Addierer names Awesome Adder dae, der eine Binär- und Hexadezimalzahl addiert, und das Ergebnis als Dezimalzahl angibt. Zusätzlich besteht die Möglichkeit die Binär- bzw. Hexadezimalzahl in einem eigenen kleinen Converter als Dezimal anzugeben und auf Befehl in Binär- bzw. Hexadezimal konvertieren zu lassen und an den Rechner zu übertragen. Aussehen tut der kleine Rechner dann so:

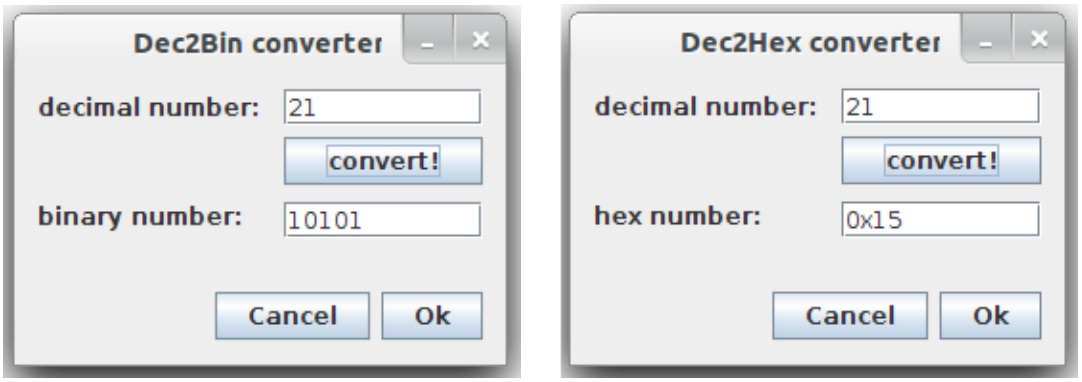

Abbildung 4.3: Dec2Bin und Dec2Hex converter

Abbildung 4.4: Hauptfenster das Awesome Adders

Um schonmal einen kurzen Überblick über den Awesome Adder zu erhalten betrachten wir ein vereinfachtes UML-Diagramm. Hierbei wurden die Observable Klasse und das Observer Interface der besseren Übersichtlichkeit wegen weggelassen. Die gestrichelten Pfeile sollen die indirekten Verbindungen aufzeigen:

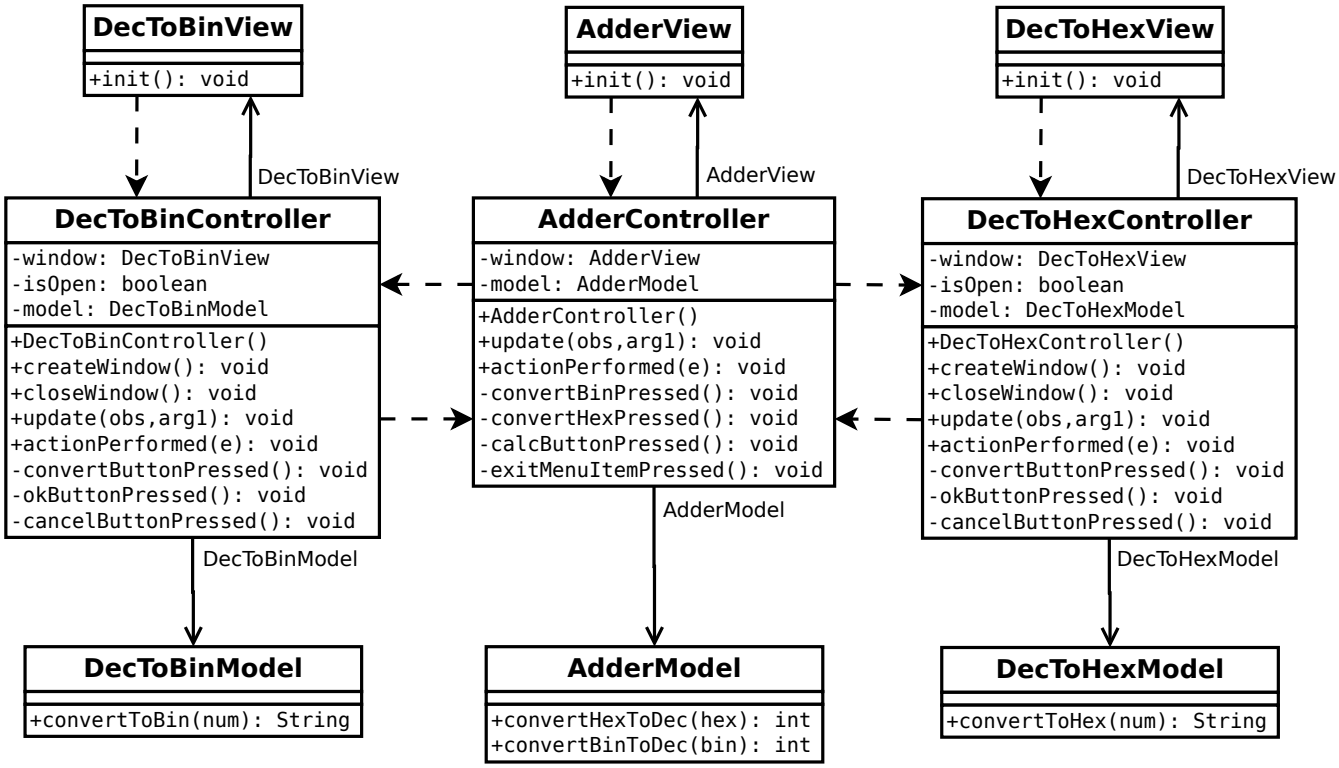

Abbildung 4.5: vereinfachtes UML des Awesome Adders

Bei der Implementierung wurde der besseren Übersichthalber auf die Fehlerbehandlung verzichtet, und es wird daher davon ausgegangen das die Nutzereingaben immer korrekt sind.Ebenso hätte man durchaus alle 3 Views von einem Model aus versorgen können, aber aus didaktischen Gründen hat jede View sein eigenes Model bekommen.

4.2 Designentscheidungen

Im Abschnitt 3.3.1 bei der Push Methode wurde erwähnt, das man sich überlegen kann ein `UpdateEvent` Objekt anzulegen, um dort alle relevanten Informationen die ein Subject bei jeder Notification mitsendet will dort abzulegen. In diesem Beispiel haben wir uns für die Push-Methode entschieden, und haben daher auch ein `UpdateEvent`, welches im Appendix in Abschnitt 5.3 in Listing 18 zu finden ist.
Im UpdateEvent speichern wir den Ursprung, also wer die Notification gesendet hat, und den Receiver, also an wen die Notification gerichtet ist. Und als letzten Parameter kann man optional noch ein Argument als String übergeben. Da dies nicht immer nötig ist, gibt es 2 Konstruktoren, einmal mit spezifischem Parameter, einmal ohne.
Für Sender und Receiver gibt es um Fehler zu vermeiden eine extra Enumeration, welche in Abschnitt 5.2 Listing 14 zu finden ist.

4.3 Views

Die Oberflächen wurden alle mit dem GUI-Builder von Netbeans erstellt, mehr zu Netbeans kann man der Netbeans Seite entnehmen.
Der Sourcecode der Views befindet sich im Appendix in Abschnitt 5.1.
Im Prinzip ist jede View ähnlich aufgebaut wie schon in Abschnitt 2.3 erklärt. Jede GUI Komponente, also Buttons etc. haben jeweils einen Getter, eine `setActionListener(ActionListener l)` Methode um jeder Komponente einen ActionListener zuzuweisen. Daher halten wir uns nicht lange bei den Views auf und beenden diesen Abschnitt auch schon.

4.4 Controller

Der Sourcecode der Controller befindet sich ebenfalls im Appendix im Abschnitt 5.2.

Wie in Abschnitt 2.3 gezeigt, haben wir die indirekte Verbindung hergestellt, indem die Controller direkt als ActionListener an die einzelnen GUI Komponenten übergeben wurde, dies geschieht hier beispielsweise in Zeile 45 des DecToHex Controllers in Listing 17.
Anders als in den anderen Beispielen zuvor dient der Controller nicht nur als Observer, sondern diesesmal als Observer und als Subject, da er sowohl andere Observer benachrichtigen muss, aber auch Benachrichtigungen von anderen Subjects entgegennehmen muss.
Zusätzlich gibt es noch jeweils eine Methode um das jeweilige Fenster zu generieren, und auch um es zu schliessen. Eine Ausnahme bildet hierbei der Adder Controller, da dieser das Main Window kontrolliert, welches immer offen ist, muss es dafür keine Methode geben um es zu schliessen.
Ebenfalls besitzt jeder Controller noch die `actionPerformed(Actionevent e)` Methode. Wie er nun rausfindet welche GUI Komponente betätigt wurde, wurde bereits in Abschnitt 2.3 erläutert.

4.5 Models

Der Sourcecode der Models befindet sich auch im Appendix im Abschnitt 5.3.

Wie bereits erwähnt hätte man hier auch durchaus nur ein Model gebraucht. Im Prinzip geschieht in den Models die ganze Rechenarbeit, beispielsweise das umrechnen von Dezimal nach Binär etc.

Das besondere an den Models ist wohl, das sie keinerlei Verbindungen zu den anderen Komponenten haben. Wir könnten die Models einfach so aus dem Projekt rausnehmen und in anderen Projekten einbinden.

5 Appendix

5.1 Sourcecode Views

```
1  package view;
2
3  import java.awt.event.ActionListener;
4  import javax.swing.*;
5
6  public class AdderView extends JFrame
7  {
8      private static final long serialVersionUID = 1L;
9
10     // Variables declaration - do not modify
11     private JLabel binNumL;
12     private JTextField binNumTF;
13     private JButton calcB;
14     private JButton convertBinB;
15     private JButton convertHexB;
16     private JLabel equL;
17     private JMenuItem exitMI;
18     private JMenu fileM;
19     private JLabel hexNumL;
20     private JTextField hexNumTF;
21     private JMenuBar jMenuBar1;
22     private JPanel jPanel1;
23     private JLabel plusL;
24     private JLabel resultL;
25     private JTextField resultTF;
26     // End of variables declaration
27
28     /**
29      * initializes gui elements
30      */
31     public void init()
32     {
33         setTitle("Awesome Adder");
34
35         jPanel1 = new JPanel();
36         binNumL = new JLabel();
37         binNumTF = new JTextField();
38         convertBinB = new JButton();
39         plusL = new JLabel();
40         hexNumTF = new JTextField();
41         hexNumL = new JLabel();
42         convertHexB = new JButton();
43         equL = new JLabel();
44         resultTF = new JTextField();
45         resultL = new JLabel();
46         calcB = new JButton();
47         jMenuBar1 = new JMenuBar();
48         fileM = new JMenu();
49         exitMI = new JMenuItem();
```

```
50
51          setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
52
53          binNumL.setText("binary number");
54
55          convertBinB.setText("convert bin");
56
57          plusL.setText("+");
58
59          hexNumL.setText("hex number");
60
61          convertHexB.setText("convert hex");
62
63          equL.setText("=");
64
65          resultL.setText("result in decimal");
66
67          calcB.setText("calculate!");
68
69          GroupLayout jPanel1Layout = new GroupLayout(jPanel1);
70          jPanel1.setLayout(jPanel1Layout);
71          jPanel1Layout.setHorizontalGroup(
72              jPanel1Layout.createParallelGroup(GroupLayout.Alignment.LEADING)
73              .addGroup(jPanel1Layout.createSequentialGroup()
74                  .addContainerGap()
75                  .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.
                        Alignment.LEADING)
76                      .addComponent(binNumL, GroupLayout.DEFAULT_SIZE, 106, Short.
                            MAX_VALUE)
77                      .addComponent(binNumTF, GroupLayout.DEFAULT_SIZE, 106, Short
                            .MAX_VALUE)
78                      .addComponent(convertBinB, GroupLayout.DEFAULT_SIZE, 106,
                            Short.MAX_VALUE))
79                  .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
80                  .addComponent(plusL)
81                  .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
82                  .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.
                        Alignment.LEADING)
83                      .addComponent(convertHexB, GroupLayout.DEFAULT_SIZE, 103,
                            Short.MAX_VALUE)
84                      .addComponent(hexNumL, GroupLayout.DEFAULT_SIZE, 103, Short.
                            MAX_VALUE)
85                      .addComponent(hexNumTF, GroupLayout.Alignment.TRAILING,
                            GroupLayout.DEFAULT_SIZE, 103, Short.MAX_VALUE))
86                  .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
87                  .addComponent(equL)
88                  .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
89                  .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.
                        Alignment.LEADING, false)
90                      .addComponent(resultL, GroupLayout.DEFAULT_SIZE, GroupLayout
                            .DEFAULT_SIZE, Short.MAX_VALUE)
91                      .addComponent(resultTF)
```

```
92                      .addComponent(calcB, GroupLayout.DEFAULT_SIZE, GroupLayout.
                            DEFAULT_SIZE, Short.MAX_VALUE))
93                  .addContainerGap(24, Short.MAX_VALUE))
94          );
95          jPanel1Layout.setVerticalGroup(
96              jPanel1Layout.createParallelGroup(GroupLayout.Alignment.LEADING)
97              .addGroup(jPanel1Layout.createSequentialGroup()
98                  .addContainerGap()
99                  .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.
                        Alignment.BASELINE)
100                     .addComponent(binNumL)
101                     .addComponent(hexNumL)
102                     .addComponent(resultL))
103                 .addPreferredGap(LayoutStyle.ComponentPlacement.UNRELATED)
104                 .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.
                        Alignment.BASELINE)
105                     .addComponent(binNumTF, GroupLayout.PREFERRED_SIZE,
                            GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
106                     .addComponent(plusL)
107                     .addComponent(hexNumTF, GroupLayout.PREFERRED_SIZE,
                            GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
108                     .addComponent(equL)
109                     .addComponent(resultTF, GroupLayout.PREFERRED_SIZE,
                            GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE))
110                 .addPreferredGap(LayoutStyle.ComponentPlacement.UNRELATED)
111                 .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.
                        Alignment.BASELINE)
112                     .addComponent(convertBinB)
113                     .addComponent(convertHexB)
114                     .addComponent(calcB))
115                 .addContainerGap(GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
116         );
117
118         fileM.setText("File");
119
120         exitMI.setText("Exit");
121         fileM.add(exitMI);
122
123         jMenuBar1.add(fileM);
124
125         setJMenuBar(jMenuBar1);
126
127         GroupLayout layout = new GroupLayout(getContentPane());
128         getContentPane().setLayout(layout);
129         layout.setHorizontalGroup(
130             layout.createParallelGroup(GroupLayout.Alignment.LEADING)
131             .addComponent(jPanel1, GroupLayout.DEFAULT_SIZE, GroupLayout.
                    DEFAULT_SIZE, Short.MAX_VALUE)
132         );
133         layout.setVerticalGroup(
134             layout.createParallelGroup(GroupLayout.Alignment.LEADING)
135             .addComponent(jPanel1, GroupLayout.PREFERRED_SIZE, GroupLayout.
                    DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
```

```
136             );
137
138             pack();
139         }
140
141
142         /* *****************
143          * Getter & Setter *
144          * *****************
145          */
146         // ActionListener
147         /**
148          * Setter for all ActionListener
149          *
150          * @param l
151          *            ActionListener
152          */
153         public void setActionListener(ActionListener l)
154         {
155             convertBinB.addActionListener(l);
156             convertHexB.addActionListener(l);
157             calcB.addActionListener(l);
158             exitMI.addActionListener(l);
159         }
160
161         // Getter
162         /**
163          * Getter for convertBin Button
164          */
165         public JButton getConvertBinButton()
166         {
167             return convertBinB;
168         }
169
170         /**
171          * Getter for convertHex Button
172          */
173         public JButton getConvertHexButton()
174         {
175             return convertHexB;
176         }
177
178         /**
179          * Getter for calculate Button
180          */
181         public JButton getCalcButton()
182         {
183             return calcB;
184         }
185
186         /**
187          * Getter for Exit MenuItem
188          */
```

```
189     public JMenuItem getExitMenuItem()
190     {
191         return exitMI;
192     }
193 
194     /**
195      * Getter for bin num
196      */
197     public String getBinNum()
198     {
199         return binNumTF.getText();
200     }
201 
202     /**
203      * Getter for hex num
204      */
205     public String getHexNum()
206     {
207         return hexNumTF.getText();
208     }
209 
210 
211     // Setter
212     /**
213      * setter for binNumTF
214      *
215      * @param txt
216      *              new number in bin
217      */
218     public void setBinNumTField(String txt)
219     {
220         binNumTF.setText(txt);
221     }
222 
223     /**
224      * setter for hexNumTF
225      *
226      * @param txt
227      *              new number in hex
228      */
229     public void setHexNumTField(String txt)
230     {
231         hexNumTF.setText(txt);
232     }
233 
234     /**
235      * setter for resultTF
236      *
237      * @param txt new Number in Dec
238      */
239     public void setResultTField(String txt)
240     {
241         resultTF.setText(txt);
```

```
242     }
243 }
```

Listing 11: Sourcecode Awesome Adder View

```
1  package view;
2
3  import java.awt.event.ActionListener;
4  import javax.swing.*;
5
6  public class DecToBinView extends JFrame
7  {
8      private static final long serialVersionUID = 1L;
9
10     // Variables declaration - do not modify
11     private JLabel binNumL;
12     private JTextField binNumTF;
13     private JButton cancelB;
14     private JButton convertB;
15     private JLabel decNumL;
16     private JTextField decNumTF;
17     private JPanel jPanel1;
18     private JButton okB;
19     // End of variables declaration
20
21     /**
22      * initializes gui elements
23      */
24     public void init()
25     {
26         setTitle("Dec2Bin converter");
27
28         jPanel1 = new JPanel();
29         decNumL = new JLabel();
30         binNumL = new JLabel();
31         decNumTF = new JTextField();
32         binNumTF = new JTextField();
33         okB = new JButton();
34         cancelB = new JButton();
35         convertB = new JButton();
36
37         setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
38
39         decNumL.setText("decimal number:");
40
41         binNumL.setText("binary number:");
42
43         okB.setText("Ok");
44
45         cancelB.setText("Cancel");
46
47         convertB.setText("convert!");
48
49         GroupLayout jPanel1Layout = new GroupLayout(jPanel1);
```

```
50          jPanel1.setLayout(jPanel1Layout);
51          jPanel1Layout.setHorizontalGroup(
52              jPanel1Layout.createParallelGroup(GroupLayout.Alignment.LEADING)
53              .addGroup(jPanel1Layout.createSequentialGroup()
54                  .addContainerGap()
55                  .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.
                        Alignment.LEADING)
56                      .addGroup(GroupLayout.Alignment.TRAILING, jPanel1Layout.
                            createSequentialGroup()
57                          .addComponent(cancelB)
58                          .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
59                          .addComponent(okB))
60                      .addGroup(GroupLayout.Alignment.TRAILING, jPanel1Layout.
                            createSequentialGroup()
61                          .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.
                                Alignment.LEADING)
62                              .addGroup(jPanel1Layout.createSequentialGroup()
63                                  .addComponent(binNumL, GroupLayout.DEFAULT_SIZE,
                                        117, Short.MAX_VALUE)
64                                  .addPreferredGap(LayoutStyle.ComponentPlacement.
                                        RELATED))
65                              .addGroup(jPanel1Layout.createSequentialGroup()
66                                  .addComponent(decNumL)
67                                  .addGap(6, 6, 6)))
68                          .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.
                                Alignment.LEADING)
69                              .addComponent(decNumTF, GroupLayout.DEFAULT_SIZE,
                                    104, Short.MAX_VALUE)
70                              .addComponent(binNumTF, GroupLayout.Alignment.
                                    TRAILING, GroupLayout.DEFAULT_SIZE, 104, Short.
                                    MAX_VALUE)
71                              .addComponent(convertB, GroupLayout.DEFAULT_SIZE,
                                    104, Short.MAX_VALUE))))
72                  .addContainerGap())
73          );
74          jPanel1Layout.setVerticalGroup(
75              jPanel1Layout.createParallelGroup(GroupLayout.Alignment.LEADING)
76              .addGroup(jPanel1Layout.createSequentialGroup()
77                  .addContainerGap()
78                  .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.
                        Alignment.BASELINE)
79                      .addComponent(decNumL)
80                      .addComponent(decNumTF, GroupLayout.PREFERRED_SIZE,
                            GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE))
81                  .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
82                  .addComponent(convertB)
83                  .addGap(9, 9, 9)
84                  .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.
                        Alignment.LEADING)
85                      .addComponent(binNumTF, GroupLayout.PREFERRED_SIZE,
                            GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
86                      .addComponent(binNumL))
87                  .addGap(28, 28, 28)
```

```
88                      .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.
                            Alignment.BASELINE)
89                          .addComponent(okB)
90                          .addComponent(cancelB))
91                      .addContainerGap(GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
92              );
93
94              GroupLayout layout = new GroupLayout(getContentPane());
95              getContentPane().setLayout(layout);
96              layout.setHorizontalGroup(
97                  layout.createParallelGroup(GroupLayout.Alignment.LEADING)
98                  .addComponent(jPanel1, GroupLayout.PREFERRED_SIZE, GroupLayout.
                        DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
99              );
100             layout.setVerticalGroup(
101                 layout.createParallelGroup(GroupLayout.Alignment.LEADING)
102                 .addComponent(jPanel1, GroupLayout.PREFERRED_SIZE, GroupLayout.
                        DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
103             );
104
105             pack();
106         }
107
108
109         /* *****************
110          * Getter & Setter *
111          * *****************
112          */
113
114         // ActionListener
115         /**
116          * Setter for all ActionListener
117          *
118          * @param l ActionListener
119          */
120         public void setActionListener(ActionListener l)
121         {
122             cancelB.addActionListener(l);
123             okB.addActionListener(l);
124             convertB.addActionListener(l);
125         }
126
127         // Getter
128         /**
129          * Getter for cancel Button
130          */
131         public JButton getCancelButton()
132         {
133             return cancelB;
134         }
135
136         /**
137          * Getter for ok Button
```

```
138          */
139         public JButton getOkButton()
140         {
141             return okB;
142         }
143 
144         /**
145          * Getter for convert Button
146          */
147         public JButton getConvertButton()
148         {
149             return convertB;
150         }
151 
152         /**
153          * Getter for decimal number
154          */
155         public String getDecNum()
156         {
157             return decNumTF.getText();
158         }
159 
160         /**
161          * Getter for bin number
162          */
163         public String getBinNum()
164         {
165             return binNumTF.getText();
166         }
167 
168 
169         // Setter
170         /**
171          * setter for decNumTF
172          *
173          * @param txt
174          *                  new number in dec
175          */
176         public void setDecNumTField(String txt)
177         {
178             decNumTF.setText(txt);
179         }
180 
181         /**
182          * setter for binNumTF
183          *
184          * @param txt
185          *                  new number in bin
186          */
187         public void setBinNumTField(String txt)
188         {
189             binNumTF.setText(txt);
190         }
```

```
191 }
```

Listing 12: Sourcecode des Dec2Bin converter View

```
1  package view;
2
3  import java.awt.event.ActionListener;
4  import javax.swing.*;
5
6  public class DecToHexView extends JFrame
7  {
8      private static final long serialVersionUID = 1L;
9
10     // Variables declaration - do not modify
11     private JLabel binNumL;
12     private JButton cancelB;
13     private JButton convertB;
14     private JLabel decNumL;
15     private JTextField decNumTF;
16     private JTextField hexNumTF;
17     private JPanel jPanel1;
18     private JButton okB;
19     // End of variables declaration
20
21     /**
22      * initializes gui elements
23      */
24     public void init()
25     {
26         setTitle("Dec2Hex converter");
27
28         jPanel1 = new JPanel();
29         decNumL = new JLabel();
30         binNumL = new JLabel();
31         decNumTF = new JTextField();
32         hexNumTF = new JTextField();
33         okB = new JButton();
34         cancelB = new JButton();
35         convertB = new JButton();
36
37         setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
38
39         decNumL.setText("decimal number:");
40
41         binNumL.setText("hex number:");
42
43         okB.setText("Ok");
44
45         cancelB.setText("Cancel");
46
47         convertB.setText("convert!");
48
49         GroupLayout jPanel1Layout = new GroupLayout(jPanel1);
50         jPanel1.setLayout(jPanel1Layout);
```

```
51          jPanel1Layout.setHorizontalGroup(
52              jPanel1Layout.createParallelGroup(GroupLayout.Alignment.LEADING)
53              .addGroup(jPanel1Layout.createSequentialGroup()
54                  .addContainerGap()
55                  .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.
                        Alignment.LEADING)
56                      .addGroup(GroupLayout.Alignment.TRAILING, jPanel1Layout.
                            createSequentialGroup()
57                          .addComponent(cancelB)
58                          .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
59                          .addComponent(okB))
60                      .addGroup(GroupLayout.Alignment.TRAILING, jPanel1Layout.
                            createSequentialGroup()
61                          .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.
                                Alignment.LEADING)
62                              .addGroup(jPanel1Layout.createSequentialGroup()
63                                  .addComponent(binNumL, GroupLayout.DEFAULT_SIZE,
                                        117, Short.MAX_VALUE)
64                                  .addPreferredGap(LayoutStyle.ComponentPlacement.
                                        RELATED))
65                              .addGroup(jPanel1Layout.createSequentialGroup()
66                                  .addComponent(decNumL)
67                                  .addGap(6, 6, 6)))
68                          .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.
                                Alignment.LEADING)
69                              .addComponent(decNumTF, GroupLayout.DEFAULT_SIZE,
                                    104, Short.MAX_VALUE)
70                              .addComponent(hexNumTF, GroupLayout.Alignment.
                                    TRAILING, GroupLayout.DEFAULT_SIZE, 104, Short.
                                    MAX_VALUE)
71                              .addComponent(convertB, GroupLayout.DEFAULT_SIZE,
                                    104, Short.MAX_VALUE))))
72                  .addContainerGap())
73          );
74          jPanel1Layout.setVerticalGroup(
75              jPanel1Layout.createParallelGroup(GroupLayout.Alignment.LEADING)
76              .addGroup(jPanel1Layout.createSequentialGroup()
77                  .addContainerGap()
78                  .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.
                        Alignment.BASELINE)
79                      .addComponent(decNumL)
80                      .addComponent(decNumTF, GroupLayout.PREFERRED_SIZE,
                            GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE))
81                  .addPreferredGap(LayoutStyle.ComponentPlacement.RELATED)
82                  .addComponent(convertB)
83                  .addGap(9, 9, 9)
84                  .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.
                        Alignment.LEADING)
85                      .addComponent(hexNumTF, GroupLayout.PREFERRED_SIZE,
                            GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
86                      .addComponent(binNumL))
87                  .addGap(28, 28, 28)
88                  .addGroup(jPanel1Layout.createParallelGroup(GroupLayout.
```

```
                        Alignment.BASELINE)
89                       .addComponent(okB)
90                       .addComponent(cancelB))
91                  .addContainerGap(GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
92          );
93
94          GroupLayout layout = new GroupLayout(getContentPane());
95          getContentPane().setLayout(layout);
96          layout.setHorizontalGroup(
97              layout.createParallelGroup(GroupLayout.Alignment.LEADING)
98              .addComponent(jPanel1, GroupLayout.PREFERRED_SIZE, GroupLayout.
                    DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
99          );
100         layout.setVerticalGroup(
101             layout.createParallelGroup(GroupLayout.Alignment.LEADING)
102             .addComponent(jPanel1, GroupLayout.PREFERRED_SIZE, GroupLayout.
                    DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
103         );
104
105         pack();
106     }
107
108
109     /* *****************
110      * Getter & Setter *
111      * *****************
112      */
113     // ActionListener
114     /**
115      * Setter for all ActionListener
116      *
117      * @param l ActionListener
118      */
119     public void setActionListener(ActionListener l)
120     {
121         cancelB.addActionListener(l);
122         okB.addActionListener(l);
123         convertB.addActionListener(l);
124     }
125
126     // Getter
127     /**
128      * Getter for cancel Button
129      */
130     public JButton getCancelButton()
131     {
132         return cancelB;
133     }
134
135     /**
136      * Getter for ok Button
137      */
138     public JButton getOkButton()
```

```
139     {
140         return okB;
141     }
142 
143     /**
144      * Getter for convert Button
145      */
146     public JButton getConvertButton()
147     {
148         return convertB;
149     }
150 
151     /**
152      * Getter for decimal number
153      */
154     public String getDecNum()
155     {
156         return decNumTF.getText();
157     }
158 
159     /**
160      * Getter for hex number
161      */
162     public String getBinNum()
163     {
164         return hexNumTF.getText();
165     }
166 
167     // Setter
168     /**
169      * setter for decNumTF
170      *
171      * @param txt
172      *              new number in dec
173      */
174     public void setDecNumTField(String txt)
175     {
176         decNumTF.setText(txt);
177     }
178 
179     /**
180      * setter for hexNumTF
181      *
182      * @param txt
183      *              new number in hex
184      */
185     public void setHexNumTField(String txt)
186     {
187         hexNumTF.setText(txt);
188     }
189 }
```

Listing 13: Sourcecode des Dec2Hex converter View

5.2 Sourcecode Controller

```
1  package controller;
2
3  public enum ControllerEnum
4  {
5      DECTOBIN, DECTOHEX, MAIN
6  }
```

Listing 14: Sourcecode der Controller Enumeration

```
1  package controller;
2
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.util.Observable;
6  import java.util.Observer;
7  import model.AdderModel;
8  import model.UpdateEvent;
9  import view.AdderView;
10
11 public class AdderController extends Observable implements Observer,
       ActionListener
12 {
13     private AdderView window;
14     private AdderModel model;
15
16     /**
17      * constructor
18      */
19     public AdderController()
20     {
21         super();
22
23         // create new MainView window
24         window = new AdderView();
25         window.init();
26         window.setVisible(true);
27
28         // set ActionListener
29         window.setActionListener(this);
30
31         // init model
32         model = new AdderModel();
33     }
34
35     /*
36      * (non-Javadoc)
37      * @see java.util.Observer#update(java.util.Observable, java.lang.Object)
38      */
39     @Override
40     public void update(Observable obs, Object arg1)
41     {
42         // cast arg to UpdateEvent
```

```
43          UpdateEvent ue = (UpdateEvent) arg1;
44
45          // get arguments
46          String arg = ue.getArg();
47
48          // check if UpdateEvent is for us, otherwise dont care
49          if(ue.getTo() == ControllerEnum.MAIN)
50          {
51              // if UpdateEvent is from DecToBin Controller set bin number
52              if(ue.getFrom() == ControllerEnum.DECTOBIN)
53                  window.setBinNumTField(arg);
54              // if UpdateEvent is from DecToHex Controller set hex number
55              else if(ue.getFrom() == ControllerEnum.DECTOHEX)
56                  window.setHexNumTField(arg);
57          }
58      }
59
60
61      /* ***************
62       * ActionListener *
63       * ***************
64       */
65      /**
66       * called if a Button got pressed
67       */
68      @Override
69      public void actionPerformed(ActionEvent e)
70      {
71          // convert to Bin Button
72          if(e.getSource() == window.getConvertBinButton())
73              convertBinPressed();
74          // convert to Hex Button
75          else if(e.getSource() == window.getConvertHexButton())
76              convertHexPressed();
77          // calc Button
78          else if(e.getSource() == window.getCalcButton())
79              calcButtonPressed();
80          // exit MenuItem
81          else if(e.getSource() == window.getExitMenuItem())
82              exitMenuItemPressed();
83      }
84
85      /**
86       * called if convertBin Button got pressed
87       */
88      private void convertBinPressed()
89      {
90          // notifies observers that something changed, in this case that DecToBin
                should open
91          super.setChanged();
92          // send Update Event to all Observers
93          super.notifyObservers(new UpdateEvent(ControllerEnum.MAIN,
                ControllerEnum.DECTOBIN));
```

```
94      }
95
96      /**
97       * called if convertHex Button got pressed
98       */
99      private void convertHexPressed()
100     {
101         // notifies observers that something changed, in this case that DecToHex
                should open
102         super.setChanged();
103         // send Update Event to all Observers
104         super.notifyObservers(new UpdateEvent(ControllerEnum.MAIN,
                ControllerEnum.DECTOHEX));
105     }
106
107     /**
108      * called if calc Button got pressed
109      */
110     private void calcButtonPressed()
111     {
112         // get binary number
113         String bin = window.getBinNum();
114         // get hex number without 0x
115         String hex = window.getHexNum().substring(2);
116
117         // convert bin and hex numbers to dec
118         int binInDec = model.convertBinToDec(bin);
119         int hexInDec = model.convertHexToDec(hex);
120
121         // calculate result
122         int result = binInDec + hexInDec;
123
124         // set result
125         window.setResultTField(new Integer(result).toString());
126     }
127
128     /**
129      * called if exit MenuItem got pressed
130      */
131     private void exitMenuItemPressed()
132     {
133         System.exit(0);
134     }
135 }
```

Listing 15: Sourcecode des Adder Controllers

```
1  package controller;
2
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.util.Observable;
6  import java.util.Observer;
7  import model.DecToBinModel;
```

```
8  import model.UpdateEvent;
9  import view.DecToBinView;
10
11 public class DecToBinController extends Observable implements Observer,
       ActionListener
12 {
13     private DecToBinView window;
14     private boolean isOpen;
15     private DecToBinModel model;
16
17     /**
18      * Constructor
19      */
20     public DecToBinController()
21     {
22         super();
23         isOpen = false;
24
25         // init model
26         model = new DecToBinModel();
27     }
28
29
30     /**
31      * creates a new DecToBinView window
32      */
33     public void createWindow()
34     {
35         // if there is a DecToBinView window open do noting
36         if(isOpen)
37             return;
38
39         // create window
40         window = new DecToBinView();
41         window.init();
42         window.setVisible(true);
43
44         // set ActionListener
45         window.setActionListener(this);
46
47         // set isOpen to true
48         isOpen = true;
49     }
50
51     /**
52      * closes the current DecToBinView window
53      */
54     public void closeWindow()
55     {
56         // if no DecToBin window is open do nothing
57         if(!isOpen)
58             return;
59
```

```
60          // closes the current DecToBin window
61          window.setVisible(false);
62          window.dispose();
63
64          // set isOpen to false
65          isOpen = false;
66      }
67
68      /*
69       * (non-Javadoc)
70       * @see java.util.Observer#update(java.util.Observable, java.lang.Object)
71       */
72      @Override
73      public void update(Observable obs, Object arg1)
74      {
75          // cast arg to UpdateEvent
76          UpdateEvent ue = (UpdateEvent) arg1;
77
78          // check if UpdateEvent is for us, otherwise dont care
79          if(ue.getTo() == ControllerEnum.DECTOBIN)
80          {
81              // if it comes from Main open a window
82              if(ue.getFrom() == ControllerEnum.MAIN)
83                  createWindow();
84          }
85      }
86
87
88      /* ****************
89       * ActionListener *
90       * ****************
91       */
92
93      /**
94       * called if a Button got pressed
95       */
96      @Override
97      public void actionPerformed(ActionEvent e)
98      {
99          // cancel Button
100         if(e.getSource() == window.getCancelButton())
101             cancelButtonPressed();
102         // ok Button
103         else if(e.getSource() == window.getOkButton())
104             okButtonPressed();
105         // convert Button
106         else if(e.getSource() == window.getConvertButton())
107             convertButtonPressed();
108     }
109
110     /**
111      * called if convert Button got pressed
112      */
```

```
113     private void convertButtonPressed()
114     {
115         // get number and convert it
116         int num = Integer.parseInt(window.getDecNum());
117         String bin = model.convertToBin(num);
118 
119         // set bin number
120         window.setBinNumTField(bin);
121     }
122 
123     /**
124      * called if Ok Button got pressed
125      */
126     private void okButtonPressed()
127     {
128         // notifies observers that something changed
129         super.setChanged();
130         // send Update Event to all Observers
131         super.notifyObservers(new UpdateEvent(ControllerEnum.DECTOBIN,
                ControllerEnum.MAIN, window.getBinNum()));
132         // close window
133         closeWindow();
134     }
135 
136     /**
137      * called if Cancel Button got pressed
138      */
139     private void cancelButtonPressed()
140     {
141         // if canceled close window and do nothing
142         closeWindow();
143     }
144 }
```

Listing 16: Sourcecode des DecToBin Controllers

```
1  package controller;
2 
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.util.Observable;
6  import java.util.Observer;
7  import model.DecToHexModel;
8  import model.UpdateEvent;
9  import view.DecToHexView;
10 
11 public class DecToHexController extends Observable implements Observer,
       ActionListener
12 {
13     private DecToHexView window;
14     private boolean isOpen;
15     private DecToHexModel model;
16 
17     /**
```

```
18      * constructor
19      */
20     public DecToHexController()
21     {
22         super();
23         isOpen = false;
24 
25         // init model
26         model = new DecToHexModel();
27     }
28 
29 
30     /**
31      * creates a new DecToHexView window
32      */
33     public void createWindow()
34     {
35         // if there is a DecToHexView window open do noting
36         if(isOpen)
37             return;
38 
39         // create window
40         window = new DecToHexView();
41         window.init();
42         window.setVisible(true);
43 
44         // set ActionListener
45         window.setActionListener(this);
46 
47         // set isOpen to true
48         isOpen = true;
49     }
50 
51     /**
52      * closes the current DecToHexView window
53      */
54     public void closeWindow()
55     {
56         // if no HexToBin window is open do nothing
57         if(!isOpen)
58             return;
59 
60         // closes the current DecToBin window
61         window.setVisible(false);
62         window.dispose();
63 
64         // set isOpen to false
65         isOpen = false;
66     }
67 
68     /*
69      * (non-Javadoc)
70      * @see java.util.Observer#update(java.util.Observable, java.lang.Object)
```

```
71          */
72         @Override
73         public void update(Observable obs, Object arg1)
74         {
75             // cast arg to UpdateEvent
76             UpdateEvent ue = (UpdateEvent) arg1;
77 
78             // check if UpdateEvent is for us, otherwise dont care
79             if(ue.getTo() == ControllerEnum.DECTOHEX)
80             {
81                 // if it comes from Main open a window
82                 if(ue.getFrom() == ControllerEnum.MAIN)
83                     createWindow();
84             }
85         }
86 
87 
88         /* ***************
89          * ActionListener *
90          * ***************
91          */
92         /**
93          * called if a Button got pressed
94          */
95         @Override
96         public void actionPerformed(ActionEvent e)
97         {
98             // cancel Button
99             if(e.getSource() == window.getCancelButton())
100                cancelButtonPressed();
101            // ok Button
102            else if(e.getSource() == window.getOkButton())
103                okButtonPressed();
104            // convert Button
105            else if(e.getSource() == window.getConvertButton())
106                convertButtonPressed();
107        }
108 
109        /**
110         * called if convert Button got pressed
111         */
112        private void convertButtonPressed()
113        {
114            // get number and convert it
115            int num = Integer.parseInt(window.getDecNum());
116            String hex = model.convertToHex(num);
117 
118            // set bin number
119            window.setHexNumTField("0x"+hex);
120        }
121 
122        /**
123         * called if Ok Button got pressed
```

```
124         */
125         private void okButtonPressed()
126         {
127             // notifies observers that something changed
128             super.setChanged();
129             // send Update Event to all Observers
130             super.notifyObservers(new UpdateEvent(ControllerEnum.DECTOHEX,
                    ControllerEnum.MAIN, window.getBinNum()));
131
132             // close window
133             closeWindow();
134         }
135
136         /**
137          * called if Cancel Button got pressed
138          */
139         private void cancelButtonPressed()
140         {
141             closeWindow();
142         }
143 }
```

Listing 17: Sourcecode des DecToHex Controllers

5.3 Sourcecode Models

```
1  package model;
2
3  import controller.ControllerEnum;
4
5  public class UpdateEvent
6  {
7      ControllerEnum from;
8      ControllerEnum to;
9      String arg;
10
11     /**
12      * constructor
13      *
14      * @param from
15      *              sender
16      * @param to
17      *              receiver
18      */
19     public UpdateEvent(ControllerEnum from, ControllerEnum to)
20     {
21         this.from = from;
22         this.to = to;
23         this.arg = "";
24     }
25
26     /**
27      * constructor
```

```
28          *
29          * @param from
30          *                  sender
31          * @param to
32          *                  receiver
33          * @param arg
34          *                  specific argument
35          */
36         public UpdateEvent(ControllerEnum from, ControllerEnum to, String arg)
37         {
38             this.from = from;
39             this.to = to;
40             this.arg = arg;
41         }
42
43
44         /* *****************
45          * Getter & Setter *
46          * *****************
47          */
48         /**
49          * Getter for from
50          */
51         public ControllerEnum getFrom()
52         {
53             return from;
54         }
55
56         /**
57          * Getter for to
58          */
59         public ControllerEnum getTo()
60         {
61             return to;
62         }
63
64         /**
65          * Getter for arg
66          */
67         public String getArg()
68         {
69             return arg;
70         }
71 }
```

Listing 18: Sourcecode des UpdateEvents

```
1  package model;
2
3  public class AdderModel
4  {
5      /**
6       * converts a given hex number to dec
7       *
```

```
8         * @param hex
9         *              given hex number
10        *
11        * @return given number in dec
12        */
13       public int convertHexToDec(String hex)
14       {
15           return Integer.parseInt(hex,16);
16       }
17
18       /**
19        * converts a given bin number to dec
20        *
21        * @param hex
22        *              given bin number
23        *
24        * @return given number in dec
25        */
26       public int convertBinToDec(String bin)
27       {
28           return Integer.parseInt(bin, 2);
29       }
30   }
```

Listing 19: Sourcecode des Adder Models

```
1   package model;
2
3   public class DecToBinModel
4   {
5       /**
6        * converts a given number to bin
7        *
8        * @param num
9        *              given number
10       *
11       * @return given number in bin
12       */
13      public String convertToBin(int num)
14      {
15          return Integer.toBinaryString(num);
16      }
17  }
```

Listing 20: Sourcecode des DecToBin Models

```
1   package model;
2
3   public class DecToHexModel
4   {
5       /**
6        * converts a given number to hex
7        *
8        * @param num
```

```
9         *                 given number
10        *
11        * @return given number in hex
12        */
13       public String convertToHex(int num)
14       {
15           return Integer.toHexString(num);
16       }
17   }
```

Listing 21: Sourcecode des DecToHex Models

5.4 Sourcecode Main

```
1   package main;
2
3   import controller.DecToBinController;
4   import controller.DecToHexController;
5   import controller.AdderController;
6
7   public class Main
8   {
9       /**
10       * @param unused
11       */
12      public static void main(String[] unused)
13      {
14          // init Controllers
15          AdderController mainCon = new AdderController();
16          DecToBinController binCon = new DecToBinController();
17          DecToHexController hexCon = new DecToHexController();
18
19          // add Observers to mainCon
20          mainCon.addObserver(hexCon);
21          mainCon.addObserver(binCon);
22
23          // add Observers to binCon
24          binCon.addObserver(mainCon);
25
26          // add Observers hexCon
27          hexCon.addObserver(mainCon);
28      }
29  }
```

Listing 22: Sourcecode der Main