Assignment 4

Object-Oriented Design, IV1350

Emil Tullstedt emiltu@kth.se

2014-05-15

Contents

1 Introduction

This report describes the implementation of a previously designed cache simulator made for a pre-graduate project in the course **IV1350** at KTH in Stockholm. The implementation is not for production usage.

While implementing the application, I co-operated with *Martin Alge*, *Jesper Falk* and *Erik Pettersson*.

2 Method

2.1 Design Patterns

The theory behind design patterns is that there's a certain amount of abstract requirements that developers will face more often than others. When a requirement is often occurring for developers of a specific paradigm or language due to the nature of the applications that are built or the language itself, that requirement justifies a *pattern*.

The *pattern* is a tried way of solving a common issue that may be applied on different kinds of implementations, so the very same pattern that's applied to a rocket propellant system may also be used to program interactive Tamagochi-toys.

In object-oriented programming, due to the modular structure of the programming paradigm, patterns are quite commonly occurring and is thus part of any self-respecting object-orientational programmer's toolkit. During the mid 1990's, the book *"Design Patterns: Elements of Reusable Object-Oriented Software"* was released by four authors, who subsequently became known as the *Gang of Four*. The patterns they proposed is commonly referred to as GoF-patterns, and are widely-spread.

As per further developing the Cache Simulator implemented in *Assignment 3*[1] one of the requirements was to understand and implement design patterns. The application was already implementing the *Singleton*-pattern in the `Storage`-class and an implementation of the *Observer*-pattern was added to the `DataCache`-class and the `View`. The theory behind and the implementation of the patterns will be explained in the subsections 2.1.1 and 2.1.2.

2.1.1 Storage as Singleton

The singleton is a pattern describing a class that describes a single object. The usage of the singleton is to find a middle-ground between a static class and a normal class. Implementing an enforcing singleton class is made by having a non-public constructor which is called when initializing a static variable containing the singleton itself. This variable may then be fetched by a static getter-method if you need to keep a good encapsulation.

For every advantage for using a singleton, there's quite often an equivalently strong disadvantage, such as singleton-classes quite often later are found being needed as normal classes, which can require a substantial rewrite of the application. There's also the issue of introducing a global state to the object oriented nature of the program, which by purist object oriented programmers is seen as a procedural programming plague.[2] Without arguing for or against the usage of singletons, it is widely used, and it's place in *"Design Patterns"* by Gang of Four is hard to argue against.

When implementing the singleton (patch file is seen in it's entirety at 3.8.1) the nature of the assignment called for several design choices, primarily, the `Storage`'s construction isn't thread-safe or memory-efficient (it'll probably never be collected by the garbage collection).

---

[1] http://web.ict.kth.se/~emiltu/iv1350-emiltu-s3.pdf
[2] See for example https://sites.google.com/site/steveyegge2/singleton-considered-stupid

2.1.2 Observer Observing DataCache

The *observer* pattern attempts to solve the issue where an entity needs to track another entity for updates without having to constantly poll the other entity. The elegance of the *observer* pattern is how it avoids causing a mess but still slightly sidestepping the notion that the *models* never may communicate directly with the *view*. The *observer* uses a pre-defined interface (for reference, see 3.3.2) that defines how a view that wishes to be able to observe a observable object must implement a specific class.

Except for the slightly confusing concept of sidestepping the rules of *MVC*, the observer-pattern also has the disadvantage that it's dependent on the observable object having access to call methods in the observing object and sometimes sending an unnecessary amount of data. Say for example a debugger which tracks memory, there polling might be a good idea to prevent a denial of service-attack caused by an overflow of updates being sent from the debugger when memory locations are changed. In this case, a second-wise poll may be an efficient solution. Another issue is that the observer pattern builds upon a mutually agreed upon API between anything that observes and the observed object, the creators of views must conform to a code style which they may not have agreed to. Whether this is an issue or not is a quite philosophic question.

The implementation of the *observer* pattern in the cache simulator application is kept as simple as possible, where the `DataCache` (3.3.1) stores a list of `DataCacheObserver`-implementors (3.3.2) which are added and removed by a public function, which is also possibly being relayed through the `Controller` (3.1.1). The `View` is implemented as the observer, and is added to the list of observers on the user's request. Whenever the Data-Cache is updated (i.e. on misses when loading or storing data from it), the public function `recvDataCacheUpdates(String s)` is called with the content of the new cache to any observer.

2.2 Exceptions

3 Result

3.1 Controller and View

3.1.1 Class: Controller

The `Controller` has been updated to communicate adding and removing observers for the `DataCache` (3.3.1) and handling `IllegalAddressLayout`. It also treats `Storage` (3.2.1) as a singleton-object correctly.

```
1  package is.mjuk.cache;
2
3  /**
4   * Controller for communication between storage, models and view.
5   */
6  public class Controller
7  {
8      private User user;
9      private CacheLayout cacheLayout;
10     private DataCache dataCache;
11     private AddressLayout addressLayout;
12     private Storage store = Storage.getStorage();
13
14     /**
15      * Constructs a controller with an embedded
16      * {@link is.mjuk.cache.User}-object
17      */
18     public Controller() {
19         user = new User();
20     }
21
22     /**
23      * Returns the {@link is.mjuk.cache.User}-object's stored datetime
24      * as a string.
25      */
26     public String getDateTimeString() {
27         return user.getDateTime().toString();
28     }
29
30     /**
31      * Creates an Instruction object and executes a single instruction
32      * <p>
33      * Creates an Instruction object and loads it with the datacache and
34      * addresslayout objects and then requests it to perform it's
35      * instruction which status is sent through back through a
36      * {@link is.mjuk.cache.InstructionDTO} object which is subsequently
37      * stored in the {@link is.mjuk.cache.Storage}-singleton and returned
38      * to the callee.
39      *
40      * @throws is.mjuk.cache.IllegalAddressException For unparseable addresses.
41      * @param type String containing data of instruction type
42      * (either store or load)
```

```
43        * @param address A long containing the address to perform the
44        * instruction on.
45        * @return {@link is.mjuk.cache.InstructionDTO} containing data about
46        * the instruction.
47        * @see is.mjuk.cache.Instruction
48        */
49       public InstructionDTO executeInstruction(String type, long address)
50       throws IllegalAddressException {
51           Instruction instruction;
52
53           if(type.equals("load")) {
54               instruction = new Instruction(dataCache, addressLayout,
55                   InstructionType.LOAD, address);
56           } else if (type.equals("store")) {
57               instruction = new Instruction(dataCache, addressLayout,
58                   InstructionType.STORE, address);
59           } else {
60               InstructionDTO rv = null;
61               return rv;
62           }
63
64           InstructionDTO instructionDTO = instruction.executeInstruction();
65
66           store.addInstructionDTO(instructionDTO);
67
68           return instructionDTO;
69       };
70
71       /**
72        * Gets hitrate from the datacache object
73        */
74       public double getHitrate() {
75           return dataCache.getHitrate();
76       }
77
78       /**
79        * Calculates and displays missrate.
80        * <p>
81        * Subtracts hitrate from 1.
82        */
83       public double getMissrate() {
84           double hitrate = dataCache.getHitrate();
85           return 1.00 - hitrate;
86       }
87
88       /**
89        * Setter for the user-property's nickname.
90        * @see is.mjuk.cache.User#setNickname(String newNickname)
91        */
92       public void setNickname(String newNick) {
93           user.setNickname(newNick);
94       }
95
96       /**
97        * Getter for the user-property's nickname.
98        * @see is.mjuk.cache.User#getNickname()
99        */
100      public String getNickname() {
101          return user.getNickname();
```

```
102         }
103
104         /**
105          * WARNING: Will flush cache. Updates the controller's cache layout
106          * object.
107          * <p>
108          * Updates the object containing a {@link is.mjuk.cache.CacheLayout}
109          * contained within the controller. Also updates the
110          * {@link is.mjuk.cache.AddressLayout} and
111          * {@link is.mjuk.cache.DataCache}.
112          * Stores the {@link is.mjuk.cache.LayoutDTO} of the created cache
113          * layout in the {@link is.mjuk.cache.Storage} singleton.
114          */
115         public void setCacheLayout(int blockSize, int blockCount,
116             int associativity) {
117             cacheLayout = new CacheLayout(blockSize, blockCount, associativity);
118             addressLayout = cacheLayout.getAddressLayout();
119             dataCache = cacheLayout.getDataCache();
120             store.storeLayoutDTO(this.cacheLayout.generateLayoutDTO());
121         }
122
123         /**
124          * Returns a string representation of the cache.
125          */
126         public String displayCache() {
127             return dataCache.displayCache();
128         }
129
130         /**
131          * Creates and returns data relevant to the simulation as a
132          * {@link is.mjuk.cache.SimulationDTO} object.
133          * @see is.mjuk.cache.SimulationDTO
134          */
135         public SimulationDTO getSimulationData(){
136             store.storeHitrate(this.dataCache.getHitrate());
137             store.storeNickname(this.user.getNickname());
138             store.storeDateTime(this.user.getDateTime());
139             store.storeLayoutDTO(this.cacheLayout.generateLayoutDTO());
140
141             SimulationDTO simDTO = store.createDTO();
142             simDTO.setStores(this.dataCache.getStores());
143             simDTO.setLoads(this.dataCache.getLoads());
144             return simDTO;
145         }
146
147         /**
148          * Adds an observer-object to the DataCache-object
149          * @param observer A {@link is.mjuk.cache.DataCacheObserver}-implementation
150          * @see is.mjuk.cache.DataCache#addObserver
151          */
152         public void addDataCacheObserver(DataCacheObserver observer) {
153             this.dataCache.addObserver(observer);
154         }
155
156         /**
157          * Removes an observer-object from the DataCache-object
158          * @param observer A {@link is.mjuk.cache.DataCacheObserver}-implementation
159          * @see is.mjuk.cache.DataCache#removeObserver
160          */
```

```
161      public void removeDataCacheObserver(DataCacheObserver observer) {
162          this.dataCache.removeObserver(observer);
163      }
164  }
```

3.1.2 Class: View

The `View` has been updated to act as an observer for `DataCache` (3.3.1)

```
 1  package is.mjuk.cache;
 2
 3  import java.util.Scanner;
 4  import java.util.Date;
 5
 6  /**
 7   * User interaction class
 8   * <p>
 9   * Handles interaction between user and the application
10   */
11  public class View implements DataCacheObserver
12  {
13      public static Scanner scanner = new Scanner(System.in);
14      Controller c;
15
16      /**
17       * Constructs the View
18       * <p>
19       * Creates a new view and attaches it to a {@link is.mjuk.cache.Controller}
20       * and then launches the text-based UI of the application.
21       *
22       * @param controller Controller for communication with the application logic
23       */
24      public View(Controller controller)
25      {
26          this.c = controller;
27          this.requireNickname();
28          this.getCacheInformation();
29          this.getUserInstruction();
30          this.endSimulation();
31      }
32
33      private void requireNickname()
34      {
35          System.out.println("ASKS_USER_TO_ENTER_USER_PROPERTIES");
36          System.out.println("---------------------------------\n");
37          System.out.println("Please_enter_nickname:_");
38          String newNick = scanner.nextLine();
39
40          c.setNickname(newNick);
41          System.out.println("Your_nickname_is:_" + c.getNickname());
42      }
43
44
45
46      private void getCacheInformation()
47      {
48          boolean legalData = true;
49
50          System.out.println("\nUSER_SPECIFIES_BLOCK_PROPERTIES");
```

```
51          System.out.println("-----------------------------\n");
52       do {
53            System.out.println("Enter block size in bytes: ");
54            int blockSize = scanner.nextInt();
55            System.out.println("Enter block count: ");
56            int blockCount = scanner.nextInt();
57            System.out.println("Enter associativity: ");
58            int associativity = scanner.nextInt();
59
60            System.out.println("CALCULATES CACHE LAYOUT & CREATES CACHE");
61            try{
62                c.setCacheLayout(blockSize, blockCount, associativity);
63                legalData = true;
64            }
65            catch(java.lang.IllegalArgumentException e){
66                System.out.println("Illegal Data was entered.");
67                System.out.println("Block Count and Block Size must be powers"
68                    + " of two.");
69                legalData = false;
70            }
71       } while (!legalData);
72
73       System.out.println("Displaying Cache Data");
74       System.out.println(c.displayCache());
75    }
76
77    private void getUserInstruction()
78    {
79       System.out.println("USER INPUTS INSTRUCTIONS");
80       System.out.println("Write 'exit' to stop the application");
81         System.out.println(
82           "To use instruction load, write 'load <memaddress>'"
83       );
84         System.out.println(
85           "To use instruction store, write 'store <memaddress>'"
86       );
87       System.out.println(
88           "To observe changes in the cache, write 'observe'"
89       );
90       System.out.println(
91           "To stop observing changes in the cache, write 'deobserve'"
92       );
93
94       while (true) {
95            String input = scanner.nextLine();
96            input = input.trim();
97
98            String regex = "^[ls](oad|tore)?\\s(0[x]?)?[0-9a-fA-F]+";
99
100           if (input.toLowerCase().equals("exit")
101               || input.toLowerCase().equals("x")) {
102              break;
103           } else if (input.matches(regex)) {
104              sendInstruction(input);
105           } else if (input.matches("^$")) {
106              // Intentionally left empty
107           } else if (input.equals("observe")) {
108              System.out.println("Now observing the DataCache for changes");
109              c.addDataCacheObserver(this);
```

```
110            } else if (input.equals("deobserve")) {
111                System.out.println("No_longer_observing_the_DataCache");
112                c.removeDataCacheObserver(this);
113            } else {
114                System.err.println("Instruction_not_found_'" + input + "'");
115            }
116        }
117    }
118
119    private void endSimulation() {
120        System.out.println(c.displayCache());    // TODO: Remove
121        SimulationDTO simDTO = c.getSimulationData();
122        System.out.println("Simulation_data:");
123        System.out.println("Username:_" + simDTO.getNickname());
124        System.out.println("Load_instructions:_" + simDTO.getLoads());
125        System.out.println("Store_instructions:_" + simDTO.getStores());
126        System.out.println("Hit_rate:_" + simDTO.getHitrate());
127        System.out.println("Miss_rate:_" + simDTO.getMissrate());
128        System.out.println("Block_Size:_"
129            + simDTO.getLayoutDTO().getBlockSize() + "_bytes");
130        System.out.println("Block_Count:_"
131            + simDTO.getLayoutDTO().getBlockCount() + "_blocks");
132        System.out.println("Associativity:_"
133            + simDTO.getLayoutDTO().getAssociativity());
134        System.out.println("Address_tag_size:_"
135            + simDTO.getLayoutDTO().getTagSize() + "_bits");
136        System.out.println("Address_index_size:_"
137            + simDTO.getLayoutDTO().getIndexSize() + "_bits");
138        System.out.println("Address_offset_size:_"
139            + simDTO.getLayoutDTO().getOffsetSize() + "_bits");
140    }
141
142    /**
143     * recvDataCacheUpdate recieves data from the DataCache observer and
144     * prints it on the screen
145     * @param dataCacheContent New content of the DataCache.
146     */
147    public void recvDataCacheUpdate(String dataCacheContent) {
148        System.out.println(dataCacheContent);
149    }
150
151    private void sendInstruction(String input) {
152        long address = Long.decode(input.split("\\s")[1]);
153
154        try {
155            if (input.split("\\s")[0].matches("^l(oad)?$")) {
156                System.out.println(
157                    c.executeInstruction("load", address).toString()
158                );
159            } else if (input.split("\\s")[0].matches("^s(tore)?$")) {
160                System.out.println(
161                    c.executeInstruction("store", address).toString()
162                );
163            }
164        } catch(IllegalAddressException e) {
165            System.out.println("Error_parsing_memory_address:_" + e);
166            return;
167        }
168
```

```
169          System.out.printf(
170              "Hitrate is %.2f and missrate is %.2f.\n",
171              c.getHitrate(), c.getMissrate()
172          );
173      }
174  }
```

3.2 Singleton

3.2.1 Class: Storage

The `Storage` was written as a *singleton* from the very beginning and hasn't been changed since **Assignment 3**. Explanation of the implementation may be found in 2.1.1.

```
1   package is.mjuk.cache;
2
3   import java.util.ArrayList;
4   import java.util.Date;
5
6   /**
7    * Templete for storage of data
8    * <p>
9    * Stores different variables for logging and
10   * storage to database or likewise.
11   * @author  Emil Tullstedt <emiltu@kth.se>
12   */
13  public class Storage {
14      private static final Storage storage = new Storage();
15      private ArrayList<InstructionDTO> instructionStore;
16      private LayoutDTO layoutStore;
17      private String nickname;
18      private Date datetime;
19      private double hitrate;
20
21      private Storage() {
22          this.instructionStore = new ArrayList<InstructionDTO>();
23      }
24
25      /**
26       * Gets the saved storage
27       */
28      public static Storage getStorage() {
29          return storage;
30      }
31
32      /**
33       * Adds an {@link is.mjuk.cache.InstructionDTO} to the
34       * list of instructions.
35       *
36       * @param instruction {@link is.mjuk.cache.InstructionDTO} to be stored.
37       */
38      public void addInstructionDTO (InstructionDTO instruction) {
39          this.instructionStore.add(instruction);
40      }
41
42      /**
43       * Get one {@link is.mjuk.cache.InstructionDTO} from the
44       * list of saved InstructionDTOs.
```

```
45            *
46            * @param count The index of the DTO to be retrieved
47            */
48           public InstructionDTO getInstructionDTO (int count) {
49               return instructionStore.get(count);
50           }
51
52           /**
53            * Sets the {@link is.mjuk.cache.LayoutDTO} of the data
54            *
55            * @param layout {@link is.mjuk.cache.LayoutDTO} to be stored.
56            */
57           public void storeLayoutDTO(LayoutDTO layout) {
58               this.layoutStore = layout;
59           }
60
61           /**
62            * Get the saved {@link is.mjuk.cache.LayoutDTO}
63            */
64           public LayoutDTO getLayoutDTO() {
65               return this.layoutStore;
66           }
67
68           /**
69            * Set the date to store
70            *
71            * @param datetime The date to set to
72            */
73           public void storeDateTime(Date datetime) {
74               this.datetime = datetime;
75           }
76
77           /**
78            * Get the stored Date
79            */
80           public Date getDateTime() {
81               return this.datetime;
82           }
83
84           /**
85            * Set hitrate to store
86            *
87            * @param hitrate The hitrate to store
88            */
89           public void storeHitrate(double hitrate) {
90               this.hitrate = hitrate;
91           }
92
93           /**
94            * Get the stored date
95            */
96           public double getHitrate() {
97               return this.hitrate;
98           }
99
100          /**
101           * Set nickname to store
102           */
103          public void storeNickname(String nickname) {
```

```
104            this.nickname = nickname;
105        }
106
107        /**
108         * Get the stored nickname
109         */
110        public String getNickname() {
111            return this.nickname;
112        }
113
114        /**
115         * Creates a {@link is.mjuk.cache.SimulationDTO} from all saved data
116         */
117        public SimulationDTO createDTO() {
118            return new SimulationDTO (hitrate, nickname, datetime, layoutStore);
119        }
120
121        /**
122         * Reset all variables to emty instances
123         */
124        public void clean() {
125            instructionStore = new ArrayList<InstructionDTO>();
126            layoutStore = new LayoutDTO();
127            datetime = new Date();
128            nickname = new String();
129            hitrate = Double.NaN;
130        }
131    }
```

3.3 Observer

3.3.1 Class: DataCache

The `DataCache` is tested in 3.6.3.

```
 1  package is.mjuk.cache;
 2
 3  import java.util.ArrayList;
 4  import java.lang.StringBuilder;
 5  import java.lang.Math;
 6
 7  /**
 8   * Stores and operates on cache blocks
 9   * <p>
10   * Creates a set of cache blocks which can be accessed using indexes.
11   * Supports associativity (i.e. multiple blocks with same index)
12   *
13   */
14  public class DataCache {
15      private int hits = 0;
16      private int misses = 0;
17      private int loads = 0;
18      private int stores = 0;
19      private Block[][] blockset;
20      private ArrayList<DataCacheObserver> observers =
21          new ArrayList<DataCacheObserver>();
22
23      /**
```

```
24          * Parses a cache layout and generates the block objects
25          *
26          * @param layout Contains the layout for the cache in a
27          * {@link is.mjuk.cache.LayoutDTO}
28          */
29         public DataCache(LayoutDTO layout) {
30             this.blockset =
31                 new Block[layout.getAssociativity()][layout.getBlockCount()];
32             for (int i = 0; i < this.blockset[0].length; i++) {
33                 for (int ii = 0; ii < this.blockset.length; ii++) {
34                     blockset[ii][i] = new Block();
35                 }
36             }
37         }
38
39         /**
40          * Returns the data in the cache as a string
41          * <p>
42          * Returns every {@link is.mjuk.cache.Block} of the cache in string
43          * representation.
44          *
45          */
46         public String displayCache() {
47             StringBuilder cacheDisplay = new StringBuilder();
48             for (int i = 0; i < this.blockset[0].length; i++) {
49                 cacheDisplay.append("Index␣0x" + Integer.toString(i, 16));
50                 cacheDisplay.append(":␣");
51                 for (int ii = 0; ii < this.blockset.length; ii++) {
52                     cacheDisplay.append(blockset[ii][i].toString() + "␣");
53                 }
54                 cacheDisplay.append("\n");
55             }
56             return cacheDisplay.toString();
57         }
58
59         /**
60          * Adds an {@link is.mjuk.cache.DataCacheObserver} to the list of observers.
61          * <p>
62          * Silently ignores readding the observer if the observer is already in the
63          * list.
64          *
65          * @param observer Observer to be added to the list of observers.
66          */
67         public void addObserver(DataCacheObserver observer) {
68             if (!this.observers.contains(observer)) {
69                 this.observers.add(observer);
70             }
71         }
72
73         /**
74          * Removes an {@link is.mjuk.cache.DataCacheObserver} from the list of
75          * observers.
76          * <p>
77          * Silently ignores removing the observer if the observer doesn't exist in
78          * the list of observers.
79          *
80          * @param observer Observer to be removed from the list of observers
81          */
82         public void removeObserver(DataCacheObserver observer) {
```

```
83              if (this.observers.contains(observer)) {
84                  this.observers.remove(observer);
85              }
86          }
87
88          /**
89           * @return Number of sets in the cache (associativity)
90           */
91          public int getNumberOfSets() {
92              return this.blockset.length;
93          }
94
95          /**
96           * @return Number of blocks in each set (block count)
97           */
98          public int getNumberOfBlocks() {
99              return this.blockset[0].length;
100         }
101
102         /**
103          * @return Number of hits since creation of DataCache
104          */
105         public int getHits() {
106             return this.hits;
107         }
108
109         /**
110          * @return Number of misses since creation of DataCache
111          */
112         public int getMisses() {
113             return this.misses;
114         }
115
116         /**
117          * Returns the hitrate of the cache
118          * <p>
119          * Calculates number of hits divided by number of hits and misses.
120          * Starts at zero if there are no hits nor misses.
121          *
122          * @return Hitrate since creation of DataCache
123          */
124         public double getHitrate() {
125             if ((this.hits+this.misses) == 0) {
126                 return 0.0;
127             }
128
129             return (double) this.hits / (double) (this.hits+this.misses);
130         }
131
132         /**
133          * @return Number of store operations on cache since creation
134          */
135         public int getStores() {
136             return this.stores;
137         }
138
139         /**
140          * @return Number of load operations on cache since creation
141          */
```

```
142        public int getLoads() {
143            return this.loads;
144        }
145
146        /**
147         * Loads data from a specific cache address
148         * <p>
149         * Increments the load counter and then checks if a single position
150         * in the cache is already existing. If it's not, updates cache
151         * accordingly
152         */
153        public boolean loadData(AddressDTO address){
154            this.loads += 1;
155            return updateCachePosition(address);
156        }
157
158        /**
159         * Stores data at a specific cache address
160         * <p>
161         * Increments the store counter and then checks if a single position
162         * in the cache is already existing. If it's not, updates cache
163         * accordingly
164         */
165        public boolean storeData(AddressDTO address){
166            this.stores += 1;
167            return updateCachePosition(address);
168        }
169
170        private boolean updateCachePosition(AddressDTO address) {
171            int cacheSet = -1;
172            Block currentBlock;
173
174            for (int i = 0; i < this.blockset.length; i++) {
175                currentBlock = this.blockset[i][(int) address.getIndex()];
176
177                if (currentBlock.isValid(address.getTag())) {
178                    this.hits += 1;
179                    return true;
180                } else if (currentBlock.isValid() == false) {
181                    cacheSet = i;
182                }
183            }
184
185            if (cacheSet == -1) {
186                cacheSet = (int) Math.floor(Math.random()
187                    * this.blockset.length);
188            }
189
190            currentBlock = this.blockset[cacheSet][(int) address.getIndex()];
191            currentBlock.setTag(address.getTag());
192
193            this.notifyObservers();
194
195            this.misses += 1;
196            return false;
197        }
198
199        private void notifyObservers() {
200            for (DataCacheObserver observer : this.observers) {
```

```
201                    observer.recvDataCacheUpdate(this.displayCache());
202            }
203        }
204 }
```

3.3.2 New Interface: DataCacheObserver

```
1  package is.mjuk.cache;
2
3  /**
4  * DataCacheObserver is an interface for objects that wish to observe a
5  * {@link is.mjuk.cache.DataCache}-object
6  *
7  * @see is.mjuk.cache.DataCache
8  */
9  public interface DataCacheObserver {
10
11     /**
12     * The observed object will call this on the observing object whenever a
13     * observeable event occurs.
14     * @param dataCacheContent Updated content of the observed object
15     */
16     public void recvDataCacheUpdate(String dataCacheContent);
17 }
```

3.4 Exceptions

3.4.1 Class: AddressLayout

```
1  package is.mjuk.cache;
2
3  import is.mjuk.utils.MisMath;
4
5  /**
6  * Stores data relevant to parsing addresses for the cache
7  */
8  public class AddressLayout {
9
10     private int tagSize;
11     private int indexSize;
12     private int offsetSize;
13
14     public AddressLayout(int tag, int index, int offset) {
15         this.tagSize = tag;
16         this.indexSize = index;
17         this.offsetSize = offset;
18     }
19
20     /**
21     * Splits an address to an {@link is.mjuk.cache.AddressDTO}
22     * <p>
23     * The address is split into a tag, index and offset to be used when
24     * locating the correct cache block for storing/loading the data.
25     *
26     * @see is.mjuk.cache.AddressDTO
27     * @param address Address to be parsed into an AddressDTO
28     * @return An {@link is.mjuk.cache.AddressDTO} containing the
```

19(56)

```
29          * tag, index and offset of the input address
30          * @throws IllegalAddressException For unparseable addresses
31          */
32         public AddressDTO parseAddress (long address) throws IllegalAddressException
33         {
34             if ((0b11 & address) != 0x0) {
35                 String error = "Memory␣address␣0x" + Long.toString(address, 16)
36                     + "␣is␣not␣divisible␣by␣four␣and␣do␣not␣point␣at␣a␣valid"
37                     + "␣block␣address.";
38                 throw new IllegalAddressException(error);
39             }
40
41             AddressDTO rv = new AddressDTO();
42             rv.setOffset(MisMath.intToUnary(this.offsetSize) & address);
43             rv.setIndex(MisMath.intToUnary(this.indexSize) & address
44                 >>> offsetSize);
45             rv.setTag(MisMath.intToUnary(this.tagSize) & address
46                 >>> offsetSize + indexSize);
47             return rv;
48         }
49
50         /**
51          * Returns the number of bits in the address tag
52          *
53          * @return Bits in address tag
54          */
55         public int getTagSize() {
56             return this.tagSize;
57         }
58
59         /**
60          * Returns the number of bits in the address index
61          *
62          * @return Bits in address index
63          */
64         public int getIndexSize() {
65             return this.indexSize;
66         }
67
68         /**
69          * Returns the number of bits in the address offset
70          *
71          * @return Bits in the address offset
72          */
73         public int getOffsetSize() {
74             return this.offsetSize;
75         }
76
77 }
```

3.4.2 New Exception: IllegalAddressException

Exception-class for creating IllegalAddressExceptions for the AddressLayout when
an address is unparseable.

```
1  package is.mjuk.cache;
2
3  /**
4   * IllegalAddressException is a throwable exception for addresses that's not
5   * legally readable.
```

```
 6   * <p>
 7   * For example, addresses that's not evenly divisible by the word size,
 8   * are out of bounds or simply doesn't make sense.
 9   */
10  public class IllegalAddressException extends Exception {
11          /**
12           * Constructor for an IllegalAddressException called without any error
13           * errormessage or a cause-throwable attached to it.
14           */
15          public IllegalAddressException() {
16                  super();
17          }
18
19          /**
20           * Constructor containing a message for specifying details.
21           * @param errormsg Details of the exception.
22           */
23          public IllegalAddressException(String errormsg) {
24                  super(errormsg);
25          }
26  }
```

3.4.3 Class: Instruction

Instruction has gotten a minor change to throw any IllegalAddressException that's thrown by AddressLayout (3.4.1).

```
 1  package is.mjuk.cache;
 2
 3  /**
 4   * A single cache instruction.
 5   * <p>
 6   * The class is handling execution of a single instruction to be
 7   * performed on the inputted {@link is.mjuk.cache.DataCache}.
 8   */
 9  public class Instruction {
10      private DataCache dataCache;
11      private InstructionType type;
12      private AddressDTO address;
13
14      /**
15       * Parses address and prepares the instruction for execution.
16       *
17       * @param dataCache {@link is.mjuk.cache.DataCache} to be used for
18       * executing instruction.
19       * @param addressLayout {@link is.mjuk.cache.AddressLayout} for
20       * parsing the address parameter in order to be able to successfully
21       * select the correct block in the cache.
22       * @param type Type of the instruction to be executed on the cache.
23       * See {@link is.mjuk.cache.InstructionType} for available types.
24       * @param address Address of the destinated memory block.
25       * @throws IllegalAddressException Thrown for unparseable addresses.
26       */
27      public Instruction(DataCache dataCache, AddressLayout addressLayout,
28          InstructionType type, long address)
29      throws IllegalAddressException {
30          this.type = type;
31
```

```
32            this.address = addressLayout.parseAddress(address);
33
34            this.dataCache = dataCache;
35        }
36
37        /**
38         * Executes the instruction
39         * <p>
40         * Creates a call to the datacache requesting the data for
41         * the specified address.
42         *
43         * @return {@link is.mjuk.cache.InstructionDTO}
44         */
45        public InstructionDTO executeInstruction() {
46            boolean hit = false;
47
48            if (this.type == InstructionType.LOAD) {
49                hit = this.dataCache.loadData(this.address);
50            } else if (this.type == InstructionType.STORE) {
51                hit = this.dataCache.storeData(this.address);
52            }
53
54            InstructionDTO rv = new InstructionDTO(hit, this.address, this.type);
55
56            return rv;
57        }
58
59
60 }
```

3.5 Unchanged Files

These files have not been changed since assignment 3.

3.5.1 Class: AddressDTO

```
1  package is.mjuk.cache;
2
3  /**
4   * Data relevant for parsing a single address
5   */
6  public class AddressDTO {
7      private long tag;
8      private long index;
9      private long offset;
10
11     /**
12      * Get the tag of this address
13      */
14     public long getTag() {
15         return this.tag;
16     }
17
18     /**
19      * Set the tag of this address
20      *
```

```
21         * @param tag tag to use
22         */
23        public void setTag(long tag) {
24            this.tag = tag;
25        }
26
27        /**
28         * Get the index of this address
29         */
30        public long getIndex() {
31            return this.index;
32        }
33
34        /**
35         * Set the index of this address
36         *
37         * @param index index to use
38         */
39        public void setIndex(long index) {
40            this.index = index;
41        }
42
43        /**
44         * Get the offset of this address
45         */
46        public long getOffset() {
47            return this.offset;
48        }
49
50        /**
51         * Set the offset of this address
52         *
53         * @param offset the offset to use
54         */
55        public void setOffset(long offset) {
56            this.offset = offset;
57        }
58    }
```

3.5.2 Class: Block

```
1  package is.mjuk.cache;
2
3  import java.lang.StringBuilder;
4
5  /**
6  * A single block within a cache
7  * <p>
8  * Stores a tag with a memory address of a single unit of data
9  * (when combined) with index.
10 */
11 public class Block {
12     private boolean validity;
13     private long tag;
14
15     /**
16      * Initializes an empty block
17      */
18     public Block()
```

```
19        {
20            this.validity = false;
21            this.tag = 0x00000000;
22        }
23
24        /**
25         * Checks an input tag to see if block is containing the same tag
26         * and is valid.
27         *
28         * @param tag Tag to be checked against block
29         */
30        public boolean isValid(long tag) {
31            if (this.tag == tag && this.validity == true) {
32                return true;
33            } else {
34                return false;
35            }
36        }
37
38        /**
39         * Checks the block if it's validity property is set to true
40         */
41        public boolean isValid() {
42            if (this.validity == true) {
43                return true;
44            } else {
45                return false;
46            }
47        }
48
49        /**
50         * Updates the content of the block.
51         * <p>
52         * Sets the tag and the validity to reflect change in the cacheblock.
53         *
54         * @param tag Address-tag to be stored
55         * @param validity New validity of the tag
56         */
57        public void setData(long tag, boolean validity) {
58            this.validity = validity;
59            this.tag = tag;
60        }
61
62        /**
63         * Change the validity of the tag.
64         *
65         * @param validity New validity of the tag.
66         */
67        public void setValidity(boolean validity) {
68            this.validity = validity;
69        }
70
71        /**
72         * Saves a single tag
73         * <p>
74         * Also sets <code>validity</code> to <i>true</i>.
75         *
76         * @see is.mjuk.cache.Block#setData
77         * @param tag Address-tag to be saved
```

```
78        */
79       public void setTag(long tag) {
80            setData(tag, true);
81       }
82
83       /**
84        * Returns a string representation of the block
85        *
86        * @return A string containing the validity and tag of the block
87        */
88       public String toString() {
89            StringBuilder rv = new StringBuilder();
90            if (this.validity) {
91                rv.append("valid");
92            } else {
93                rv.append("invalid");
94            }
95            rv.append(" 0x");
96            rv.append(Long.toString(this.tag, 16));
97            return rv.toString();
98       }
99  }
```

3.5.3 Class: CacheLayout

CacheLayout's comments has been changed minorly.

```
1  package is.mjuk.cache;
2
3  import is.mjuk.utils.MisMath;
4  import java.lang.IllegalArgumentException;
5
6  /**
7   * Template for a cache
8   * <p>
9   * Calculating the necessary parameters to generate a datacache and
10  * parsing addresses for accessing the cache.
11  * @author   Emil Tullstedt <emiltu@kth.se>
12  * @version  0.1
13  */
14 public class CacheLayout
15 {
16      private static final int MEMORY_ADDRESS_SIZE = 32;
17
18      private int blockSize;
19      private int blockCount;
20      private int associativity;
21
22      private AddressLayout addressLayout = null;
23      private DataCache dataCache = null;
24
25      /**
26       * Constructor for CacheLayout
27       * <p>
28       * Setting the internal values to specified data.
29       *
30       * @param blockSize Amount of <i>bytes</i> in a single block
31       * @param blockCount Amount of blocks in the cache
32       * @param associativity Associativity of the cache
33       */
```

```java
34        public CacheLayout(int blockSize, int blockCount, int associativity) {
35            this.blockSize = blockSize;
36            this.blockCount = blockCount;
37            this.associativity = associativity;
38        }
39
40        /**
41         * Generating a object containing the properties of the cache and
42         * it's address.
43         *
44         *
45         * @return A LayoutDTO containing the block size, block count,
46         * associativity, tag size, index size and offset size of the
47         * cache.
48         */
49        public LayoutDTO generateLayoutDTO()
50        {
51            addressLayout = this.getAddressLayout();
52            LayoutDTO dto = new LayoutDTO();
53            dto.setBlockSize(this.blockSize);
54            dto.setBlockCount(this.blockCount);
55            dto.setAssociativity(this.associativity);
56            dto.setTagSize(addressLayout.getTagSize());
57            dto.setIndexSize(addressLayout.getIndexSize());
58            dto.setOffsetSize(addressLayout.getOffsetSize());
59            return dto;
60        }
61
62        /**
63         * Returns the {@link is.mjuk.cache.AddressLayout} of the current
64         * CacheLayout. The AddressLayout is generated if it's not defined
65         * already.
66         *
67         * @return {@link is.mjuk.cache.AddressLayout}
68         */
69        public AddressLayout getAddressLayout() {
70            if (this.addressLayout == null) {
71                this.addressLayout = this.calculateAddressLayout();
72            }
73
74            return this.addressLayout;
75        }
76
77        /**
78         * Returns the {@link is.mjuk.cache.DataCache} associated with the
79         * CacheLayout-object. The DataCache is generated if it's not defined
80         * already.
81         *
82         * @return {@link is.mjuk.cache.DataCache}
83         */
84        public DataCache getDataCache()
85        {
86            if (this.dataCache == null) {
87                this.dataCache = this.generateDataCache();
88            }
89
90            return this.dataCache;
91        }
92
```

26(56)

```
93        private DataCache generateDataCache() {
94            LayoutDTO layout = this.generateLayoutDTO();
95            return new DataCache(layout);
96        }
97
98        private AddressLayout calculateAddressLayout() {
99            int offset = 0;
100           int index = 0;
101
102           if (MisMath.log2(this.blockSize) % 1.00 == 0.00) {
103               offset = (int) MisMath.log2(this.blockSize);
104           } else {
105               throw new IllegalArgumentException();
106           }
107
108           if (MisMath.log2(this.blockCount) % 1.00 == 0.00) {
109               index = (int) MisMath.log2(this.blockCount);
110           } else {
111               throw new IllegalArgumentException();
112           }
113
114           int tag = MEMORY_ADDRESS_SIZE - (index + offset);
115           if (tag < 0) {
116               throw new IllegalArgumentException();
117           }
118
119           AddressLayout rv = new AddressLayout(tag, index, offset);
120           return rv;
121       }
122
123   }
```

3.5.4 Class: CacheSimulator

```
1   package is.mjuk.cache;
2
3   /**
4   * Application main class
5   * <p>
6   * Creates {@link is.mjuk.cache.Controller} and {@link is.mjuk.cache.View}
7   * objects and passes the Controller to the view.
8   * <p>
9   * Does not have any return values or in parameters.
10  */
11  public class CacheSimulator
12  {
13
14      /**
15      * Application main
16      * <p>
17      * Method is called by commandline
18      * Creates {@link is.mjuk.cache.Controller} and sends the controller as a
19      * paramter to the creation of a {@link is.mjuk.cache.View}.
20      */
21      public static void main(String[] args)
22      {
23          Controller c = new Controller();
24          View view = new View(c);
```

```
25        }
26
27  }
```

3.5.5 Class: InstructionDTO

```
 1  package is.mjuk.cache;
 2
 3  import java.lang.StringBuilder;
 4
 5  /**
 6   * Stores the result, address and type of a single instruction
 7   * <p>
 8   * InstructionDTO contains relevant data regarding a single execution of
 9   * a {@link is.mjuk.cache.Instruction}-object.
10   */
11  public class InstructionDTO {
12      private boolean hit;
13      private AddressDTO address;
14      private InstructionType type;
15
16      public InstructionDTO(boolean hit, AddressDTO address,
17          InstructionType type){
18          this.hit = hit;
19          this.address = address;
20          this.type = type;
21      }
22
23      /**
24       * Emty constructor
25       */
26      public InstructionDTO() {
27          // Intentionally left empty.
28      }
29
30      /**
31       * Get hit
32       */
33      public boolean getHit() {
34          return this.hit;
35      }
36
37      /**
38       * Set if it was a hit
39       *
40       * @param hit true if it was a hit
41       */
42      public void setHit(boolean hit) {
43          this.hit = hit;
44      }
45
46      /**
47       * Return this instructions address
48       */
49      public AddressDTO getAddress() {
50          return this.address;
51      }
52
53      /**
```

```
54          * Set this instructions address
55          *
56          * @param address {@link is.mjuk.cache.AddressDTO} for this instruction
57          */
58         public void setAddress(AddressDTO address) {
59             this.address = address;
60         }
61
62         /**
63          * Get the {@link is.mjuk.cache.InstructionType} of this instruction
64          */
65         public InstructionType getType() {
66             return type;
67         }
68
69         /**
70          * Set the {@link is.mjuk.cache.InstructionType} of this instryction
71          */
72         public void setType(InstructionType type) {
73             this.type = type;
74         }
75
76         /**
77          * Converts this DTO to a string
78          */
79         public String toString() {
80             StringBuilder rv = new StringBuilder();
81             if (this.hit) {
82                 rv.append("Hit ");
83             } else {
84                 rv.append("Miss ");
85             }
86
87             if (this.type == InstructionType.LOAD) {
88                 rv.append("load ");
89             } else if (this.type == InstructionType.STORE) {
90                 rv.append("store ");
91             }
92
93             rv.append("0x");
94             rv.append(Long.toString(this.address.getTag(), 16));
95             rv.append(" 0x");
96             rv.append(Long.toString(this.address.getIndex(), 16));
97             rv.append(" 0x");
98             rv.append(Long.toString(this.address.getOffset(), 16));
99             return rv.toString();
100        }
101    }
```

3.5.6 Enum: InstructionType

```
1  package is.mjuk.cache;
2
3  /**
4   * List of Cache Instructions
5   */
6  public enum InstructionType {
7      LOAD,
8      STORE
```

```
 9   }
```

3.5.7 Class: LayoutDTO

```
 1   package is.mjuk.cache;
 2
 3   /**
 4    * Object for transferring cache layout data.
 5    */
 6   public class LayoutDTO
 7   {
 8       private int blockSize;
 9       private int blockCount;
10       private int associativity;
11       private int tagSize;
12       private int indexSize;
13       private int offsetSize;
14
15       /**
16        * Getter for the blocksize property of the object
17        * @return The blocksize property of the object
18        */
19       public int getBlockSize() {
20           return this.blockSize;
21       }
22
23       /**
24        * Updates the blocksize property of the object
25        * @param blockSize New value for the blocksize of the object
26        */
27       public void setBlockSize(int blockSize) {
28           this.blockSize = blockSize;
29       }
30
31       /**
32        * Getter for the blockcount property of the object
33        * @return The blockcount property of the object
34        */
35       public int getBlockCount() {
36           return this.blockCount;
37       }
38
39       /**
40        * Updates the blockcount property of the object
41        * @param blockCount New value for the block count of the object
42        */
43       public void setBlockCount(int blockCount) {
44           this.blockCount = blockCount;
45       }
46
47       /**
48        * Returns the value of the stored associativity-property
49        * @return The associativity property of the object
50        */
51       public int getAssociativity() {
52           return this.associativity;
53       }
54
55       /**
```

```
56          * Updates the associativity property of the object
57          * @param associativity New value for associativity
58          */
59         public void setAssociativity(int associativity) {
60             this.associativity = associativity;
61         }
62
63         /**
64          * Getter for the tagsize property of the object
65          * @return Tagsize property of the object
66          */
67         public int getTagSize() {
68             return this.tagSize;
69         }
70
71         /**
72          * Updates the tagsize property of the object
73          * @param tagSize New value for the tag size of the object
74          */
75         public void setTagSize(int tagSize) {
76             this.tagSize = tagSize;
77         }
78
79         /**
80          * Getter for the index size of the object
81          * @return The index size property of the object
82          */
83         public int getIndexSize() {
84             return this.indexSize;
85         }
86
87         /**
88          * Updates the index size property of the object
89          * @param indexSize New value for the index size of the object
90          */
91         public void setIndexSize(int indexSize) {
92             this.indexSize = indexSize;
93         }
94
95         /**
96          * Getter for the offset size
97          * @return Offset size property of the object
98          */
99         public int getOffsetSize() {
100            return this.offsetSize;
101        }
102
103        /**
104         * Updates the offset size value of the object
105         * @param offsetSize New value for offset size property.
106         */
107        public void setOffsetSize(int offsetSize) {
108            this.offsetSize = offsetSize;
109        }
110
111 }
```

3.5.8 Class: SimulationDTO

```java
1  package is.mjuk.cache;
2  import java.util.Date;
3
4  /**
5  * Collection of data related to the simulation
6  */
7  public class SimulationDTO {
8
9      private double hitRate;
10     private int loads;
11     private int stores;
12     private String nickname;
13     private Date dateTime;
14
15     private LayoutDTO layoutDTO;
16
17     /**
18     * Constructs an empty DTO for simulation data.
19     */
20     public SimulationDTO(){
21     }
22
23     /**
24     * Constructs a DTO with certain data from initialization.
25     * @param hitRate Hitrate value for the new DTO
26     * @param nickname Nickname value for the new DTO
27     * @param dateTime Date and time for attaching the new DTO
28     * @param layoutDTO Layout DTO to store in the new DTO
29     */
30     public SimulationDTO(double hitRate, String nickname,
31         Date dateTime, LayoutDTO layoutDTO) {
32         this.hitRate = hitRate;
33         this.nickname = nickname;
34         this.dateTime = dateTime;
35         this.layoutDTO = layoutDTO;
36     }
37
38     /**
39     * Updates the hitrate value for the DTO
40     * @param hitRate New hitrate value for the DTO
41     */
42     public void setHitRate(double hitRate){
43         this.hitRate = hitRate;
44     }
45
46     /**
47     * Getter for the hitrate property
48     * @return Hitrate property of the DTO
49     */
50     public double getHitrate(){
51         return this.hitRate;
52     }
53
54     /**
55     * Calculates the hitrate from one and returns the value.
56     * @return 1 - {@link is.mjuk.cache.SimulationDTO#getHitrate()}
57     */
58     public double getMissrate(){
59         return 1-this.hitRate;
```

```
60          }
61
62          /**
63           * Extracts datetime and nickname properties from a user object and stores.
64           * @param user User to update nickname and datetime from.
65           */
66          public void setUser(User user){
67              this.nickname = user.getNickname();
68              this.dateTime = user.getDateTime();
69          }
70
71          /**
72           * Getter for the nickname property.
73           */
74          public String getNickname(){
75              return this.nickname;
76          }
77
78          /**
79           * Getter for the datetime property.
80           */
81          public Date getDateTime(){
82              return this.dateTime;
83          }
84
85          /**
86           * Setter for the stores property
87           */
88          public void setStores(int stores){
89              this.stores = stores;
90          }
91
92          /**
93           * Setter for the loads property
94           */
95          public void setLoads(int loads){
96              this.loads = loads;
97          }
98
99          /**
100          * Getter for the stores property
101          */
102         public int getStores(){
103             return this.stores;
104         }
105
106         /**
107          * Getter for the loads property
108          */
109         public int getLoads(){
110             return this.loads;
111         }
112
113         /**
114          * Sets the attached LayoutDTO.
115          * @param layoutDTO New {@link is.mjuk.cache.LayoutDTO} to store
116          */
117         public void setLayoutDTO(LayoutDTO layoutDTO){
118             this.layoutDTO = layoutDTO;
```

```
119        }
120
121        /**
122         * Getter for the stored layoutDTO.
123         */
124        public LayoutDTO getLayoutDTO(){
125            return this.layoutDTO;
126        }
127    }
```

3.5.9 Class: User

```
 1  package is.mjuk.cache;
 2
 3  import java.util.Date;
 4
 5  /**
 6   * Class for user-specific data.
 7   *
 8   * @author Emil Tullstedt
 9   */
10  public class User {
11      private Date datetime;
12      private String nickname;
13
14      public User() {
15          datetime = new Date();
16      }
17
18      /**
19       * Get currently stored datetime object as a string.
20       */
21      public Date getDateTime() {
22          return this.datetime;
23      }
24
25      /**
26       * Sets the user's datetime variable to whatever the date and time is when
27       * the function is called.
28       */
29      public void updateDateTime()
30      {
31          datetime = new Date();
32      }
33
34      /**
35       * Sets the user's datetime variable to unixtime defined in parameters
36       * @param unixtime Timestamp for a specific time to set the users date too.
37       */
38      public void updateDateTime(long unixtime) {
39          datetime = new Date(unixtime);
40      }
41
42      /**
43       * @param nickname - Sets new nickname for the user
44       */
45      public void setNickname(String nickname) {
46          this.nickname = nickname;
47      }
```

```
48
49      /**
50       * Gets the stored nickname
51       *
52       * @return Current value of nickname.
53       */
54      public String getNickname() {
55          return nickname;
56      }
57  }
```

3.6 Tests

3.6.1 New Class: AddressLayout Test

The `AddressLayoutTest`-class is testing if the `AddressLayout` class can be created, parse
a valid address and throw an exception on parsing an invalid address.

Test for the `AddressLayout`-class in 3.4.1.

```
1   package is.mjuk.cache;
2
3   import static org.junit.Assert.assertTrue;
4   import static org.junit.Assert.assertFalse;
5   import static org.junit.Assert.assertEquals;
6
7   import org.junit.AfterClass;
8   import org.junit.BeforeClass;
9   import org.junit.Test;
10
11  public class AddressLayoutTest {
12
13      @Test
14      public void creatingValidAddressLayout() {
15          CacheLayout cacheLayout = new CacheLayout(4, 4, 1);
16          LayoutDTO cacheData = cacheLayout.generateLayoutDTO();
17
18          assertEquals("Tag␣should␣be␣", 28, cacheData.getTagSize());
19          assertEquals("Index␣should␣be␣4", 2, cacheData.getIndexSize());
20          assertEquals("Offset␣should␣be␣1", 2, cacheData.getOffsetSize());
21      }
22
23      @Test
24      public void parseValidAddress() throws IllegalAddressException {
25          CacheLayout cacheLayout = new CacheLayout(16, 16, 1);
26          AddressLayout addressLayout = cacheLayout.getAddressLayout();
27
28          AddressDTO addressDTO = addressLayout.parseAddress(0xABAD1DEC);
29          assertEquals("Offset␣should␣be␣0xC␣", 0xC, addressDTO.getOffset());
30          assertEquals("Index␣should␣be␣0xE␣", 0xE, addressDTO.getIndex());
31          assertEquals("Tag␣should␣be␣0xABAD1D␣", 0xABAD1D,
32              addressDTO.getTag());
33      }
34
35      @Test(expected=IllegalAddressException.class)
36      public void parseInvalidAddressNonDivideableByFour()
37      throws IllegalAddressException {
38          CacheLayout cacheLayout = new CacheLayout(16, 16, 1);
39          AddressLayout addressLayout = cacheLayout.getAddressLayout();
40
```

```
41              AddressDTO addressDTO = addressLayout.parseAddress(0xABAD1DEA);
42          }
43
44  }
```

3.6.2 Class: CacheLayout Test

Updated to ensure to avoid `IllegalAddressException` (3.4.2).

Test for the `CacheLayout`-class in 3.5.3.

```
 1  package is.mjuk.cache;
 2
 3  import static org.junit.Assert.assertTrue;
 4  import static org.junit.Assert.assertFalse;
 5  import static org.junit.Assert.assertEquals;
 6
 7  import org.junit.AfterClass;
 8  import org.junit.BeforeClass;
 9  import org.junit.Test;
10  import java.lang.Math;
11
12  public class CacheLayoutTest {
13
14      @Test
15      public void creatingValidCacheLayout() {
16          CacheLayout cacheLayout = new CacheLayout(4, 4, 1);
17          LayoutDTO cacheData = cacheLayout.generateLayoutDTO();
18
19          assertEquals("BlockSize should be 4", cacheData.getBlockSize(), 4);
20          assertEquals("BlockCount should be 4", cacheData.getBlockCount(), 4);
21          assertEquals("Associativity should be 1",
22              cacheData.getAssociativity(), 1);
23      }
24
25      @Test
26      public void creatingDataCache() {
27          CacheLayout cacheLayout = new CacheLayout(4, 4, 1);
28          DataCache dataCache = cacheLayout.getDataCache();
29
30          assertEquals("There should be one set", dataCache.getNumberOfSets(), 1);
31          assertEquals("There should be four blocks",
32              dataCache.getNumberOfBlocks(), 4);
33      }
34
35      @Test(expected = IllegalArgumentException.class)
36      public void createInvalidCacheLayout() {
37          // BlockSize and BlockCount must be powers of two.
38          CacheLayout cacheLayout = new CacheLayout(3, 4, 1);
39          cacheLayout.getAddressLayout();
40      }
41
42      @Test(expected = IllegalArgumentException.class)
43      public void tooBigAddress() {
44          // Will result in an address of size 34 bits, which exceeds
45          // maximum 32.
46          CacheLayout cacheLayout = new CacheLayout((int) Math.pow(2, 30),
47              (int) Math.pow(2, 4), 1);
48          cacheLayout.getAddressLayout();
```

```
49          }
50    }
```

3.6.3 Class: DataCache Test

Updated to ensure to avoid `IllegalAddressException` (3.4.2).
   Test for the `DataCache`-class in 3.6.3.

```
 1    package is.mjuk.cache;
 2
 3    import static org.junit.Assert.assertTrue;
 4    import static org.junit.Assert.assertFalse;
 5    import static org.junit.Assert.assertEquals;
 6
 7    import org.junit.AfterClass;
 8    import org.junit.BeforeClass;
 9    import org.junit.Test;
10
11    public class DataCacheTest {
12
13        @Test
14        public void constructCache() {
15            CacheLayout cacheLayout = new CacheLayout(4, 4, 1);
16            DataCache dataCache = cacheLayout.getDataCache();
17
18            assertEquals("There should be one set", dataCache.getNumberOfSets(), 1);
19            assertEquals("There should be four blocks",
20                dataCache.getNumberOfBlocks(), 4);
21            assertEquals("There shouldn't be any hits", dataCache.getHits(), 0);
22            assertEquals("There shouldn't be any misses", dataCache.getMisses(), 0);
23            assertEquals("There shouldn't be any stores", dataCache.getStores(), 0);
24            assertEquals("There shouldn't be any loads", dataCache.getLoads(), 0);
25        }
26
27        @Test
28        public void loadData() throws IllegalAddressException {
29            CacheLayout cacheLayout = new CacheLayout(16, 16, 1);
30            DataCache dataCache = cacheLayout.getDataCache();
31            AddressLayout addressLayout = cacheLayout.getAddressLayout();
32
33            assertFalse("1) Miss",
34                dataCache.loadData(addressLayout.parseAddress(0xABAD1DE0)));
35
36            assertTrue("2) Hit",
37                dataCache.loadData(addressLayout.parseAddress(0xABAD1DE0)));
38
39            assertFalse("3) Miss",
40                dataCache.loadData(addressLayout.parseAddress(0xADA5F000)));
41
42            assertFalse("4) Miss",
43                dataCache.loadData(addressLayout.parseAddress(0xAAAABBE0)));
44
45            assertFalse("5) Miss (unloaded)",
46                dataCache.loadData(addressLayout.parseAddress(0xABAD1DE0)));
47
48            assertTrue("6) Hit",
49                dataCache.loadData(addressLayout.parseAddress(0xADA5F000)));
50
```

```
51          assertEquals("There should be 6 loads", 6, dataCache.getLoads());
52          assertEquals("There should be 0 stores", 0, dataCache.getStores());
53          assertEquals("There should be 4 misses", 4, dataCache.getMisses());
54          assertEquals("There should be 2 hits", 2, dataCache.getHits());
55      }
56
57      @Test
58      public void storeData() throws IllegalAddressException {
59          CacheLayout cacheLayout = new CacheLayout(16, 16, 1);
60          DataCache dataCache = cacheLayout.getDataCache();
61          AddressLayout addressLayout = cacheLayout.getAddressLayout();
62
63          assertFalse("1) Miss",
64              dataCache.storeData(addressLayout.parseAddress(0xABAD1DE0)));
65
66          assertTrue("2) Hit",
67              dataCache.storeData(addressLayout.parseAddress(0xABAD1DE0)));
68
69          assertFalse("3) Miss",
70              dataCache.storeData(addressLayout.parseAddress(0xADA5F000)));
71
72          assertFalse("4) Miss",
73              dataCache.storeData(addressLayout.parseAddress(0xAAAABBE0)));
74
75          assertFalse("5) Miss (unloaded)",
76              dataCache.storeData(addressLayout.parseAddress(0xABAD1DE0)));
77
78          assertTrue("6) Hit",
79              dataCache.storeData(addressLayout.parseAddress(0xADA5F000)));
80
81          assertEquals("There should be 0 loads", 0, dataCache.getLoads());
82          assertEquals("There should be 6 stores", 6 , dataCache.getStores());
83          assertEquals("There should be 4 misses", 4, dataCache.getMisses());
84          assertEquals("There should be 2 hits", 2, dataCache.getHits());
85      }
86
87      @Test
88      public void hitRate() throws IllegalAddressException {
89          CacheLayout cacheLayout = new CacheLayout(16, 16, 1);
90          DataCache dataCache = cacheLayout.getDataCache();
91          AddressLayout addressLayout = cacheLayout.getAddressLayout();
92
93          assertEquals("0 Hitrate should begin at 0.00",
94              0.00, dataCache.getHitrate(), 0.00);
95
96          dataCache.loadData(addressLayout.parseAddress(0xABAD1DE0));
97
98          assertEquals("1 Hitrate should be 0.00",
99              0.00, dataCache.getHitrate(), 0.00);
100
101          dataCache.loadData(addressLayout.parseAddress(0xABAD1DE0));
102
103          assertEquals("2 Hitrate should be 0.50",
104              0.50, dataCache.getHitrate(), 0.00);
105
106          dataCache.storeData(addressLayout.parseAddress(0xABAD1DE0));
107
108          assertEquals("3 Hitrate should be 0.66",
109              0.66, dataCache.getHitrate(), 0.01);
```

```
110
111        dataCache.storeData(addressLayout.parseAddress(0xABAD1DC0));
112
113        assertEquals("4␣Hitrate␣should␣be␣0.50",
114            0.5, dataCache.getHitrate(), 0.0);
115
116        dataCache.storeData(addressLayout.parseAddress(0xAAAABBE0));
117
118        assertEquals("5␣Hitrate␣should␣be␣0.40",
119            0.4, dataCache.getHitrate(), 0.01);
120
121        dataCache.loadData(addressLayout.parseAddress(0xABAD1DE0));
122
123        assertEquals("6␣Hitrate␣should␣be␣0.33",
124            0.33, dataCache.getHitrate(), 0.01);
125    }
126
127 }
```

3.6.4 Class: Storage Test

Not updated since Assignment 3
Test for the Storage-class in 3.6.4.

```
1  package is.mjuk.cache;
2
3  import static org.junit.Assert.assertTrue;
4  import static org.junit.Assert.assertFalse;
5  import static org.junit.Assert.assertEquals;
6
7  import org.junit.AfterClass;
8  import org.junit.BeforeClass;
9  import org.junit.Test;
10
11 public class StorageTest {
12     Storage store = Storage.getStorage();
13
14     @Test
15     public void addInstructionDTO() throws IllegalAddressException {
16         store.clean();
17
18         CacheLayout cacheLayout = new CacheLayout(16, 16, 1);
19         DataCache dataCache = cacheLayout.getDataCache();
20         AddressLayout addressLayout = cacheLayout.getAddressLayout();
21
22         Instruction instruction = new Instruction(dataCache, addressLayout,
23             InstructionType.LOAD, 0xABAD1DEC);
24         InstructionDTO instructionDTO = instruction.executeInstruction();
25         store.addInstructionDTO(instructionDTO);
26         assertFalse(store.getInstructionDTO(0).getHit());
27
28         instruction = new Instruction(dataCache, addressLayout,
29             InstructionType.LOAD, 0xABAD1DEC);
30         instructionDTO = instruction.executeInstruction();
31         store.addInstructionDTO(instructionDTO);
32         assertTrue(store.getInstructionDTO(1).getHit());
33
34         store.clean();
```

```
35          }
36
37      @Test
38      public void clean() throws IllegalAddressException {
39          store.clean();
40
41          CacheLayout cacheLayout = new CacheLayout(16, 16, 1);
42          DataCache dataCache = cacheLayout.getDataCache();
43          AddressLayout addressLayout = cacheLayout.getAddressLayout();
44
45          Instruction instruction = new Instruction(dataCache, addressLayout,
46              InstructionType.LOAD, 0xABAD1DEC);
47          InstructionDTO instructionDTO = instruction.executeInstruction();
48          store.addInstructionDTO(instructionDTO);
49          assertEquals("The type of the instruction saved should be LOAD",
50              InstructionType.LOAD, store.getInstructionDTO(0).getType());
51
52          store.clean();
53
54          instruction = new Instruction(dataCache, addressLayout,
55              InstructionType.STORE, 0xABAD1DEC);
56          instructionDTO = instruction.executeInstruction();
57          store.addInstructionDTO(instructionDTO);
58          assertEquals("The type of the instruction saved should be STORE",
59              InstructionType.STORE, store.getInstructionDTO(0).getType());
60
61          store.clean();
62      }
63
64      @Test
65      public void storeLayout() {
66          CacheLayout cacheLayout = new CacheLayout(8, 4, 2);
67
68          store.storeLayoutDTO(cacheLayout.generateLayoutDTO());
69
70          LayoutDTO layoutDTO = store.getLayoutDTO();
71
72          assertEquals("BlockSize should be 8", 8, layoutDTO.getBlockSize());
73          assertEquals("Index should be 2", 2, layoutDTO.getIndexSize());
74      }
75
76      @Test
77      public void storeDateTime() {
78          User u = new User();
79
80          store.storeDateTime(u.getDateTime());
81          assertEquals("Date in Storage should be same as date from user",
82              u.getDateTime(), store.getDateTime());
83
84          u.updateDateTime(1234567890000L);
85          store.storeDateTime(u.getDateTime());
86          assertEquals("Date in storage should be same as date from user",
87              u.getDateTime(), store.getDateTime());
88      }
89
90      @Test
91      public void storeHitrate() throws IllegalAddressException {
92          CacheLayout cacheLayout = new CacheLayout(16, 16, 1);
93          DataCache dataCache = cacheLayout.getDataCache();
```

```
94              AddressLayout addressLayout = cacheLayout.getAddressLayout();
95
96              store.storeHitrate(dataCache.getHitrate());
97              assertEquals("Stored␣hitrate␣should␣be␣same␣as␣in␣the␣datacache",
98                  dataCache.getHitrate(), store.getHitrate(), 0.01);
99
100             dataCache.loadData(addressLayout.parseAddress(0xABAD1DE0));
101             dataCache.loadData(addressLayout.parseAddress(0xABAD1DE0));
102             dataCache.storeData(addressLayout.parseAddress(0xABAD1DE0));
103             dataCache.storeData(addressLayout.parseAddress(0xABAD1DCC));
104             dataCache.storeData(addressLayout.parseAddress(0xAAAABBEC));
105             dataCache.loadData(addressLayout.parseAddress(0xABAD1DE0));
106
107             store.storeHitrate(dataCache.getHitrate());
108             assertEquals("Stored␣hitrate␣should␣be␣same␣as␣in␣the␣datacache",
109                 dataCache.getHitrate(), store.getHitrate(), 0.01);
110         }
111
112     @Test
113     public void storeNickname() {
114         User u = new User();
115         u.setNickname("Huxley");
116
117         store.storeNickname(u.getNickname());
118         assertEquals("Name␣in␣storage␣should␣be␣same␣as␣name␣in␣user␣object",
119             u.getNickname(), store.getNickname());
120     }
121
122 }
```

3.7 Sample Run

```
1  ASKS USER TO ENTER USER PROPERTIES
2  --------------------------------
3
4  Please enter nickname:
5  Emil
6  Your nickname is: Emil
7
8  USER SPECIFIES BLOCK PROPERTIES
9  -----------------------------
10
11 Enter block size in bytes:
12 4
13 Enter block count:
14 8
15 Enter associativity:
16 2
17 CALCULATES CACHE LAYOUT & CREATES CACHE
18 Displaying Cache Data
19 Index 0x0: invalid 0x0 invalid 0x0
20 Index 0x1: invalid 0x0 invalid 0x0
21 Index 0x2: invalid 0x0 invalid 0x0
22 Index 0x3: invalid 0x0 invalid 0x0
23 Index 0x4: invalid 0x0 invalid 0x0
24 Index 0x5: invalid 0x0 invalid 0x0
25 Index 0x6: invalid 0x0 invalid 0x0
```

```
26  Index 0x7: invalid 0x0 invalid 0x0
27
28  USER INPUTS INSTRUCTIONS
29  Write 'exit' to stop the application
30  To use instruction load, write 'load_<memaddress>'
31  To use instruction store, write 'store_<memaddress>'
32  To observe changes in the cache, write 'observe'
33  To stop observing changes in the cache, write 'deobserve'
34  load 0x10
35  Miss load 0x0 0x4 0x0
36  Hitrate is 0.00 and missrate is 1.00.
37  store 0x11
38  Error parsing memory address: is.mjuk.cache.IllegalAddressException: Memory address
        0x11 is not divisible by four and do not point at a valid block address.
39  load 0x4
40  Miss load 0x0 0x1 0x0
41  Hitrate is 0.00 and missrate is 1.00.
42  command_that_doesnt_exist
43  Instruction not found 'command_that_doesnt_exist'
44  observe
45  Now observing the DataCache for changes
46  l 0x100
47  Index 0x0: invalid 0x0 valid 0x8
48  Index 0x1: invalid 0x0 valid 0x0
49  Index 0x2: invalid 0x0 invalid 0x0
50  Index 0x3: invalid 0x0 invalid 0x0
51  Index 0x4: invalid 0x0 valid 0x0
52  Index 0x5: invalid 0x0 invalid 0x0
53  Index 0x6: invalid 0x0 invalid 0x0
54  Index 0x7: invalid 0x0 invalid 0x0
55
56  Miss load 0x8 0x0 0x0
57  Hitrate is 0.00 and missrate is 1.00.
58  store 0x300
59  Index 0x0: valid 0x18 valid 0x8
60  Index 0x1: invalid 0x0 valid 0x0
61  Index 0x2: invalid 0x0 invalid 0x0
62  Index 0x3: invalid 0x0 invalid 0x0
63  Index 0x4: invalid 0x0 valid 0x0
64  Index 0x5: invalid 0x0 invalid 0x0
65  Index 0x6: invalid 0x0 invalid 0x0
66  Index 0x7: invalid 0x0 invalid 0x0
67
68  Miss store 0x18 0x0 0x0
69  Hitrate is 0.00 and missrate is 1.00.
70  load 0x100
71  Hit load 0x8 0x0 0x0
72  Hitrate is 0.20 and missrate is 0.80.
73  store 0x500
74  Index 0x0: valid 0x28 valid 0x8
75  Index 0x1: invalid 0x0 valid 0x0
76  Index 0x2: invalid 0x0 invalid 0x0
77  Index 0x3: invalid 0x0 invalid 0x0
78  Index 0x4: invalid 0x0 valid 0x0
79  Index 0x5: invalid 0x0 invalid 0x0
80  Index 0x6: invalid 0x0 invalid 0x0
81  Index 0x7: invalid 0x0 invalid 0x0
82
83  Miss store 0x28 0x0 0x0
```

```
 84  Hitrate is 0.17 and missrate is 0.83.
 85  store 0x300
 86  Index 0x0: valid 0x18 valid 0x8
 87  Index 0x1: invalid 0x0 valid 0x0
 88  Index 0x2: invalid 0x0 invalid 0x0
 89  Index 0x3: invalid 0x0 invalid 0x0
 90  Index 0x4: invalid 0x0 valid 0x0
 91  Index 0x5: invalid 0x0 invalid 0x0
 92  Index 0x6: invalid 0x0 invalid 0x0
 93  Index 0x7: invalid 0x0 invalid 0x0
 94
 95  Miss store 0x18 0x0 0x0
 96  Hitrate is 0.14 and missrate is 0.86.
 97  load 0x45C
 98  Index 0x0: valid 0x18 valid 0x8
 99  Index 0x1: invalid 0x0 valid 0x0
100  Index 0x2: invalid 0x0 invalid 0x0
101  Index 0x3: invalid 0x0 invalid 0x0
102  Index 0x4: invalid 0x0 valid 0x0
103  Index 0x5: invalid 0x0 invalid 0x0
104  Index 0x6: invalid 0x0 invalid 0x0
105  Index 0x7: invalid 0x0 valid 0x22
106
107  Miss load 0x22 0x7 0x0
108  Hitrate is 0.13 and missrate is 0.88.
109  load 0xABAD1DEA
110  Error parsing memory address: is.mjuk.cache.IllegalAddressException: Memory address
         0xabad1dea is not divisible by four and do not point at a valid block address.
111  store 0xABAD1DEC
112  Index 0x0: valid 0x18 valid 0x8
113  Index 0x1: invalid 0x0 valid 0x0
114  Index 0x2: invalid 0x0 invalid 0x0
115  Index 0x3: invalid 0x0 valid 0x55d68ef
116  Index 0x4: invalid 0x0 valid 0x0
117  Index 0x5: invalid 0x0 invalid 0x0
118  Index 0x6: invalid 0x0 invalid 0x0
119  Index 0x7: invalid 0x0 valid 0x22
120
121  Miss store 0x55d68ef 0x3 0x0
122  Hitrate is 0.11 and missrate is 0.89.
123  exit
124  Index 0x0: valid 0x18 valid 0x8
125  Index 0x1: invalid 0x0 valid 0x0
126  Index 0x2: invalid 0x0 invalid 0x0
127  Index 0x3: invalid 0x0 valid 0x55d68ef
128  Index 0x4: invalid 0x0 valid 0x0
129  Index 0x5: invalid 0x0 invalid 0x0
130  Index 0x6: invalid 0x0 invalid 0x0
131  Index 0x7: invalid 0x0 valid 0x22
132
133  Simulation data:
134  Username: Emil
135  Load instructions: 5
136  Store instructions: 4
137  Hit rate: 0.1111111111111111
138  Miss rate: 0.8888888888888888
139  Block Size: 4 bytes
140  Block Count: 8 blocks
141  Associativity: 2
```

```
142  Address tag size: 27 bits
143  Address index size: 3 bits
144  Address offset size: 2 bits
```

3.8 Patches

These patches are the unchanged initial diffs for the git commit that introduced the running code. These do not necessarily represent the final state, which is accessible by reading the source files in it's entirety. Notably, the Javadoc-comments are not complete.

3.8.1 Storage Patch

```
1   diff --git a/source/is/mjuk/cache/Controller.java b/source/is/mjuk/cache/Controller.java
2   index d31dc02..6aed3d5 100644
3   --- a/source/is/mjuk/cache/Controller.java
4   +++ b/source/is/mjuk/cache/Controller.java
5   @@ -6,6 +6,7 @@ public class Controller
6        private CacheLayout cacheLayout;
7        private DataCache dataCache;
8        private AddressLayout addressLayout;
9   +    private Storage store = Storage.getStorage();
10
11       public Controller() {
12           user = new User();
13   @@ -29,7 +30,11 @@ public class Controller
14               return rv;
15           }
16
17   -       return instruction.executeInstruction();
18   +       InstructionDTO instructionDTO = instruction.executeInstruction();
19   +
20   +       store.addInstructionDTO(instructionDTO);
21   +
22   +       return instructionDTO;
23       };
24
25       /**
26   @@ -65,6 +70,7 @@ public class Controller
27           cacheLayout = new CacheLayout(blockSize, blockCount, associativity);
28           addressLayout = cacheLayout.getAddressLayout();
29           dataCache = cacheLayout.getDataCache();
30   +       store.storeLayoutDTO(this.cacheLayout.generateLayoutDTO());
31       }
32
33       public String displayCache() {
34   @@ -82,12 +88,14 @@ public class Controller
35       }
36
37       public SimulationDTO getSimulationData(){
38   -       SimulationDTO simDTO = new SimulationDTO();
39   -       simDTO.setUser(this.user);
40   -       simDTO.setHitRate(this.dataCache.getHitrate());
41   +       store.storeHitrate(this.dataCache.getHitrate());
42   +       store.storeNickname(this.user.getNickname());
43   +       store.storeDateTime(this.user.getDateTime());
44   +       store.storeLayoutDTO(this.cacheLayout.generateLayoutDTO());
45   +
```

```
46  +          SimulationDTO simDTO = store.createDTO();
47  +          simDTO.setStores(this.dataCache.getStores());
48  +          simDTO.setLoads(this.dataCache.getLoads());
49  -          simDTO.setLayoutDTO(this.cacheLayout.generateLayoutDTO());
50  +          return simDTO;
51  +      }
52   }
53  diff --git a/source/is/mjuk/cache/Storage.java b/source/is/mjuk/cache/Storage.java
54  new file mode 100644
55  index 0000000..e5002be
56  --- /dev/null
57  +++ b/source/is/mjuk/cache/Storage.java
58  @@ -0,0 +1,131 @@
59  +package is.mjuk.cache;
60  +
61  +import java.util.ArrayList;
62  +import java.util.Date;
63  +
64  +/**
65  + * Templete for storage of data
66  + * <p>
67  + * Stores different variables for logging and
68  + * storage to database or likewise.
69  + * @author  Emil Tullstedt <emiltu@kth.se>
70  + */
71  +public class Storage {
72  +    private static final Storage storage = new Storage();
73  +    private ArrayList<InstructionDTO> instructionStore;
74  +    private LayoutDTO layoutStore;
75  +    private String nickname;
76  +    private Date datetime;
77  +    private double hitrate;
78  +
79  +    private Storage() {
80  +        this.instructionStore = new ArrayList<InstructionDTO>();
81  +    }
82  +
83  +    /**
84  +     * Gets the saved storage
85  +     */
86  +    public static Storage getStorage() {
87  +        return storage;
88  +    }
89  +
90  +    /**
91  +     * Adds an {@link is.mjuk.cache.InstructionDTO} to the
92  +     * list of instructions.
93  +     *
94  +     * @param instruction {@link is.mjuk.cache.InstructionDTO} to be stored.
95  +     */
96  +    public void addInstructionDTO (InstructionDTO instruction) {
97  +        this.instructionStore.add(instruction);
98  +    }
99  +
100 +    /**
101 +     * Get one {@link is.mjuk.cache.InstructionDTO} from the
102 +     * list of saved InstructionDTOs.
103 +     *
104 +     * @param count The index of the DTO to be retrived
```

```
105  +      */
106  +      public InstructionDTO getInstructionDTO (int count) {
107  +          return instructionStore.get(count);
108  +      }
109  +
110  +      /**
111  +       * Sets the {@link is.mjuk.cache.LayoutDTO} of the data
112  +       *
113  +       * @param layout {@link is.mjuk.cache.LayoutDTO} to be stored.
114  +       */
115  +      public void storeLayoutDTO(LayoutDTO layout) {
116  +          this.layoutStore = layout;
117  +      }
118  +
119  +      /**
120  +       * Get the saved {@link is.mjuk.cache.LayoutDTO}
121  +       */
122  +      public LayoutDTO getLayoutDTO() {
123  +          return this.layoutStore;
124  +      }
125  +
126  +      /**
127  +       * Set the date to store
128  +       *
129  +       * @param satetime The date to set to
130  +       */
131  +      public void storeDateTime(Date datetime) {
132  +          this.datetime = datetime;
133  +      }
134  +
135  +      /**
136  +       * Get the stored Date
137  +       */
138  +      public Date getDateTime() {
139  +          return this.datetime;
140  +      }
141  +
142  +      /**
143  +       * Set hitrate to store
144  +       *
145  +       * @param hitrate The hitrate to store
146  +       */
147  +      public void storeHitrate(double hitrate) {
148  +          this.hitrate = hitrate;
149  +      }
150  +
151  +      /**
152  +       * Get the stored date
153  +       */
154  +      public double getHitrate() {
155  +          return this.hitrate;
156  +      }
157  +
158  +      /**
159  +       * Set nickname to store
160  +       */
161  +      public void storeNickname(String nickname) {
162  +          this.nickname = nickname;
163  +      }
```

46(56)

```
164  +
165  +      /**
166  +       * Get the stored nickname
167  +       */
168  +      public String getNickname() {
169  +          return this.nickname;
170  +      }
171  +
172  +      /**
173  +       * Creates a {@link is.mjuk.cache.SimulationDTO} from all saved data
174  +       */
175  +      public SimulationDTO createDTO() {
176  +          return new SimulationDTO (hitrate, nickname, datetime, layoutStore);
177  +      }
178  +
179  +      /**
180  +       * Reset all variables to emty instances
181  +       */
182  +      public void clean() {
183  +          instructionStore = new ArrayList<InstructionDTO>();
184  +          layoutStore = new LayoutDTO();
185  +          datetime = new Date();
186  +          nickname = new String();
187  +          hitrate = Double.NaN;
188  +      }
189  +}
190  diff --git a/test/is/mjuk/cache/StorageTest.java b/test/is/mjuk/cache/StorageTest.java
191  new file mode 100644
192  index 0000000..e71b023
193  --- /dev/null
194  +++ b/test/is/mjuk/cache/StorageTest.java
195  @@ -0,0 +1,122 @@
196  +package is.mjuk.cache;
197  +
198  +import static org.junit.Assert.assertTrue;
199  +import static org.junit.Assert.assertFalse;
200  +import static org.junit.Assert.assertEquals;
201  +
202  +import org.junit.AfterClass;
203  +import org.junit.BeforeClass;
204  +import org.junit.Test;
205  +
206  +public class StorageTest {
207  +      Storage store = Storage.getStorage();
208  +
209  +      @Test
210  +      public void addInstructionDTO() {
211  +          store.clean();
212  +
213  +          CacheLayout cacheLayout = new CacheLayout(16, 16, 1);
214  +          DataCache dataCache = cacheLayout.getDataCache();
215  +          AddressLayout addressLayout = cacheLayout.getAddressLayout();
216  +
217  +          Instruction instruction = new Instruction(dataCache, addressLayout,
218  +              InstructionType.LOAD, 0xABAD1DEA);
219  +          InstructionDTO instructionDTO = instruction.executeInstruction();
220  +          store.addInstructionDTO(instructionDTO);
221  +          assertFalse(store.getInstructionDTO(0).getHit());
222  +
```

```
223  +          instruction = new Instruction(dataCache, addressLayout,
224  +              InstructionType.LOAD, 0xABAD1DEA);
225  +          instructionDTO = instruction.executeInstruction();
226  +          store.addInstructionDTO(instructionDTO);
227  +          assertTrue(store.getInstructionDTO(1).getHit());
228  +
229  +          store.clean();
230  +      }
231  +
232  +      @Test
233  +      public void clean() {
234  +          store.clean();
235  +
236  +          CacheLayout cacheLayout = new CacheLayout(16, 16, 1);
237  +          DataCache dataCache = cacheLayout.getDataCache();
238  +          AddressLayout addressLayout = cacheLayout.getAddressLayout();
239  +
240  +          Instruction instruction = new Instruction(dataCache, addressLayout,
241  +              InstructionType.LOAD, 0xABAD1DEA);
242  +          InstructionDTO instructionDTO = instruction.executeInstruction();
243  +          store.addInstructionDTO(instructionDTO);
244  +          assertEquals("The_type_of_the_instruction_saved_should_be_LOAD",
245  +              InstructionType.LOAD, store.getInstructionDTO(0).getType());
246  +
247  +          store.clean();
248  +
249  +          instruction = new Instruction(dataCache, addressLayout,
250  +              InstructionType.STORE, 0xABAD1DEA);
251  +          instructionDTO = instruction.executeInstruction();
252  +          store.addInstructionDTO(instructionDTO);
253  +          assertEquals("The_type_of_the_instruction_saved_should_be_STORE",
254  +              InstructionType.STORE, store.getInstructionDTO(0).getType());
255  +
256  +          store.clean();
257  +      }
258  +
259  +      @Test
260  +      public void storeLayout() {
261  +          CacheLayout cacheLayout = new CacheLayout(8, 4, 2);
262  +
263  +          store.storeLayoutDTO(cacheLayout.generateLayoutDTO());
264  +
265  +          LayoutDTO layoutDTO = store.getLayoutDTO();
266  +
267  +          assertEquals("BlockSize_should_be_8", 8, layoutDTO.getBlockSize());
268  +          assertEquals("Index_should_be_2", 2, layoutDTO.getIndexSize());
269  +      }
270  +
271  +      @Test
272  +      public void storeDateTime() {
273  +          User u = new User();
274  +
275  +          store.storeDateTime(u.getDateTime());
276  +          assertEquals("Date_in_Storage_should_be_same_as_date_from_user",
277  +              u.getDateTime(), store.getDateTime());
278  +
279  +          u.updateDateTime(1234567890000L);
280  +          store.storeDateTime(u.getDateTime());
281  +          assertEquals("Date_in_storage_should_be_same_as_date_from_user",
```

```
282  +              u.getDateTime(), store.getDateTime());
283  +      }
284  +
285  +      @Test
286  +      public void storeHitrate() {
287  +          CacheLayout cacheLayout = new CacheLayout(16, 16, 1);
288  +          DataCache dataCache = cacheLayout.getDataCache();
289  +          AddressLayout addressLayout = cacheLayout.getAddressLayout();
290  +
291  +          store.storeHitrate(dataCache.getHitrate());
292  +          assertEquals("Stored hitrate should be same as in the datacache",
293  +              dataCache.getHitrate(), store.getHitrate(), 0.01);
294  +
295  +          dataCache.loadData(addressLayout.parseAddress(0xABAD1DEA));
296  +          dataCache.loadData(addressLayout.parseAddress(0xABAD1DEA));
297  +          dataCache.storeData(addressLayout.parseAddress(0xABAD1DEA));
298  +          dataCache.storeData(addressLayout.parseAddress(0xABAD1DCA));
299  +          dataCache.storeData(addressLayout.parseAddress(0xAAAABBEA));
300  +          dataCache.loadData(addressLayout.parseAddress(0xABAD1DEA));
301  +
302  +          store.storeHitrate(dataCache.getHitrate());
303  +          assertEquals("Stored hitrate should be same as in the datacache",
304  +              dataCache.getHitrate(), store.getHitrate(), 0.01);
305  +      }
306  +
307  +      @Test
308  +      public void storeNickname() {
309  +          User u = new User();
310  +          u.setNickname("Huxley");
311  +
312  +          store.storeNickname(u.getNickname());
313  +          assertEquals("Name in storage should be same as name in user object",
314  +              u.getNickname(), store.getNickname());
315  +      }
316  +
317  +}
318  \ No newline at end of file
319  diff --git a/test/is/mjuk/cache/TestMain.java b/test/is/mjuk/cache/TestMain.java
320  index 6b3804f..410b21e 100644
321  --- a/test/is/mjuk/cache/TestMain.java
322  +++ b/test/is/mjuk/cache/TestMain.java
323  @@ -8,7 +8,8 @@ import org.junit.runners.Suite.SuiteClasses;
324   @RunWith(Suite.class)
325   @SuiteClasses({
326       CacheLayoutTest.class,
327  -    DataCacheTest.class
328  +    DataCacheTest.class,
329  +    StorageTest.class
330   })
331   public class TestMain {
332       // Intentionally left empty
```

3.8.2 Exception Patch

```
1  diff --git a/source/is/mjuk/cache/AddressLayout.java b/source/is/mjuk/cache/AddressLayout.java
2  index 1866c9f..8d46beb 100644
3  --- a/source/is/mjuk/cache/AddressLayout.java
4  +++ b/source/is/mjuk/cache/AddressLayout.java
5  @@ -28,13 +28,21 @@ public class AddressLayout {
```

49(56)

```
 6          * @return An {@link is.mjuk.cache.AddressDTO} containing the
 7          * tag, index and offset of the input address
 8          */
 9   -      public AddressDTO parseAddress(long address)
10   +      public AddressDTO parseAddress (long address) throws IllegalAddressException
11   +      {
12   +          if ((0b11 & address) != 0x0) {
13   +              String error = "Memory address 0x" + Long.toString(address, 16)
14   +                  + " is not divisible by four and do not point at a valid"
15   +                  + " block address.";
16   +              throw new IllegalAddressException(error);
17   +          }
18   +
19              AddressDTO rv = new AddressDTO();
20              rv.setOffset(MisMath.intToUnary(this.offsetSize) & address);
21   -          rv.setIndex(MisMath.intToUnary(this.indexSize) & address >>> offsetSize);
22   +          rv.setIndex(MisMath.intToUnary(this.indexSize) & address
23   +              >>> offsetSize);
24              rv.setTag(MisMath.intToUnary(this.tagSize) & address
25   -              >>> offsetSize + indexSize);
26   +              >>> offsetSize + indexSize);
27              return rv;
28          }
29
30   diff --git a/source/is/mjuk/cache/CacheLayout.java b/source/is/mjuk/cache/CacheLayout.java
31   index aa3cc05..c8d1779 100644
32   --- a/source/is/mjuk/cache/CacheLayout.java
33   +++ b/source/is/mjuk/cache/CacheLayout.java
34   @@ -30,7 +30,6 @@ public class CacheLayout
35          * @param blockSize Amount of <i>bytes</i> in a single block
36          * @param blockCount Amount of blocks in the cache
37          * @param associativity Associativity of the cache
38   -      * @throws java.lang.IllegalArgumentException
39          */
40          public CacheLayout(int blockSize, int blockCount, int associativity) {
41              this.blockSize = blockSize;
42   @@ -66,6 +65,7 @@ public class CacheLayout
43          * already.
44          *
45          * @return {@link is.mjuk.cache.AddressLayout}
46   +      * @throws TODO
47          */
48          public AddressLayout getAddressLayout() {
49              if (this.addressLayout == null) {
50   diff --git a/source/is/mjuk/cache/Controller.java b/source/is/mjuk/cache/Controller.java
51   index 4176e89..21474a6 100644
52   --- a/source/is/mjuk/cache/Controller.java
53   +++ b/source/is/mjuk/cache/Controller.java
54   @@ -45,7 +45,8 @@ public class Controller
55          * the instruction.
56          * @see is.mjuk.cache.Instruction
57          */
58   -      public InstructionDTO executeInstruction(String type, long address) {
59   +      public InstructionDTO executeInstruction(String type, long address)
60   +      throws IllegalAddressException {
61              Instruction instruction;
62
63              if(type.equals("load")) {
64   diff --git a/source/is/mjuk/cache/IllegalAddressException.java b/source/is/mjuk/cache/IllegalAddr
```

```
65   new file mode 100644
66   index 0000000..0247e66
67   --- /dev/null
68   +++ b/source/is/mjuk/cache/IllegalAddressException.java
69   @@ -0,0 +1,23 @@
70   +package is.mjuk.cache;
71   +
72   +/**
73   +* IllegalAddressException is a throwable exception for
74   +*/
75   +public class IllegalAddressException extends Exception {
76   +        public IllegalAddressException() {
77   +                super();
78   +        }
79   +
80   +
81   +        public IllegalAddressException(String errormsg) {
82   +                super(errormsg);
83   +        }
84   +
85   +        public IllegalAddressException(Throwable throwable) {
86   +                super(throwable);
87   +        }
88   +
89   +        public IllegalAddressException(String errormsg, Throwable throwable) {
90   +                super(errormsg, throwable);
91   +        }
92   +}
93   \ No newline at end of file
94   diff --git a/source/is/mjuk/cache/Instruction.java b/source/is/mjuk/cache/Instruction.java
95   index 85e180e..714120b 100644
96   --- a/source/is/mjuk/cache/Instruction.java
97   +++ b/source/is/mjuk/cache/Instruction.java
98   @@ -24,7 +24,8 @@ public class Instruction {
99        * @param address Address of the destinated memory block.
100       */
101       public Instruction(DataCache dataCache, AddressLayout addressLayout,
102   -        InstructionType type, long address) {
103   +        InstructionType type, long address)
104   +    throws IllegalAddressException {
105           this.type = type;
106
107           this.address = addressLayout.parseAddress(address);
108   diff --git a/source/is/mjuk/cache/View.java b/source/is/mjuk/cache/View.java
109   index b6a7c36..3060d7e 100644
110   --- a/source/is/mjuk/cache/View.java
111   +++ b/source/is/mjuk/cache/View.java
112   @@ -92,14 +92,19 @@ public class View
113               } else if (input.matches("^[ls](oad|tore)?\\s\\d+$")) {
114                   long address = Long.parseLong(input.split("\\s")[1]);
115
116   -                if (input.split("\\s")[0].matches("^l(oad)?$")) {
117   -                    System.out.println(
118   -                        c.executeInstruction("load", address).toString()
119   -                    );
120   -                } else if (input.split("\\s")[0].matches("^s(tore)?$")) {
121   -                    System.out.println(
122   -                        c.executeInstruction("store", address).toString()
123   -                    );
```

51(56)

```
124  +                        try {
125  +                            if (input.split("\\s")[0].matches("^l(oad)?$")) {
126  +                                System.out.println(
127  +                                    c.executeInstruction("load", address).toString()
128  +                                );
129  +                            } else if (input.split("\\s")[0].matches("^s(tore)?$")) {
130  +                                System.out.println(
131  +                                    c.executeInstruction("store", address).toString()
132  +                                );
133  +                            }
134  +                        } catch(IllegalAddressException e) {
135  +                            System.out.println("Error parsing memory address: " + e);
136  +                            continue;
137  +                        }
138
139                        System.out.printf(
```

3.8.3 Observer Patch

```
 1  diff --git a/source/is/mjuk/cache/Controller.java b/source/is/mjuk/cache/Controller.java
 2  index 21474a6..954cfa8 100644
 3  --- a/source/is/mjuk/cache/Controller.java
 4  +++ b/source/is/mjuk/cache/Controller.java
 5  @@ -142,4 +142,12 @@ public class Controller
 6             simDTO.setLoads(this.dataCache.getLoads());
 7             return simDTO;
 8         }
 9  +
10  +    public void addDataCacheObserver(DataCacheObserver observer) {
11  +        this.dataCache.addObserver(observer);
12  +    }
13  +
14  +    public void removeDataCacheObserver(DataCacheObserver observer) {
15  +        this.dataCache.removeObserver(observer);
16  +    }
17   }
18  diff --git a/source/is/mjuk/cache/DataCache.java b/source/is/mjuk/cache/DataCache.java
19  index 9edbd95..4a6d51e 100644
20  --- a/source/is/mjuk/cache/DataCache.java
21  +++ b/source/is/mjuk/cache/DataCache.java
22  @@ -1,5 +1,6 @@
23   package is.mjuk.cache;
24
25  +import java.util.ArrayList;
26   import java.lang.StringBuilder;
27   import java.lang.Math;
28
29  @@ -16,6 +17,8 @@ public class DataCache {
30       private int loads = 0;
31       private int stores = 0;
32       private Block[][] blockset;
33  +    private ArrayList<DataCacheObserver> observers =
34  +        new ArrayList<DataCacheObserver>();
35
36       /**
37        * Parses a cache layout and generates the block objects
38  @@ -54,6 +57,22 @@ public class DataCache {
39       }
40
```

```
41          /**
42     +    */
43     +    public void addObserver(DataCacheObserver observer) {
44     +        if (!this.observers.contains(observer)) {
45     +            this.observers.add(observer);
46     +        }
47     +    }
48     +
49     +    /**
50     +    */
51     +    public void removeObserver(DataCacheObserver observer) {
52     +        if (this.observers.contains(observer)) {
53     +            this.observers.remove(observer);
54     +        }
55     +    }
56     +
57     +    /**
58          * @return Number of sets in the cache (associativity)
59          */
60          public int getNumberOfSets() {
61  @@ -158,7 +177,15 @@ public class DataCache {
62              currentBlock = this.blockset[cacheSet][(int) address.getIndex()];
63              currentBlock.setTag(address.getTag());
64
65     +        this.notifyObservers();
66     +
67              this.misses += 1;
68              return false;
69          }
70     +
71     +    private void notifyObservers() {
72     +        for (DataCacheObserver observer : this.observers) {
73     +            observer.recvDataCacheUpdate(this.displayCache());
74     +        }
75     +    }
76      }
77  diff --git a/source/is/mjuk/cache/DataCacheObserver.java b/source/is/mjuk/cache/DataCacheObserver
78  new file mode 100644
79  index 0000000..b6fb6f7
80  --- /dev/null
81  +++ b/source/is/mjuk/cache/DataCacheObserver.java
82  @@ -0,0 +1,11 @@
83  +package is.mjuk.cache;
84  +
85  +/**
86  +* DataCacheObserver is an interface for objects that wish to observe a
87  +* {@link is.mjuk.cache.DataCache}-object
88  +*
89  +* @see is.mjuk.cache.DataCache
90  +*/
91  +public interface DataCacheObserver {
92  +    public void recvDataCacheUpdate(String dataCacheContent);
93  +}
94  \ No newline at end of file
95  diff --git a/source/is/mjuk/cache/View.java b/source/is/mjuk/cache/View.java
96  index ba71bdb..baa68f0 100644
97  --- a/source/is/mjuk/cache/View.java
98  +++ b/source/is/mjuk/cache/View.java
99  @@ -8,7 +8,7 @@ import java.util.Date;
```

```
100    * <p>
101    * Handles interaction between user and the application
102    */
103   -public class View
104   +public class View implements DataCacheObserver
105    {
106        public static Scanner scanner = new Scanner(System.in);
107        Controller c;
108   @@ -69,6 +69,7 @@ public class View
109               legalData = false;
110            }
111        } while (!legalData);
112   +
113        System.out.println("Displaying␣Cache␣Data");
114        System.out.println(c.displayCache());
115    }
116   @@ -83,47 +84,39 @@ public class View
117            System.out.println(
118                "To␣use␣instruction␣store,␣write␣'store␣<memaddress>'"
119            );
120   +        System.out.println(
121   +            "To␣observe␣changes␣in␣the␣cache,␣write␣'observe'"
122   +        );
123   +        System.out.println(
124   +            "To␣stop␣observing␣changes␣in␣the␣cache,␣write␣'deobserve'"
125   +        );
126
127            while (true) {
128                String input = scanner.nextLine();
129   -             input = input.trim();
130   +            input = input.trim();
131   +
132   +            String regex = "^[ls](oad|tore)?\\s(0[x]?)?[0-9a-fA-F]+";
133
134                if (input.toLowerCase().equals("exit")
135                    || input.toLowerCase().equals("x")) {
136                    break;
137   -            } else if (input.matches(
138   -                "^[ls](oad|tore)?\\s(0[x]?)?[0-9a-fA-F]+"
139   -            )) {
140   -                long address = Long.decode(input.split("\\s")[1]);
141   -
142   -                try {
143   -                    if (input.split("\\s")[0].matches("^l(oad)?$")) {
144   -                        System.out.println(
145   -                            c.executeInstruction("load", address).toString()
146   -                        );
147   -                    } else if (input.split("\\s")[0].matches("^s(tore)?$")) {
148   -                        System.out.println(
149   -                            c.executeInstruction("store", address).toString()
150   -                        );
151   -                    }
152   -                } catch(IllegalAddressException e) {
153   -                    System.out.println("Error␣parsing␣memory␣address:␣" + e);
154   -                    continue;
155   -                }
156   -
157   -                System.out.printf(
158   -                    "Hitrate␣is␣%.2f␣and␣missrate␣is␣%.2f.\n",
```

```
159  –                      c.getHitrate(), c.getMissrate()
160  –                  );
161  +              } else if (input.matches(regex)) {
162  +                  sendInstruction(input);
163  +              } else if (input.matches("^$")) {
164                   // Intentionally left empty
165  +              } else if (input.equals("observe")) {
166  +                  System.out.println("Now observing the DataCache for changes");
167  +                  c.addDataCacheObserver(this);
168  +              } else if (input.equals("deobserve")) {
169  +                  System.out.println("No longer observing the DataCache");
170  +                  c.removeDataCacheObserver(this);
171                } else {
172                   System.err.println("Instruction not found '" + input + "'");
173                }
174            }
175        }
176
177  –     private void endSimulation(){
178  +     private void endSimulation() {
179            System.out.println(c.displayCache());    // TODO: Remove
180            SimulationDTO simDTO = c.getSimulationData();
181            System.out.println("Simulation data:");
182  @@ -145,4 +138,35 @@ public class View
183            System.out.println("Address offset size: "
184                + simDTO.getLayoutDTO().getOffsetSize() + " bits");
185        }
186  +
187  +     /**
188  +      * Recieves data from a DataCache observee.
189  +      */
190  +     public void recvDataCacheUpdate(String dataCacheContent) {
191  +         System.out.println(dataCacheContent);
192  +     }
193  +
194  +     private void sendInstruction(String input) {
195  +         long address = Long.decode(input.split("\\s")[1]);
196  +
197  +         try {
198  +             if (input.split("\\s")[0].matches("^l(oad)?$")) {
199  +                 System.out.println(
200  +                     c.executeInstruction("load", address).toString()
201  +                 );
202  +             } else if (input.split("\\s")[0].matches("^s(tore)?$")) {
203  +                 System.out.println(
204  +                     c.executeInstruction("store", address).toString()
205  +                 );
206  +             }
207  +         } catch(IllegalAddressException e) {
208  +             System.out.println("Error parsing memory address: " + e);
209  +             return;
210  +         }
211  +
212  +         System.out.printf(
213  +             "Hitrate is %.2f and missrate is %.2f.\n",
214  +             c.getHitrate(), c.getMissrate()
215  +         );
216  +     }
217    }
```

3.9 Utilities

3.9.1 Class: MisMath

```java
1  package is.mjuk.utils;
2
3  import java.lang.Math;
4
5  /**
6   * mjuk.is Mathematics Library for Java
7   */
8  public class MisMath {
9
10     /**
11      * Returns the unary representation of an integer.
12      * <p>
13      * Sets the amount of bits in input to one.
14      *
15      * @param input Amount of bits to set to one
16      * @return A digit with a row of bits set to one
17      */
18     public static long intToUnary(int input)
19     {
20         return ((long) Math.pow(2, input)-1);
21     }
22
23     /**
24      * Calculates the logarithm of two for a double.
25      */
26     public static double log2(double n) {
27         return Math.log(n)/Math.log(2.0);
28     }
29
30     /**
31      * Calculates the logarithm of two for a integer.
32      *
33      * @see is.mjuk.utils.MisMath#log2(double n)
34      */
35     public static double log2(int n) {
36         return log2((double) n);
37     }
38  }
```