

Source code of SimJ-Basic

Table des matières

1	UML Diagram	2
2	Package simj	3
2.1	simj.SimEntity	3
2.2	simj.SimEntityFactory	3
2.3	simj.SimEvent	4
2.4	simj.SimEventEndService	5
2.5	simj.SimEventResource	6
2.6	simj.SimEventScanResources	6
2.7	simj.SimEventStartService	7
2.8	simj.SimLogger	8
2.9	simj.SimRandom	10
2.10	simj.SimResource	11
2.11	simj.SimScheduler	14
2.12	simj.SimSimulation	16
3	Package simj.util	18
4	Package simj.util.logging	18
4.1	simj.util.logging.HTMLFormatter	18
4.2	simj.util.logging.SimJFormatter	20
5	Package supermarket	21
5.1	supermarket.Caisse	21
5.2	supermarket.Client	22
5.3	supermarket.ClientFactory	23
5.4	supermarket.EvenementNouveauClient	24
5.5	supermarket.Magasin	25
5.6	supermarket.SuperMarche	26
5.7	supermarket.SuperMarcheFrame	28

1 UML Diagram

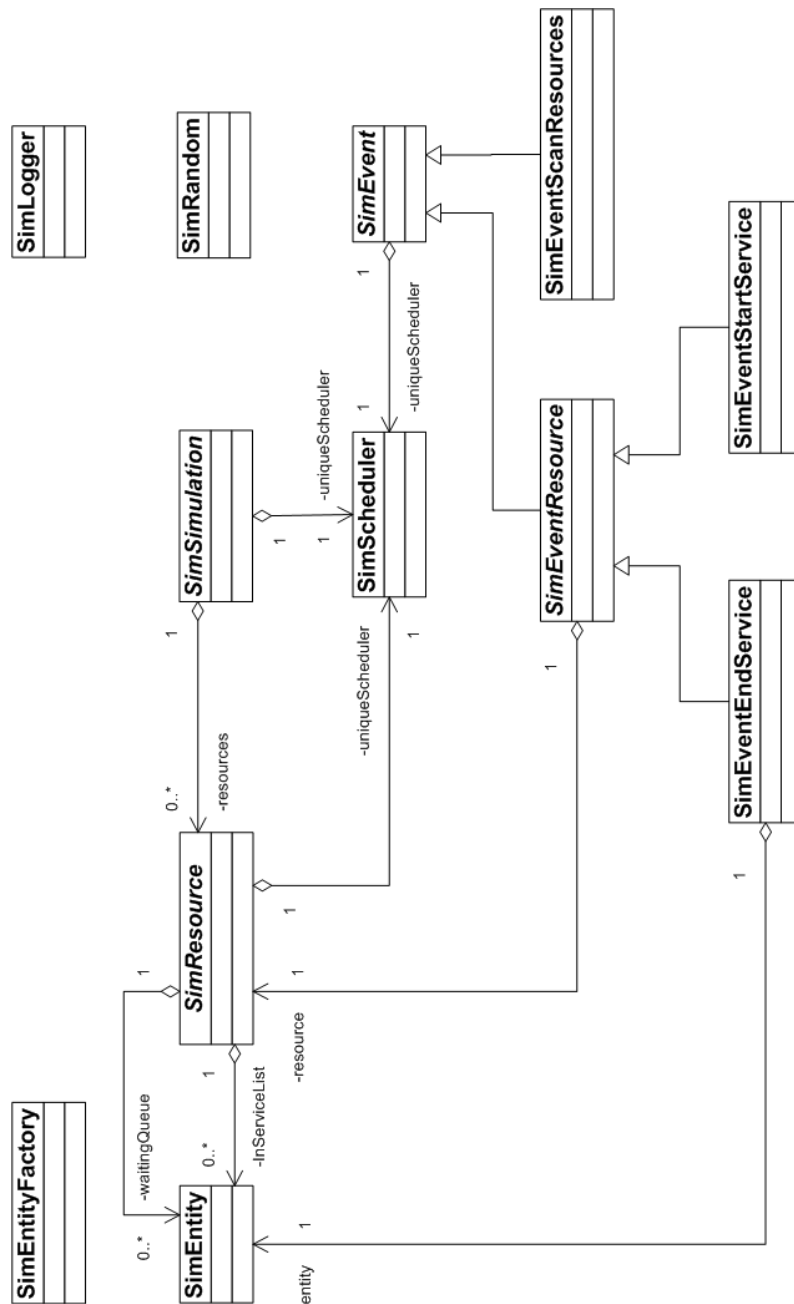


FIGURE 1 – UML diagram of SimJ

2 Package simj

2.1 simj.SimEntity

```
1  /* SimJ — A framework for discrete event simulation.
2  * @(#)SimEntity.java    08/05/09
3  *
4  * Copyright (C) 2006 Software Engineering Group — University of Fribourg (CH)
5  * URL:      http://diuf.unifr.ch/softeng
6  *
7  * This program is free software; you can redistribute it and/or
8  * modify it under the terms of the GNU General Public License
9  * as published by the Free Software Foundation:
10 * you may find a copy at the FSF website at 'www.fsf.org '.
11 */
12
13 package simj;
14
15 /**
16  * This class allows for the creation of the simplest possible temporary
17  * entity, its single feature being its ID. Subclasses must be used in order to
18  * work with entities containing more features.
19  *
20  * @version 1.0
21  * @author  <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
22  */
23 public class SimEntity {
24
25     /** The unique ID of this entity. */
26     private final int id;
27
28     /**
29      * Constructs a SimEntity object.
30      *
31      * @param pld
32      *        An int specifying the unique ID of this entity.
33      */
34     public SimEntity(final int pld) {
35         this.id = pld;
36     }
37
38     /**
39      * Returns the unique ID value.
40      *
41      * @return An int representing the unique ID of this entity.
42      */
43     public final int getId() {
44         return this.id;
45     }
46 }
```

2.2 simj.SimEntityFactory

```
1  /* SimJ — A framework for discrete event simulation.
2  * @(#)SimEntityFactory.java    08/05/09
3  *
4  * Copyright (C) 2006 Software Engineering Group — University of Fribourg (CH)
5  * URL:      http://diuf.unifr.ch/softeng
6  *
7  * This program is free software; you can redistribute it and/or
8  * modify it under the terms of the GNU General Public License
9  * as published by the Free Software Foundation:
10 * you may find a copy at the FSF website at 'www.fsf.org '.
11 */
12
13 package simj;
14
15 /**
16  * This singleton class implements an entity factory.
17  *
18  * @version 1.0
19  * @author  <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
20  */
21 public class SimEntityFactory {
22
23     private static SimEntityFactory instance = new SimEntityFactory();
24     private int counter;
25 }
```

```

27  /**
    * Protected constructor to avoid external instantiation of this singleton class.
    */
29  protected SimEntityFactory() {}

31  /**
    * Creates a new entity with an unique ID.
    * This is a template method.
    *
    * @return A new entity with an unique ID.
    */
37  public SimEntity createSimEntity() {
    counter++;

    return doCreateSimEntity(counter);
41  }

43  /**
    * Resets the entity factory. To be specific, the next generated entity
    * will have ID = 1. This method must be invoked only before the beginning
    * of a simulation. Indeed, one must be sure that the ID of an entity is
    * unique in a simulation.
    */
49  public void reset() {
    counter = 0;
51  }

53  /**
    * Creates a new entity with the given unique ID.
    * This is a factory method, and may be overridden by subclasses.
    * For instance to create more specific entities.
    *
    * @param pld The unique ID of the entity that will be created.
    *
    * @return The new entity.
    */
61  protected SimEntity doCreateSimEntity(final int pld) {
63      return new SimEntity(pld);
65  }

67  /**
    * Returns the unique instance of this singleton class.
    *
    * @return The unique instance of this singleton class.
    */
71  public static SimEntityFactory getUniqueInstance() {
    return instance;
73  }
}

```

2.3 simj.SimEvent

```

1  /* SimJ — A framework for discrete event simulation.
    * @(#)SimEvent.java    08/05/09
    *
    * Copyright (C) 2006 Software Engineering Group — University of Fribourg (CH)
    * URL:      http://diuf.unifr.ch/softeng
    *
    * This program is free software; you can redistribute it and/or
    * modify it under the terms of the GNU General Public License
    * as published by the Free Software Foundation:
    * you may find a copy at the FSF website at 'www.fsf.org'.
    */
13 package simj;

15 /**
    * This class implements the generic concept of a discrete event. The management
    * of the event firing is implemented, but the abstract execute feature must be
    * implemented.
    *
    * @version 1.0
    * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
    */
23 public abstract class SimEvent {

25     private double firingTime;

27     private SimScheduler uniqueScheduler = SimScheduler.getUniqueInstance();

```

```

29  /**
30   * Constructs a SimEvent object.
31   *
32   * @param pFiringTime
33   *       The firing or execution time of this event.
34   */
35  public SimEvent(final double pFiringTime) {
36      this.schedule(pFiringTime);
37  }
38
39  /**
40   * Does the actual work of this event.
41   * This is a primitive operation and each concrete event has to
42   * implement it appropriately.
43   *
44   */
45  public abstract void execute();
46
47  /**
48   * Schedules this event at a given time. This visibility of this
49   * method is protected to allow subclasses to reschedule themselves
50   * if needed.
51   *
52   * @param pFiringTime
53   *       The firing or execution time of this event.
54   */
55  protected final void schedule(final double pFiringTime) {
56      firingTime = pFiringTime;
57      this.uniqueScheduler.insertEvent(this);
58  }
59
60  /**
61   * Returns the firing or execution time of this event.
62   *
63   * @return The firing or execution time of this event.
64   */
65  public double getFiringTime() {
66      return firingTime;
67  }
68
69  /**
70   * Returns the current time of the simulation.
71   * This commodity method is useful for subclasses.
72   *
73   * @return The current time of the simulation.
74   */
75  protected final double getCurrentTime() {
76      return this.uniqueScheduler.getCurrentTime();
77  }
78  }

```

2.4 simj.SimEventEndService

```

1  /* SimJ – A framework for discrete event simulation.
2   * @(#)SimEventEndService.java 08/05/09
3   *
4   * Copyright (C) 2006 Software Engineering Group – University of Fribourg (CH)
5   * URL: http://diuf.unifr.ch/softeng
6   *
7   * This program is free software; you can redistribute it and/or
8   * modify it under the terms of the GNU General Public License
9   * as published by the Free Software Foundation:
10  * you may find a copy at the FSF website at 'www.fsf.org'.
11  */
12
13 package simj;
14
15 /**
16  * This class implements the discrete event, which consists of ending serving an
17  * entity at a given resource.
18  *
19  * @version 1.0
20  * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
21  */
22 public final class SimEventEndService extends SimEventResource {
23
24     /** The entity associated with this end service event. */
25     private SimEntity entity;
26
27     /**

```

```

29  * Constructs a SimEventEndService object.
30  *
31  * @param pResource
32  *       The resource associated with this end service event.
33  * @param pEntity
34  *       The entity associated with this end service event.
35  * @param pFiringTime
36  *       The firing or execution time of this event.
37  */
38  SimEventEndService(final SimResource pResource, final SimEntity pEntity,
39                    final double pFiringTime) {
40      super(pResource, pFiringTime);
41      this.entity = pEntity;
42  }
43
44  /**
45   * Terminates the service of the associated entity by the associated
46   * resource. If there are other entities waiting, it creates the
47   * corresponding <code>SimEventStartService</code>.
48   */
49  public void execute() {
50      resource.endServing(entity);
51
52      // Take the next entity which is waiting (if any) and create a
53      // corresponding start service event.
54      if (resource.hasEntitiesWaitingToBeServed()) {
55          new SimEventStartService(resource, getCurrentTime());
56      }
57  }

```

2.5 simj.SimEventResource

```

/* SimJ — A framework for discrete event simulation.
 * @(#)SimEventResource.java 08/05/09
 *
 * Copyright (C) 2006 Software Engineering Group — University of Fribourg (CH)
 * URL: http://diuf.unifr.ch/softeng
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation:
 * you may find a copy at the FSF website at 'www.fsf.org'.
 */
package simj;

/**
 * This class implements the generic concept of a discrete event associated with
 * the request or the relinquish of a resource by a temporary entity. The
 * execute feature depends on the specific event.
 */
 * @version 1.0
 * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
 */
public abstract class SimEventResource extends SimEvent {

    /** The resource associated with this event. */
    protected SimResource resource;

    /**
     * Constructs a SimEventResource object.
     *
     * @param pResource
     *       The resource associated with this event.
     * @param pFiringTime
     *       The firing or execution time of this event.
     */
    public SimEventResource(final SimResource pResource,
                           final double pFiringTime) {
        super(pFiringTime);
        this.resource = pResource;
    }
}

```

2.6 simj.SimEventScanResources

```

/* SimJ — A framework for discrete event simulation.

```

```

2  * @(#)SimEventScanResources.java    08/05/09
3  *
4  * Copyright (C) 2006 Software Engineering Group — University of Fribourg (CH)
5  * URL:      http://diuf.unifr.ch/softeng
6  *
7  * This program is free software; you can redistribute it and/or
8  * modify it under the terms of the GNU General Public License
9  * as published by the Free Software Foundation:
10 * you may find a copy at the FSF website at 'www.fsf.org '.
11 */
12
13 package simj;
14
15 /**
16  * This class implements an event, which schedules itself every
17  * <code>interval</code> and prints the status (# being served
18  * and # waiting) for each resource.
19  *
20  * @version 1.0
21  * @author  <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
22  */
23 public final class SimEventScanResources extends SimEvent {
24
25     /** The time between two resources scanning events. */
26     private final double interval;
27
28     /**
29      * Constructs a SimEventScanResources object.
30      *
31      * @param pInterval
32      *        The time between two resource scanning events.
33      */
34     public SimEventScanResources(final double pInterval) {
35         // This event will be scheduled for the first time at pInterval.
36         super(pInterval);
37         this.interval = pInterval;
38     }
39
40     /**
41      * Logs the status of all the resources of the simulation and
42      * reschedules itself.
43      *
44      */
45     public final void execute() {
46         this.displayInfo();
47         // This event reschedule itself...
48         this.schedule(getCurrentTime() + this.interval);
49     }
50
51     /**
52      * Sends the status of each resource to the logger. Information consists of
53      * the number of entities being served and the number of entities waiting
54      * to be served.
55      */
56     private void displayInfo() {
57         SimSimulation uniqueSimulation = SimSimulation.getInstance();
58         final int numberOfRes = uniqueSimulation.getNumberOfResources();
59         StringBuffer message = new StringBuffer(256);
60
61         message.append(
62             "\n=====At time ");
63         message.append(getCurrentTime());
64         for (int i = 1; i <= numberOfRes; i++) {
65             SimResource currentR = uniqueSimulation.getResource(i);
66
67             message.append("\nin resource ");
68             message.append(currentR.getResourceName());
69             message.append(", there are :\n");
70             message.append(currentR.numberofEntitiesBeingServed());
71             message.append(" entities being served and ");
72             message.append(currentR.numberofEntitiesWaitingToBeServed());
73             message.append(" waiting .\n");
74         }
75         message.append("=====\n");
76         SimLogger.getUniqueLogger().info(message.toString());
77     }
78 }

```

2.7 simj.SimEventStartService

```

1  /* SimJ – A framework for discrete event simulation.
   * @(#)SimEventStartService.java    08/05/09
   *
   * Copyright (C) 2006 Software Engineering Group – University of Fribourg (CH)
   * URL:      http://diuf.unifr.ch/softeng
   *
   * This program is free software; you can redistribute it and/or
   * modify it under the terms of the GNU General Public License
   * as published by the Free Software Foundation:
   * you may find a copy at the FSF website at 'www.fsf.org'.
   */
11
13 package simj;
14
15 /**
   * This class implements the discrete event, which consists of starting serving an
   * entity at a given resource.
   *
   * @version 1.0
   * @author  <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
   */
21 public final class SimEventStartService extends SimEventResource {
22
23     /**
24      * Constructs a SimEventStartService object.
25      *
26      * @param pResource
27      *         The resource associated with this start of service event.
28      * @param pFiringTime
29      *         The firing or execution time of this event.
30      */
31     SimEventStartService(final SimResource pResource,
32                          final double pFiringTime) {
33         super(pResource, pFiringTime);
34     }
35
36     /**
37      * Gets the next entity waiting to be served, tells the associated
38      * resource to serve it and creates the corresponding
39      * <code>SimEventEndService</code>.
40      */
41     public void execute() {
42         SimEntity nextEntityToServe = resource.getNextEntityToServe();
43         resource.startServing(nextEntityToServe);
44         double endServiceTime = resource.getEndServiceTime(nextEntityToServe);
45         new SimEventEndService(resource, nextEntityToServe, endServiceTime);
46     }
47 }

```

2.8 simj.SimLogger

```

1  /* SimJ – A framework for discrete event simulation.
   * @(#)SimLogger.java    08/05/09
   *
   * Copyright (C) 2006 Software Engineering Group – University of Fribourg (CH)
   * URL:      http://diuf.unifr.ch/softeng
   *
   * This program is free software; you can redistribute it and/or
   * modify it under the terms of the GNU General Public License
   * as published by the Free Software Foundation:
   * you may find a copy at the FSF website at 'www.fsf.org'.
   */
11
13 package simj;
14
15 import simj.util.logging.HTMLFormatter;
16 import simj.util.logging.SimJFormatter;
17
18 import java.io.File;
19 import java.io.IOException;
20
21 import java.text.MessageFormat;
22 import java.text.SimpleDateFormat;
23
24 import java.util.Calendar;
25 import java.util.logging.ConsoleHandler;
26 import java.util.logging.FileHandler;
27 import java.util.logging.Level;
28 import java.util.logging.Logger;
29
30 /**

```



```

31  * This singleton class implements a logging service.
32  *
33  * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
34  * @version 1.0
35  */
36 public class SimLogger {
37
38     private static ConsoleHandler console;
39     private static FileHandler detailedLogFile;
40     private static String detailedLogFileName = "detailed.html";
41     private static SimLogger instance = new SimLogger();
42     private static Logger logger;
43     private static final String outputFolder = System.getProperty("logging.output.dir");// "output";
44     private static FileHandler simpleLogFile;
45     private static String simpleLogFileName = "short.html";
46     private final String loggerName = "simj";
47
48     /**
49      * Private constructor in order to avoid direct instantiation of this
50      * singleton class.
51      */
52     private SimLogger() {
53         logger = Logger.getLogger(loggerName);
54         logger.setUseParentHandlers(false);
55         console = new ConsoleHandler();
56         setConsoleHandler();
57         logger.addHandler(console);
58         logger.setLevel(Level.ALL);
59         this.set(false);
60     }
61
62     private void closeFileHandlers() {
63         logger.removeHandler(simpleLogFile);
64         logger.removeHandler(detailedLogFile);
65
66         if (simpleLogFile != null) {
67             simpleLogFile.close();
68         }
69
70         if (detailedLogFile != null) {
71             detailedLogFile.close();
72         }
73     }
74
75     /**
76      * Returns the unique logger managed by this class.
77      *
78      * @return The unique instance of the logger.
79      */
80     public static Logger getUniqueLogger() {
81         return logger;
82     }
83
84     /**
85      * Returns the unique instance of this singleton class.
86      *
87      * @return The unique instance of this singleton class.
88      */
89     public static SimLogger getUniqueInstance() {
90         return instance;
91     }
92
93     /**
94      * Enables or disables the detailed logging for the current simulation.
95      * This method is invoked at the initialization of each simulation.
96      *
97      * @param pFineLogging A boolean indicating if detailed logging is enabled
98      *                    or no. If <code>>false</code> there are just the global
99      *                    scan messages displayed to the console. If
100      *                    <code>true</code> the global scan messages are
101      *                    displayed to the console, logged to a simple log file,
102      *                    and detailed messages are logged to a detailed log file.
103      */
104     public void set(final boolean pFineLogging) {
105         // First remove and close the "old" file handlers, if any.
106         closeFileHandlers();
107
108         if (pFineLogging) {
109             setFileHandlers();
110         }
111     }

```

```

113 private void setConsoleHandler() {
114     console.setLevel(Level.INFO);
115     console.setFormatter(new SimJFormatter());
116 }
117
118 private void setFileHandlers() {
119     String nowID = new SimpleDateFormat("yyMMdd_HHmss").format(
120         Calendar.getInstance().getTime());
121
122     try {
123         Object[] filenameArgs = {outputFolder, File.separator, nowID,
124             simpleLogFileName, detailedLogFileName};
125
126         simpleLogFileName = new MessageFormat(
127             "{0}{1}{2}-{3}").format(filenameArgs);
128         detailedLogFileName = new MessageFormat(
129             "{0}{1}{2}-{4}").format(filenameArgs);
130
131         simpleLogFile = new FileHandler(simpleLogFileName);
132         detailedLogFile = new FileHandler(detailedLogFileName);
133
134         HTMLFormatter htmlFormatter = new HTMLFormatter(logger.getName());
135
136         simpleLogFile.setFormatter(htmlFormatter);
137         detailedLogFile.setFormatter(htmlFormatter);
138
139         // send FINE and INFO messages to detailed logfile
140         detailedLogFile.setLevel(Level.FINE);
141         // send only INFO message to simple logfile
142         simpleLogFile.setLevel(Level.INFO);
143         logger.addHandler(simpleLogFile);
144         logger.addHandler(detailedLogFile);
145     } catch (SecurityException e) {
146         logger.severe("SecurityException during creation of logger files.");
147     } catch (IOException e) {
148         logger.severe("IOException during creation of logger files.");
149     }
150 }
151 }

```

2.9 simj.SimRandom

```

/* SimJ — A framework for discrete event simulation.
 * @(#)SimRandom.java    08/05/09
 *
 * Copyright (C) 2006 Software Engineering Group — University of Fribourg (CH)
 * URL:    http://diuf.unifr.ch/softeng
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation:
 * you may find a copy at the FSF website at 'www.fsf.org'.
 */
package simj;

/**
 * This singleton class has features for producing random numbers. There is
 * one feature for each defined statistical law.
 *
 * @version 1.0
 * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
 */
public final class SimRandom extends java.util.Random {

    private static final long serialVersionUID = 3761691199454458674L;
    private static SimRandom instance = new SimRandom();

    /**
     * Private constructor to prevent external instantiation of this singleton.
     */
    private SimRandom() {}

    /**
     * Returns a double random number following an exponential law of mean <code>mu</code>.
     *
     * @param mu The mean of the exponential random variable.
     */
}

```

```

38  * @return A random number following an exponential law of mean <code>mu</code>
39  */
40  public double expo(final double mu) {
41      return -mu * java.lang.Math.log(nextDouble());
42  }
43
44  /**
45   * Returns a double random number following a uniform law between
46   * <code>minValue</code> and <code>maxValue</code>.
47   *
48   * @param minValue The minimal possible value of the uniform random variable.
49   * @param maxValue The maximal possible value of the uniform random variable.
50   *
51   * @return A double random number following a uniform law between
52   * <code>minValue</code> and <code>maxValue</code>.
53   */
54  public double uniform(final double minValue, final double maxValue) {
55      return minValue + (maxValue - minValue) * nextDouble();
56  }
57
58  /**
59   * Returns the unique instance of this Singleton class.
60   *
61   * @return The unique instance of this class.
62   */
63  public static SimRandom getUniqueInstance() {
64      return instance;
65  }
66
67  // TODO add other probabilistic laws:
68  // cash desk (truncated) normal law with parameters mu and sigma, ...
69  // autres: Erlang, Weibull, Gamma
70  }

```

2.10 simj.SimResource

```

1  /* SimJ - A framework for discrete event simulation.
2   * @(#)SimResource.java 08/05/09
3   *
4   * Copyright (C) 2006 Software Engineering Group - University of Fribourg (CH)
5   * URL: http://diuf.unifr.ch/softeng
6   *
7   * This program is free software; you can redistribute it and/or
8   * modify it under the terms of the GNU General Public License
9   * as published by the Free Software Foundation:
10  * you may find a copy at the FSF website at 'www.fsf.org'.
11  */
12
13 package simj;
14
15 import java.util.ArrayList;
16 import java.util.List;
17 import java.util.logging.Logger;
18
19 /**
20  * This class implements the generic concept of a resource. The
21  * resource is initialized with a name and a number of service
22  * channels (capacity). It offers features for managing the arrival,
23  * the service, the departure and further management of temporary entities.
24  * The concrete service time must be reimplemented.
25  *
26  * @version 1.0
27  * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
28  */
29 public abstract class SimResource {
30
31     private int capacity;
32     private String resourceName;
33     private List<SimEntity> inServiceList;
34     private List<SimEntity> waitingQueue;
35
36     private SimScheduler uniqueScheduler = SimScheduler.getUniqueInstance();
37     private Logger uniqueLogger = SimLogger.getUniqueLogger();
38     protected SimRandom uniqueRandomizer = SimRandom.getUniqueInstance();
39
40     /**
41      * Constructs a resource.
42      *
43      * @param pCapacity The capacity of this resource, that is the maximum

```

```

45  * number of entities that can be served simultaneously by this resource.
46  * @param pResourceName The name of this resource.
47  */
48  public SimResource(final int pCapacity, final String pResourceName) {
49      this.capacity = pCapacity;
50      this.resourceName = pResourceName;
51      this.waitingQueue = new ArrayList<SimEntity>();
52      this.inServiceList = new ArrayList<SimEntity>();
53      SimSimulation.getInstance().registerResource(this);
54  }
55
56  /**
57   * Returns the number of entities being served by this resource.
58   *
59   * @return The number of entities being served by this resource.
60   */
61  public int numberOfEntitiesBeingServed() {
62      return inServiceList.size();
63  }
64
65  /**
66   * Returns the number of entities waiting to be served by this resource.
67   *
68   * @return The number of entities waiting to be served by this resource.
69   */
70  public int numberOfEntitiesWaitingToBeServed() {
71      return waitingQueue.size();
72  }
73
74  /**
75   * Starts "managing" the entity, for which the resource is requested.
76   *
77   * @param pEntity The entity requesting this resource.
78   */
79  public void request(final SimEntity pEntity) {
80      displayInfoRequest(pEntity);
81      startWaiting(pEntity);
82
83      if (isAvailable()) {
84          new SimEventStartService(this, getCurrentTime());
85      }
86  }
87
88  /**
89   * Ends the service of an entity.
90   *
91   * @param pEntity The entity that is being served by this resource.
92   */
93  void endServing(final SimEntity pEntity) {
94      displayInfoEndService(pEntity);
95      inServiceList.remove(pEntity);
96      afterEndService(pEntity);
97  }
98
99  /**
100   * Start serving an entity.
101   *
102   * @param pEntity The entity this resource is starting to serve.
103   */
104  void startServing(final SimEntity pEntity) {
105      displayInfoStartService(pEntity);
106      waitingQueue.remove(pEntity);
107      inServiceList.add(pEntity);
108  }
109
110  /**
111   * Do whatever is appropriate after the end of the service of an entity.
112   * By default it does nothing but this is a hook operation and may be
113   * overridden by subclasses (for example, "send" the entity to another
114   * resource).
115   *
116   * @param pEntity The entity that has been served by this resource.
117   */
118  protected void afterEndService(final SimEntity pEntity) {}
119
120  /**
121   * Logs default information about the end of service of an entity by this
122   * resource.
123   * This is a hook operation and may be overridden by subclasses.
124   *
125   * @param pEntity The entity that has been served by this resource.
126   */

```

```

127 protected void displayInfoEndService(final SimEntity pEntity) {
128     StringBuffer message = new StringBuffer(64);
129
130     message.append("The temp entity ");
131     message.append(pEntity.getId());
132     message.append(" is relinquished by the ");
133     message.append(this.getResourceName());
134     message.append(" at time ");
135     message.append(getCurrentTime());
136     uniqueLogger.fine(message.toString());
137 }
138
139 /**
140  * Logs default information about an entity requesting this resource.
141  * This is a hook operation and may be overridden by subclasses.
142  *
143  * @param pEntity The entity that is requesting this resource.
144  */
145 protected void displayInfoRequest(final SimEntity pEntity) {
146     StringBuffer message = new StringBuffer(64);
147
148     message.append("The temp entity ");
149     message.append(pEntity.getId());
150     message.append(" requests the ");
151     message.append(this.getResourceName());
152     message.append(" at time ");
153     message.append(getCurrentTime());
154     uniqueLogger.fine(message.toString());
155 }
156
157 /**
158  * Logs default information about the start of service of an entity by
159  * this resource.
160  * This is a hook operation and may be overridden by subclasses.
161  *
162  * @param pEntity The entity beginning to be served by this resource.
163  */
164 protected void displayInfoStartService(final SimEntity pEntity) {
165     StringBuffer message = new StringBuffer(64);
166
167     message.append("The temp entity ");
168     message.append(pEntity.getId());
169     message.append(" starts being served by the ");
170     message.append(this.getResourceName());
171     message.append(" at time ");
172     message.append(getCurrentTime());
173     uniqueLogger.fine(message.toString());
174 }
175
176 /**
177  * Logs default information each time an entity starts waiting in
178  * front of this resource.
179  * This is a hook operation and may be overridden by subclasses.
180  *
181  * @param pEntity The entity that starts waiting to be served by
182  * this resource.
183  */
184 protected void displayInfoStartWaiting(final SimEntity pEntity) {
185     StringBuffer message = new StringBuffer(64);
186
187     message.append("The temp entity ");
188     message.append(pEntity.getId());
189     message.append(" starts waiting at time ");
190     message.append(getCurrentTime());
191     uniqueLogger.fine(message.toString());
192 }
193
194 private void startWaiting(final SimEntity pEntity) {
195     displayInfoStartWaiting(pEntity);
196     waitingQueue.add(pEntity);
197 }
198
199 /**
200  * Returns the next entity to serve.
201  *
202  * @return The next entity to serve.
203  */
204 public SimEntity getNextEntityToServe() {
205     return waitingQueue.get(0);
206 }
207
208 /**

```

```

209  * Returns how much time is needed to serve the entity.
210  * This is a primitive operation, and each concrete resource has to
211  * implement it appropriately.
212  *
213  * @param pEntity The entity this resource will serve.
214  *
215  * @return The duration the entity will spend in this resource.
216  */
217  protected abstract double getServiceTime(SimEntity pEntity);
218
219  /**
220  * Returns the time at which this resource will relinquish the entity.
221  *
222  * @param pEntity The entity this resource will serve.
223  *
224  * @return The time at which the entity will have been finished serving.
225  */
226  public double getEndServiceTime(SimEntity pEntity) {
227      return getCurrentTime() + getServiceTime(pEntity);
228  }
229
230  /**
231  * Returns the current time of the simulation.
232  * This commodity method is useful for subclasses.
233  *
234  * @return The current time of the simulation.
235  */
236  protected final double getCurrentTime() {
237      return uniqueScheduler.getCurrentTime();
238  }
239
240  /**
241  * Returns the name of this resource.
242  *
243  * @return The name of this resource.
244  */
245  protected String getResourceName() {
246      return resourceName;
247  }
248
249  /**
250  * Tells if there are entities waiting to be served by this resource or not.
251  *
252  * @return <code>true</code> if there are entities waiting to be served;
253  *         <code>false</code> otherwise.
254  */
255  boolean hasEntitiesWaitingToBeServed() {
256      return !waitingQueue.isEmpty();
257  }
258
259  /**
260  * Tells if this resource is available or not.
261  *
262  * @return <code>true</code> if the resource is available;
263  *         <code>false</code> otherwise.
264  */
265  private boolean isAvailable() {
266      return numberOfEntitiesWaitingToBeServed()
267             <= capacity - numberOfEntitiesBeingServed();
268  }
269 }

```

2.11 simj.SimScheduler

```

/* SimJ — A framework for discrete event simulation.
2  * @(#)SimScheduler.java    08/05/09
3  *
4  * Copyright (C) 2006 Software Engineering Group — University of Fribourg (CH)
5  * URL:      http://diuf.unifr.ch/softeng
6  *
7  * This program is free software; you can redistribute it and/or
8  * modify it under the terms of the GNU General Public License
9  * as published by the Free Software Foundation:
10 * you may find a copy at the FSF website at 'www.fsf.org'.
11 */
12
13 package simj;
14
15 import java.util.ArrayList;
16 import java.util.TreeMap;

```

```

18 /**
19  * This class maintains the current time and has features for inserting new
20  * events or for finding the next event in the event list or future event
21  * chain (FETCH).
22  *
23  * @version 1.0
24  * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
25  */
26 public final class SimScheduler {
27
28     private static SimScheduler instance =
29         new SimScheduler();
30     private static final long serialVersionUID =
31         3834030242716397880L;
32     private static TreeMap<Double, ArrayList<SimEvent>> futureEventChain =
33         new TreeMap<Double, ArrayList<SimEvent>>();
34     private double currentTime;
35
36     /**
37      * Private constructor in order to avoid direct instantiation of this
38      * singleton class.
39      */
40     private SimScheduler() {}
41
42     /**
43      * Empties the scheduler completely.
44      *
45      */
46     public void empty() {
47         futureEventChain.clear();
48     }
49
50     /**
51      * Insert a given event in the future event chain.
52      *
53      * @param pSimEvent The event to add to the future event chain.
54      */
55     public void insertEvent(final SimEvent pSimEvent) {
56         Double key = Double.valueOf(pSimEvent.getFiringTime());
57         ArrayList<SimEvent> events = futureEventChain.get(key);
58
59         if (events == null) {
60             events = new ArrayList<SimEvent>();
61         }
62
63         events.add(pSimEvent);
64         futureEventChain.put(key, events);
65     }
66
67     /**
68      * Returns the current time of the simulation.
69      *
70      * @return The current time of the simulation.
71      */
72     public double getCurrentTime() {
73         return this.currentTime;
74     }
75
76     /**
77      * Returns the unique instance of this Singleton class.
78      *
79      * @return The unique instance of this class.
80      */
81     public static SimScheduler getUniqueInstance() {
82         return instance;
83     }
84
85     /**
86      * Returns the next event that has to be executed.
87      *
88      * @return The next scheduled event.
89      */
90     public SimEvent getNextEvent() {
91         Double key = futureEventChain.firstKey();
92         ArrayList<SimEvent> theNextEvents = futureEventChain.remove(key);
93         SimEvent theNextEvent = theNextEvents.remove(0);
94
95         if (!theNextEvents.isEmpty()) {
96             futureEventChain.put(key, theNextEvents);
97         }
98     }

```

```

100     this.currentTime = theNextEvent.getFiringTime();
101     return theNextEvent;
102 }
103
104 /**
105  * Tells if the scheduler is empty or not.
106  *
107  * @return <code>true</code> if there are no more events in the scheduler;
108  *         <code>false</code> otherwise.
109  */
110 public boolean isEmpty() {
111     return futureEventChain.isEmpty();
112 }
113 }

```

2.12 simj.SimSimulation

```

/* SimJ — A framework for discrete event simulation.
 * @(#)SimSimulation.java    08/05/09
 *
 * Copyright (C) 2006 Software Engineering Group — University of Fribourg (CH)
 * URL:      http://diuf.unifr.ch/softeng
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation:
 * you may find a copy at the FSF website at 'www.fsf.org '.
 */
package simj;

import java.util.ArrayList;
import java.util.List;
import java.util.logging.Logger;

/**
 * This class manages the simulation event loop, and provides features for
 * initializing the simulation (that is, creating and providing the unique scheduler
 * and randomizer, creating and registering resources, setting the simulation
 * end time,...). The final event time and the creation of resources depend on
 * the given simulation. These tasks are thus deferred.
 *
 * @version 1.0
 * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
 */
public abstract class SimSimulation {

    /**
     * Like in singleton classes, this is the instance of this class.
     * Since this class is abstract, this field has protected visibility,
     * to allow implementing subclasses to access it.
     */
    protected static SimSimulation instance;

    private SimScheduler uniqueScheduler = SimScheduler.getInstance();
    private SimLogger simjLogger = SimLogger.getInstance();
    private Logger uniqueLogger = SimLogger.getUniqueLogger();
    private double finalEventTime;
    private List<SimResource> resources;
    private double scanInterval;

    /**
     * Protected (to avoid external instantiation) constructor of the general
     * simulation class.
     *
     * @param pFinalEventTime The
     * @param pScanInterval The interval between two "scan events"
     * (<code>SimEventScanResources</code>).
     * @param pFineLogging A boolean determining if detailed logging is enabled
     * (<code>true</code>) for this simulation or not (<code>false</code>).
     */
    protected SimSimulation(final double pFinalEventTime,
                             final int pScanInterval,
                             final boolean pFineLogging) {
        this.resources = new ArrayList<SimResource>();
        this.finalEventTime = pFinalEventTime;
        this.scanInterval = pScanInterval;
        simjLogger.set(pFineLogging);
    }

    /**

```



```

64  * Initializes and starts the simulation.
65  * This is a template method and calls several primitive and hook operations.
66  */
67  public final void startSimulation() {
68      createResources();
69      createEvents();
70      setupSimulation();
71      SimEntityFactory.getUniqueInstance().reset();
72      new SimEventScanResources(this.scanInterval);
73      this.eventLoopManager();
74  }
75
76  /**
77   * Adds a resource to this simulation.
78   *
79   * @param pRes The resource to add to this simulation.
80   */
81  void registerResource(final SimResource pRes) {
82      resources.add(pRes);
83  }
84
85  /**
86   * Creates the bootstrapping events of the simulation.
87   * This is a primitive operation and has to be implemented by subclasses.
88   */
89  protected abstract void createEvents();
90
91  /**
92   * Creates the several resources of the simulation.
93   * This is a primitive operation and has to be implemented by subclasses.
94   */
95  protected abstract void createResources();
96
97  /**
98   * Does some additional simulation initialization configuration.
99   * This is a hook operation and may be overridden by subclasses.
100  * By default it does nothing.
101  */
102  protected void setupSimulation() {
103  }
104
105  private void eventLoopManager() {
106      SimEvent currentEvent;
107
108      for (currentEvent = uniqueScheduler.getNextEvent();
109           !isSimulationFinished();
110           currentEvent = uniqueScheduler.getNextEvent()) {
111          currentEvent.execute();
112      }
113
114      // The end of simulation time is either the firing time of the
115      // first not executed event (if the simulation time elapsed)
116      // or the firing time of the last executed event (if the
117      // scheduler has been emptied)
118      uniqueLogger.info("End of simulation at time " + SimScheduler.getUniqueInstance().getCurrentTime() + ".");
119
120      // just in case there are still event in the future event chain
121      // or scheduler.
122      uniqueScheduler.empty();
123  }
124
125  /**
126   * Returns the unique instance of this singleton class.
127   *
128   * @return The unique instance of this singleton class.
129   */
130  public static SimSimulation getInstance() {
131      return instance;
132  }
133
134  /**
135   * Returns the number of resources of the simulation.
136   *
137   * @return The number of resources of the simulation.
138   */
139  public int getNumberOfResources() {
140      return resources.size();
141  }
142
143  /**
144   * Returns the resource at the specified index.
145   *

```

```

146  * @param index An index into the resources of this simulation
147  * (between <code>1</code> and <code>getNumberOfResources</code>).
148  *
149  * @return The resource at the specified index.
150  */
151  public SimResource getResource(final int index) {
152      return resources.get(index - 1);
153  }
154
155  private boolean isSimulationFinished() {
156      return (uniqueScheduler.isEmpty() || (uniqueScheduler.getCurrentTime() > this.finalEventTime));
157  }
158  }

```

3 Package simj.util

4 Package simj.util.logging

4.1 simj.util.logging.HTMLFormatter

```

/* SimJ - A framework for discrete event simulation.
 * @(#)HTMLFormatter.java 08/05/09
 *
 * Copyright (C) 2006 Software Engineering Group - University of Fribourg (CH)
 * URL: http://diuf.unifr.ch/softeng
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation:
 * you may find a copy at the FSF website at 'www.fsf.org'.
 */
package simj.util.logging;

import java.text.SimpleDateFormat;

import java.util.Calendar;
import java.util.Date;
import java.util.logging.Formatter;
import java.util.logging.Handler;
import java.util.logging.Level;
import java.util.logging.LogRecord;

import org.apache.commons.lang.StringEscapeUtils;

/**
 * This class implements an HTMLFormatter for log records.
 * The output is usually written to a file.
 *
 * @version 1.0
 * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
 */
public class HTMLFormatter extends Formatter {

    private String loggerName;

    /**
     * Constructs an instance of an HTML formatter.
     *
     * @param name The name this formatter uses to display a title.
     */
    public HTMLFormatter(final String name) {
        super();
        loggerName = name;
    }

    /**
     * Formats a given log record in an HTML way.
     *
     * @param record The log record to format.
     *
     * @return An HTML string representing the formatted log record.
     */
    public String format(final LogRecord record) {
        StringBuffer buf = new StringBuffer(2048);

```

```

56     buf.append(getLogEntry(record));
57
58     return buf.toString();
59 }
60
61 /**
62  * Returns the header string for the HTML file containing the formatted
63  * records of this logger.
64  * This method is called just after the handler using this formatter is
65  * created.
66  * This method overrides the empty
67  * <code>java.util.logging.Formatter.getHead</code> method.
68  *
69  * @param handler The target handler.
70  *
71  * @return The header string for the HTML file containing the formatted
72  * records of this logger.
73  */
74 public String getHead(final Handler handler) {
75     String beginTime = getTime();
76     String cssDeclarations =
77         "<style type='text/css'>\n" + "BODY {\n"
78         + "FONT-FAMILY: Verdana, Arial, Helvetica, sans-serif;\n}\n"
79         + "TABLE {\n    FONT-SIZE: 75%;\n}\n</style>";
80     String htmlHead = "<HEAD>\n<TITLE>SimJ Log: " + loggerName
81         + "</TITLE>\n" + cssDeclarations + "\n</HEAD>";
82     String htmlTitle = "<H2>" + loggerName + "<BR>\nLog started: "
83         + beginTime + "</H2>";
84
85     return "<HTML>\n" + htmlHead + "\n<BODY>\n" + htmlTitle
86         + "\n<table width='100%' border='0'>\n";
87 }
88
89 /**
90  * Returns the tail string for the HTML file containing the formatted
91  * records of this logger.
92  * This method is called just after the handler using this formatter is
93  * closed.
94  * This method overrides the empty
95  * <code>java.util.logging.Formatter.getTail</code> method.
96  *
97  * @param handler The target handler.
98  *
99  * @return The tail string for the HTML file containing the formatted
100  * records of this logger.
101  */
102 public String getTail(final Handler handler) {
103     String endTime = getTime();
104     String endLog = "<H2>Log finished: " + endTime + "</H2>";
105
106     return "</TABLE>\n" + endLog + "\n</BODY>\n</HTML>\n";
107 }
108
109 private String getLogEntry(final LogRecord record) {
110     StringBuffer buf = new StringBuffer(2048);
111
112     buf.append("<TR>\n");
113     buf.append("<TD><DIV ALIGN='right'>");
114     buf.append(record.getSequenceNumber());
115     buf.append("</DIV></TD>");
116     buf.append("<TD>");
117
118     // Bold any levels >= INFO, for instance severe error messages.
119     if (record.getLevel().intValue() >= Level.INFO.intValue()) {
120         buf.append("<b>");
121
122         if (record.getLevel().intValue() >= Level.WARNING.intValue()) {
123             buf.append("<i>");
124             buf.append(record.getLevel());
125             buf.append("</i>");
126         } else {
127             buf.append(record.getLevel());
128         }
129
130         buf.append("</b>");
131     } else {
132         buf.append(record.getLevel());
133     }
134
135     buf.append("</TD>");
136     buf.append("<TD>");

```

```

138     buf.append(record.getMillis());
139     buf.append("</TD>");
140     buf.append("<TD>");
141     buf.append(record.getSourceClassName());
142     buf.append("</TD>");
143     buf.append("<TD>");
144     buf.append(record.getSourceMethodName());
145     buf.append("</TD>");
146     buf.append("<TD>");
147     buf.append("\n</TR>\n");
148     buf.append("<TR>\n");
149     buf.append("<TD>");
150     buf.append("</TD>");
151     buf.append("<TD COLSPAN=4>");
152     buf.append("<EM>");
153     buf.append(cleanUpHTML(formatMessage(record)));
154     buf.append("</EM>");
155     buf.append("</TD>");
156     buf.append("\n</TR>\n");
157     buf.append("<TR>");
158     buf.append("<TD COLSPAN=5><HR SIZE=1 NOSHADE></TD>");
159     buf.append("</TR>\n\n");
160
161     return buf.toString();
162 }
163
164 private String cleanUpHTML(String message) {
165     return StringEscapeUtils.escapeHtml(message).replaceAll("\n", "<BR>\n");
166 }
167
168 private String getTime() {
169     Date now = Calendar.getInstance().getTime();
170
171     return new SimpleDateFormat("dd/MM/yyyy").format(now) + " at "
172         + new SimpleDateFormat("HH:MM:ss").format(now);
173 }
174 }

```

4.2 simj.util.logging.SimJFormatter

```

1  /* SimJ — A framework for discrete event simulation.
2   * @(#)SimJFormatter.java    08/05/09
3   *
4   * Copyright (C) 2006 Software Engineering Group — University of Fribourg (CH)
5   * URL:      http://diuf.unifr.ch/softeng
6   *
7   * This program is free software; you can redistribute it and/or
8   * modify it under the terms of the GNU General Public License
9   * as published by the Free Software Foundation;
10  * you may find a copy at the FSF website at 'www.fsf.org'.
11  */
12
13 package simj.util.logging;
14
15 import java.util.logging.Formatter;
16 import java.util.logging.LogRecord;
17
18 /**
19  * This class implements a simple text formatter for log records.
20  * The output is usually sent to the console.
21  *
22  * @version 1.0
23  * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
24  */
25 public class SimJFormatter extends Formatter {
26
27     /**
28      * Formats a given log record in a very simple way.
29      *
30      * @param record The log record to format.
31      *
32      * @return A simple string representing the formatted log record.
33      */
34     public String format(final LogRecord record) {
35         return record.getMessage() + '\n';
36     }
37 }

```

5 Package supermarket

5.1 supermarket.Caisse

```
/* SimJ — A framework for discrete event simulation.
 * @(#)Caisse.java 08/05/09
 *
 * Copyright (C) 2006 Software Engineering Group — University of Fribourg (CH)
 * URL: http://diuf.unifr.ch/softeng
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation:
 * you may find a copy at the FSF website at 'www.fsf.org '.
 */
package supermarket;

import java.util.logging.Logger;
import simj.SimEntity;
import simj.SimLogger;
import simj.SimRandom;
import simj.SimResource;

/**
 * This class implements a cash desk for the supermarket.
 *
 * @version 1.0
 * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
 */
public class Caisse extends SimResource {

    private Logger uniqueLogger = SimLogger.getUniqueLogger();

    private double maxServiceTime;
    private double minServiceTime;

    /**
     * Constructs a cash desk resource.
     *
     * @param pCapacity
     * @param pResourceName
     * @param pMinServiceTime
     * @param pMaxServiceTime
     */
    public Caisse(final int pCapacity, final String pResourceName,
                  final double pMinServiceTime, final double pMaxServiceTime) {
        super(pCapacity, pResourceName);
        minServiceTime = pMinServiceTime;
        maxServiceTime = pMaxServiceTime;
    }

    @Override
    protected void displayInfoEndService(final SimEntity pEntity) {
        StringBuffer message = new StringBuffer(64);

        message.append("Fin de service du client ");
        message.append(pEntity.getId());
        message.append(" par la ");
        message.append(this.getResourceName());
        message.append(" au temps ");
        message.append(getCurrentTime());
        uniqueLogger.fine(message.toString());
    }

    @Override
    protected void displayInfoRequest(final SimEntity pEntity) {
        Client pClient = (Client) pEntity;
        StringBuffer message = new StringBuffer(64);

        message.append("Le client ");
        message.append(pEntity.getId());
        message.append(" demande la ");
        message.append(this.getResourceName());
        message.append(" au temps ");
        message.append(getCurrentTime());
        message.append(" avec ");
        message.append(pClient.getNbArticles());
        message.append(" articles.");
        uniqueLogger.fine(message.toString());
    }
}
```

```

76     }
77
78     @Override
79     protected void displayInfoStartService(final SimEntity pEntity) {
80         StringBuffer message = new StringBuffer(64);
81
82         message.append("Debut de service du client ");
83         message.append(pEntity.getId());
84         message.append(" par la ");
85         message.append(this.getResourceName());
86         message.append(" au temps ");
87         message.append(getCurrentTime());
88         uniqueLogger.fine(message.toString());
89     }
90
91     @Override
92     protected void displayInfoStartWaiting(final SimEntity pEntity) {
93         StringBuffer message = new StringBuffer(64);
94
95         message.append("Le client ");
96         message.append(pEntity.getId());
97         message.append(" commence Ã attendre devant la ");
98         message.append(this.getResourceName());
99         message.append(" au temps ");
100        message.append(getCurrentTime());
101        uniqueLogger.fine(message.toString());
102    }
103
104    /**
105     * Returns how much time is needed to serve a client by this cash desk.
106     * Here, it depends on a random average time to handle an article and the
107     * number of articles the current client has.
108     *
109     * @param pEntity The client this cash desk will serve.
110     *
111     * @return The time the client will spend at this cash desk.
112     */
113    protected double getServiceTime(final SimEntity pEntity) {
114        Client pClient = (Client) pEntity;
115
116        return uniqueRandomizer.uniform(minServiceTime, maxServiceTime) * pClient.getNbArticles();
117    }
118 }

```

5.2 supermarket.Client

```

1  /* SimJ - A framework for discrete event simulation.
2   * @(#)Client.java    08/05/09
3   *
4   * Copyright (C) 2006 Software Engineering Group - University of Fribourg (CH)
5   * URL:      http://diuf.unifr.ch/softeng
6   *
7   * This program is free software; you can redistribute it and/or
8   * modify it under the terms of the GNU General Public License
9   * as published by the Free Software Foundation:
10  * you may find a copy at the FSF website at 'www.fsf.org'.
11  */
12  package supermarket;
13
14  import simj.SimEntity;
15  import simj.SimRandom;
16
17  /**
18   * This client implements a client of the supermarket.
19   *
20   * @version 1.0
21   * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
22   */
23  public class Client extends SimEntity {
24
25      /** A reference to the unique Randomizer instance. */
26      private SimRandom uniqueRandomizer = SimRandom.getUniqueInstance();
27
28      private static int maxItems;
29      private final int nbItems;
30
31      /**
32       * Constructs a new client.
33       *
34       * @param pNewId The unique ID of this client.

```

```

35  */
36  public Client(final int pNewId) {
37      super(pNewId);
38      nbItems = Double.valueOf(uniqueRandomizer.uniform(1.0,
39          getMaxArticles() + 0.5)).intValue();
40  }
41
42  /**
43   * Returns the maximum number of items a client can buy.
44   *
45   * @return The maximum number of item a client can buy.
46   */
47  public int getMaxArticles() {
48      return maxItems;
49  }
50
51  /**
52   * Returns the actual number of articles this client has bought.
53   *
54   * @return The actual number of articles this client has bought.
55   */
56  public int getNbArticles() {
57      return this.nbItems;
58  }
59
60  /**
61   * Sets the maximum number of articles a client can buy.
62   *
63   * @param pMaxArticles The maximum number of articles a client can buy.
64   */
65  static void setMaxItems(final int pMaxItems) {
66      maxItems = pMaxItems;
67  }

```

5.3 supermarket.ClientFactory

```

1  /* SimJ — A framework for discrete event simulation.
2   * @(#)ClientFactory.java    08/05/09
3   *
4   * Copyright (C) 2006 Software Engineering Group — University of Fribourg (CH)
5   * URL:      http://diuf.unifr.ch/softeng
6   *
7   * This program is free software; you can redistribute it and/or
8   * modify it under the terms of the GNU General Public License
9   * as published by the Free Software Foundation:
10  * you may find a copy at the FSF website at 'www.fsf.org '.
11  */
12
13 package supermarket;
14
15 import simj.SimEntity;
16 import simj.SimEntityFactory;
17
18 /**
19  * This class implements a client factory.
20  *
21  * @version 1.0
22  * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
23  */
24 public class ClientFactory extends SimEntityFactory {
25
26     private static ClientFactory instance = new ClientFactory();
27
28     /**
29      * Private constructor to avoid external instantiation of this singleton class.
30      */
31     private ClientFactory() {}
32
33     /**
34      * Overrides <code>SimEntityFactory.doCreateSimEntity</code> method
35      * to create supermarket clients instead of generic entities.
36      *
37      * @param pId The unique ID the client will get.
38      *
39      * @return A new client with an unique ID.
40      */
41     @Override
42     protected SimEntity doCreateSimEntity(final int pId) {
43         return new Client(pId);

```

```

45     }
46
47     /**
48      * Returns the unique instance of this singleton class.
49      *
50      * @return The unique instance of this singleton class.
51      */
52     public static SimEntityFactory getUniqueInstance() {
53         return instance;
54     }
55 }

```

5.4 supermarket.EventementNouveauClient

```

1  /* SimJ — A framework for discrete event simulation.
2   * @(#)EvenementNouveauClient.java 08/05/09
3   *
4   * Copyright (C) 2006 Software Engineering Group — University of Fribourg (CH)
5   * URL: http://diuf.unifr.ch/softeng
6   *
7   * This program is free software; you can redistribute it and/or
8   * modify it under the terms of the GNU General Public License
9   * as published by the Free Software Foundation:
10  * you may find a copy at the FSF website at 'www.fsf.org'.
11  */
12
13 package supermarket;
14
15 import simj.SimEvent;
16 import simj.SimRandom;
17 import simj.SimSimulation;
18
19 /**
20  * This class implements an event which will request the shopping area
21  * resource and reschedule this event after a random amount of time.
22  *
23  * @version 1.0
24  * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
25  */
26 public class EvenementNouveauClient extends SimEvent {
27
28     private double interArrivalTime;
29
30     private SuperMarche uniqueSimulation = (SuperMarche) SimSimulation.getInstance();
31
32     /** A reference to the unique Randomizer instance. */
33     private SimRandom uniqueRandomizer = SimRandom.getUniqueInstance();
34
35     /**
36      * Constructs an event for the arrival of a new client of the supermarket.
37      *
38      * @param pInterArrivalTime The average time between two client arrivals.
39      * @param pFiringTime The firing or execution time of this event.
40      */
41     EvenementNouveauClient(final double pInterArrivalTime,
42                             final double pFiringTime) {
43         super(pFiringTime);
44         interArrivalTime = pInterArrivalTime;
45     }
46
47     /**
48      * Does the actual work of this event.
49      * That is, send a new client to the shopping area and reschedule this event.
50      *
51      */
52     public void execute() {
53         Client entity =
54             (Client) ClientFactory.getUniqueInstance().createSimEntity();
55         Magasin magasin = uniqueSimulation.getMagasin();
56
57         magasin.request(entity);
58         this.schedule(getNextArrivalTime());
59     }
60
61     private double getNextArrivalTime() {
62         return getCurrentTime() + uniqueRandomizer.expo(interArrivalTime);
63     }
64 }

```


5.5 supermarket.Magasin

```

1  /* SimJ — A framework for discrete event simulation.
2  * @(#)Magasin.java    08/05/09
3  *
4  * Copyright (C) 2006 Software Engineering Group — University of Fribourg (CH)
5  * URL:      http://diuf.unifr.ch/softeng
6  *
7  * This program is free software; you can redistribute it and/or
8  * modify it under the terms of the GNU General Public License
9  * as published by the Free Software Foundation:
10 * you may find a copy at the FSF website at 'www.fsf.org'.
11 */
12 package supermarket;
13
14 import java.util.logging.Logger;
15 import simj.SimEntity;
16 import simj.SimLogger;
17 import simj.SimRandom;
18 import simj.SimResource;
19 import simj.SimSimulation;
20
21 /**
22 * This class implements the supermarket's shopping area resource.
23 *
24 * @version 1.0
25 * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
26 */
27 public class Magasin extends SimResource {
28
29     private Logger uniqueLogger = SimLogger.getUniqueLogger();
30     private SuperMarche uniqueSimulation = (SuperMarche) SimSimulation.getInstance();
31
32     private double maxAchatTime;
33     private double minAchatTime;
34
35     /**
36     * Constructs a shopping area resource.
37     *
38     * @param pCapacity
39     * @param pResourceName
40     * @param pMinAchatTime
41     * @param pMaxAchatTime
42     */
43     public Magasin(final int pCapacity, final String pResourceName,
44                   final double pMinAchatTime, final double pMaxAchatTime) {
45         super(pCapacity, pResourceName);
46         minAchatTime = pMinAchatTime;
47         maxAchatTime = pMaxAchatTime;
48     }
49
50     /**
51     * Sends the client who just finished its shopping to the cash desk with
52     * the shorter waiting queue.
53     *
54     * @param pEntity The client that has just finished its shopping.
55     */
56     @Override
57     protected void afterEndService(final SimEntity pEntity) {
58         Caisse caisse1 = uniqueSimulation.getCaisse1();
59         Caisse caisse2 = uniqueSimulation.getCaisse2();
60
61         if ((caisse2.numberOfEntitiesWaitingToBeServed() + caisse2.numberOfEntitiesBeingServed()) < (caisse1.
62             numberOfEntitiesWaitingToBeServed() + caisse1.numberOfEntitiesBeingServed())) {
63             caisse2.request(pEntity);
64         } else {
65             caisse1.request(pEntity);
66         }
67     }
68
69     @Override
70     protected void displayInfoEndService(final SimEntity pEntity) {
71         StringBuffer message = new StringBuffer(64);
72
73         message.append("Fin d'achats du client ");
74         message.append(pEntity.getId());
75         message.append(" au temps ");
76         message.append(getCurrentTime());
77         uniqueLogger.fine(message.toString());
78     }
79 }

```

```

79  @Override
81  protected void displayInfoRequest(final SimEntity pEntity) {
83      StringBuffer message = new StringBuffer(64);

85      message.append("Le client ");
86      message.append(pEntity.getId());
87      message.append(" demande le ");
88      message.append(this.getResourceName());
89      message.append(" au temps ");
90      message.append(getCurrentTime());
91      uniqueLogger.fine(message.toString());
92  }

93  @Override
95  protected void displayInfoStartService(final SimEntity pEntity) {
97      StringBuffer message = new StringBuffer(64);

99      message.append("Entree du client ");
100     message.append(pEntity.getId());
101     message.append(" dans le ");
102     message.append(this.getResourceName());
103     message.append(" au temps ");
104     message.append(getCurrentTime());
105     uniqueLogger.fine(message.toString());
106 }

107 @Override
109 protected void displayInfoStartWaiting(final SimEntity pEntity) {
111     StringBuffer message = new StringBuffer(64);

113     message.append("Le client ");
114     message.append(pEntity.getId());
115     message.append(" commence Ã attendre au temps ");
116     message.append(getCurrentTime());
117     uniqueLogger.fine(message.toString());
118 }

119 /**
120  * Returns how much time a client will spend in the shopping area.
121  * Here, it is simply a random time between to bounds.
122  *
123  * @param pEntity The client who will spend some time in the shopping area.
124  * @return The time the client will spend in the shopping area.
125  */
126 protected double getServiceTime(final SimEntity pEntity) {
127     return uniqueRandomizer.uniform(minAchatTime, maxAchatTime);
128 }

```

5.6 supermarket.SuperMarche

```

1  /* SimJ - A framework for discrete event simulation.
2   * @(#)SuperMarche.java    08/05/09
3   *
4   * Copyright (C) 2006 Software Engineering Group - University of Fribourg (CH)
5   * URL:    http://diuf.unifr.ch/softeng
6   *
7   * This program is free software; you can redistribute it and/or
8   * modify it under the terms of the GNU General Public License
9   * as published by the Free Software Foundation:
10  * you may find a copy at the FSF website at 'www.fsf.org'.
11  */

13 package supermarket;

15 import simj.SimRandom;
16 import simj.SimSimulation;

17 /**
18  * This class implements a supermarket. There is a shopping area (resource
19  * number 1) and two cash desks (resources number 2 and 3). Each cash desk
20  * has its waiting queue. The time spent in the shopping area and in the
21  * cash desk depends on the number of items the client buys.
22  *
23  *
24  * @version 1.0
25  * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
26  */
27 public class SuperMarche extends SimSimulation {

```

```

29 private int maxNbItemsOfClient;
30 private double tempsAchatMax;
31 private double tempsAchatMin;
32 private double tempsMoyenArrivee;
33 private double tempsServiceMax;
34 private double tempsServiceMin;
35
36 /**
37  * Constructs an instance of supermarket simulation.
38  *
39  * @param pTempsMoyenArrivee
40  * @param pTempsAchatMin
41  * @param pTempsAchatMax
42  * @param pTempsServiceMin
43  * @param pTempsServiceMax
44  * @param pFinalEventTime
45  * @param pMaxNbItemsOfClient
46  * @param pScanInterval
47  * @param pFineLogging
48  */
49 private SuperMarche( final double pTempsMoyenArrivee ,
50                      final double pTempsAchatMin ,
51                      final double pTempsAchatMax ,
52                      final double pTempsServiceMin ,
53                      final double pTempsServiceMax ,
54                      final double pFinalEventTime ,
55                      final int pMaxNbItemsOfClient ,
56                      final int pScanInterval , final boolean pFineLogging) {
57     super(pFinalEventTime , pScanInterval , pFineLogging);
58     tempsMoyenArrivee = pTempsMoyenArrivee;
59     tempsAchatMin = pTempsAchatMin;
60     tempsAchatMax = pTempsAchatMax;
61     tempsServiceMin = pTempsServiceMin;
62     tempsServiceMax = pTempsServiceMax;
63     maxNbItemsOfClient = pMaxNbItemsOfClient;
64 }
65
66 /**
67  * Creates the event for the arrival of the first supermarket's client.
68  */
69 protected void createEvents() {
70
71     // Evenement pour la premiere arrivee d'un client
72     new EvenementNouveauClient(tempsMoyenArrivee ,
73                                 SimRandom.getUniqueInstance().expo(tempsMoyenArrivee));
74 }
75
76 /**
77  * Creates the resources of the supermarket simulation. That is , a
78  * shopping area and two cash desks.
79  */
80 protected void createResources() {
81
82     // Creation: Magasin — resource no. 1 (1000 = capacite du magasin)
83     new Magasin(10000, "Magasin", tempsAchatMin, tempsAchatMax);
84
85     // Creation: Caisse 1 — resource no. 2 (1 = capacite de la caisse)
86     new Caisse(1, "Caisse 1", tempsServiceMin, tempsServiceMax);
87
88     // Creation: Caisse 2 — resource no. 3 (1 = capacite de la caisse)
89     new Caisse(1, "Caisse 2", tempsServiceMin, tempsServiceMax);
90 }
91
92 /**
93  * Does some additional initialisation configurations for the Supermarket.
94  */
95 @Override
96 protected void setupSimulation() {
97
98     // Parametres des clients
99     Client.setMaxItems(this.maxNbItemsOfClient);
100    ClientFactory.getUniqueInstance().reset();
101 }
102
103 /**
104  * Returns the first cash desk resource.
105  *
106  * @return The first Caisse (cash desk) resource.
107  */
108 public Caisse getCaisse1() {
109     return (Caisse) getResource(2);
110 }

```

```

111 }
112
113 /**
114  * Returns the second cash desk resource.
115  *
116  * @return The second Caisse (cash desk) resource.
117  */
118 public Caisse getCaisse2() {
119     return (Caisse) getResource(3);
120 }
121
122 /**
123  * Returns a new instance of the supermarket simulation: initialised and
124  * ready to run.
125  *
126  * @param pTempsMoyenArrivee
127  * @param pTempsAchatMin
128  * @param pTempsAchatMax
129  * @param pTempsServiceMin
130  * @param pTempsServiceMax
131  * @param pFinalEventTime
132  * @param pMaxNbArticlesOfClient
133  * @param pScanInterval
134  * @param pFineLogging
135  *
136  * @return A ready to run supermarket simulation instance.
137  */
138 public static SuperMarche getInstance(final double pTempsMoyenArrivee,
139                                     final double pTempsAchatMin, final double pTempsAchatMax,
140                                     final double pTempsServiceMin, final double pTempsServiceMax,
141                                     final double pFinalEventTime, final int pMaxNbArticlesOfClient,
142                                     final int pScanInterval, final boolean pFineLogging) {
143     // if (instance == null) {
144     instance = new SuperMarche(pTempsMoyenArrivee, pTempsAchatMin,
145                               pTempsAchatMax, pTempsServiceMin,
146                               pTempsServiceMax, pFinalEventTime,
147                               pMaxNbArticlesOfClient, pScanInterval,
148                               pFineLogging);
149
150     // }
151     return instance;
152 }
153
154 /**
155  * Returns the shopping area resource.
156  *
157  * @return The resource Magasin (shopping area).
158  */
159 public Magasin getMagasin() {
160     return (Magasin) getResource(1);
161 }
162 }

```

5.7 supermarket.SuperMarcheFrame

```

1  /* SimJ — A framework for discrete event simulation.
2   * @(#)SuperMarcheFrame.java    08/05/09
3   *
4   * Copyright (C) 2006 Software Engineering Group — University of Fribourg (CH)
5   * URL:      http://diuf.unifr.ch/softeng
6   *
7   * This program is free software; you can redistribute it and/or
8   * modify it under the terms of the GNU General Public License
9   * as published by the Free Software Foundation:
10  * you may find a copy at the FSF website at 'www.fsf.org'.
11  */
12
13 package supermarket;
14
15 /**
16  * This class implements a GUI for entering the several parameters needed
17  * to simulate a supermarket.
18  *
19  * @version 1.0
20  * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
21  */
22 public class SuperMarcheFrame extends javax.swing.JFrame {
23

```

```

25 private static final long serialVersionUID = 4050477932868875571L;
26 private javax.swing.JCheckBox fineLogging;
27 private javax.swing.JLabel dureeLabel;
28 private javax.swing.JTextField dureeSimulation;
29 private javax.swing.JPanel jPanel1;
30 private javax.swing.JLabel maxNbArticlesPerClient;
31 private javax.swing.JTextField maxNbArticlesOfClients;
32 private javax.swing.JTextField scanInterval;
33 private javax.swing.JLabel scanIntervalLabel;
34 private javax.swing.JButton startButton;
35 private javax.swing.JTextField tempsAchatMax;
36 private javax.swing.JLabel tempsAchatMaxLabel;
37 private javax.swing.JTextField tempsAchatMin;
38 private javax.swing.JTextField tempsMoyenArrivee;
39 private javax.swing.JTextField tempsServiceMax;
40 private javax.swing.JLabel tempsServiceMaxLabel;
41 private javax.swing.JTextField tempsServiceMin;
42 private javax.swing.JLabel tempsServiceMinLabel;
43 private javax.swing.JLabel titleLabel;
44 private javax.swing.JLabel tmpsAchatMinLabel;
45 private javax.swing.JLabel tmpsmoyenLabel;

47 /** Creates new form SupermarcheFrame. */
48 public SuperMarcheFrame() {
49     initComponents();
50 }
51
52 /**
53  * The main method of this class: it displays the GUI for the supermarket
54  * simulation parametrization and creation.
55  *
56  * @param args A string array with the command line arguments (here they
57  * are not used).
58  */
59 public static void main(String[] args) {
60     new SuperMarcheFrame().setVisible(true);
61 }
62
63 /** Exit the Application. */
64 private void exitForm() {
65     System.exit(0);
66 }
67
68 /**
69  * This method is called from within the constructor to initialize the form.
70  * WARNING: Do NOT modify this code. The content of this method is always
71  * regenerated by the Form Editor.
72  */
73 private void initComponents() {
74     jPanel1 = new javax.swing.JPanel();
75     titleLabel = new javax.swing.JLabel();
76     tmpsmoyenLabel = new javax.swing.JLabel();
77     tempsMoyenArrivee = new javax.swing.JTextField();
78     tmpsAchatMinLabel = new javax.swing.JLabel();
79     tempsAchatMin = new javax.swing.JTextField();
80     tempsAchatMaxLabel = new javax.swing.JLabel();
81     tempsAchatMax = new javax.swing.JTextField();
82     tempsServiceMinLabel = new javax.swing.JLabel();
83     tempsServiceMin = new javax.swing.JTextField();
84     tempsServiceMaxLabel = new javax.swing.JLabel();
85     tempsServiceMax = new javax.swing.JTextField();
86     fineLogging = new javax.swing.JCheckBox();
87     startButton = new javax.swing.JButton();
88     maxNbArticlesPerClient = new javax.swing.JLabel();
89     dureeLabel = new javax.swing.JLabel();
90     dureeSimulation = new javax.swing.JTextField();
91     scanIntervalLabel = new javax.swing.JLabel();
92     scanInterval = new javax.swing.JTextField();
93     maxNbArticlesOfClients = new javax.swing.JTextField();
94     getContentPane().setLayout(new java.awt.GridBagLayout());
95
96     java.awt.GridBagConstraints gridBagConstraints1;
97
98     setTitle("Application of SimJ");
99     setName("MainSuperMarche");
100    setResizable(false);
101    addWindowListener(new java.awt.event.WindowAdapter() {
102
103        public void windowClosing(final java.awt.event.WindowEvent evt) {
104            exitForm();
105        }
106    })

```

```

107    });
108    jPanel1.setLayout(new java.awt.GridBagLayout());
109
110    java.awt.GridBagConstraints gridBagConstraints2;
111
112    jPanel1.setPreferredSize(new java.awt.Dimension(500, 400));
113    jPanel1.setMinimumSize(new java.awt.Dimension(500, 400));
114    titleLabel.setText("Supermarché — Simulation");
115    titleLabel.setForeground(new java.awt.Color(51, 51, 255));
116    titleLabel.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
117    titleLabel.setFont(new java.awt.Font("Arial", 1, 24));
118    gridBagConstraints2 = new java.awt.GridBagConstraints();
119    gridBagConstraints2.gridx = 0;
120    gridBagConstraints2.gridy = 0;
121    gridBagConstraints2.gridwidth = 2;
122    gridBagConstraints2.fill = java.awt.GridBagConstraints.BOTH;
123    gridBagConstraints2.weightx = 1.0;
124    gridBagConstraints2.weighty = 0.1;
125    jPanel1.add(titleLabel, gridBagConstraints2);
126    tempsmoyenLabel.setText("Temps moyen entre deux arriv\u00e9es : ");
127    tempsmoyenLabel.setHorizontalAlignment(javax.swing.SwingConstants.LEFT);
128    gridBagConstraints2 = new java.awt.GridBagConstraints();
129    gridBagConstraints2.gridx = 0;
130    gridBagConstraints2.gridy = 1;
131    gridBagConstraints2.fill = java.awt.GridBagConstraints.BOTH;
132    gridBagConstraints2.weightx = 0.8;
133    gridBagConstraints2.weighty = 0.09;
134    jPanel1.add(tempsmoyenLabel, gridBagConstraints2);
135    tempsMoyenArrivee.setColumns(5);
136    tempsMoyenArrivee.setText("100");
137    gridBagConstraints2 = new java.awt.GridBagConstraints();
138    gridBagConstraints2.gridx = 1;
139    gridBagConstraints2.gridy = 1;
140    gridBagConstraints2.fill = java.awt.GridBagConstraints.BOTH;
141    gridBagConstraints2.weightx = 0.2;
142    gridBagConstraints2.weighty = 0.09;
143    jPanel1.add(tempsMoyenArrivee, gridBagConstraints2);
144    tempsAchatMinLabel.setText("Temps d'achat minimal : ");
145    tempsAchatMinLabel.setHorizontalAlignment(
146        javax.swing.SwingConstants.LEFT);
147    gridBagConstraints2 = new java.awt.GridBagConstraints();
148    gridBagConstraints2.gridx = 0;
149    gridBagConstraints2.gridy = 2;
150    gridBagConstraints2.fill = java.awt.GridBagConstraints.BOTH;
151    gridBagConstraints2.weightx = 0.8;
152    gridBagConstraints2.weighty = 0.09;
153    jPanel1.add(tempsAchatMinLabel, gridBagConstraints2);
154    tempsAchatMin.setColumns(5);
155    tempsAchatMin.setText("300");
156    gridBagConstraints2 = new java.awt.GridBagConstraints();
157    gridBagConstraints2.gridx = 1;
158    gridBagConstraints2.gridy = 2;
159    gridBagConstraints2.fill = java.awt.GridBagConstraints.BOTH;
160    gridBagConstraints2.weightx = 0.2;
161    gridBagConstraints2.weighty = 0.09;
162    jPanel1.add(tempsAchatMin, gridBagConstraints2);
163    tempsAchatMaxLabel.setText("Temps d'achat maximal : ");
164    tempsAchatMaxLabel.setHorizontalAlignment(
165        javax.swing.SwingConstants.LEFT);
166    gridBagConstraints2 = new java.awt.GridBagConstraints();
167    gridBagConstraints2.gridx = 0;
168    gridBagConstraints2.gridy = 3;
169    gridBagConstraints2.fill = java.awt.GridBagConstraints.BOTH;
170    gridBagConstraints2.weightx = 0.8;
171    gridBagConstraints2.weighty = 0.09;
172    jPanel1.add(tempsAchatMaxLabel, gridBagConstraints2);
173    tempsAchatMax.setColumns(5);
174    tempsAchatMax.setText("800");
175    gridBagConstraints2 = new java.awt.GridBagConstraints();
176    gridBagConstraints2.gridx = 1;
177    gridBagConstraints2.gridy = 3;
178    gridBagConstraints2.fill = java.awt.GridBagConstraints.BOTH;
179    gridBagConstraints2.weightx = 0.2;
180    gridBagConstraints2.weighty = 0.09;
181    jPanel1.add(tempsAchatMax, gridBagConstraints2);
182    tempsServiceMinLabel.setText(
183        "Temps minimal de service pour un article : ");
184    tempsServiceMinLabel.setHorizontalAlignment(
185        javax.swing.SwingConstants.LEFT);
186    gridBagConstraints2 = new java.awt.GridBagConstraints();
187    gridBagConstraints2.gridx = 0;
188    gridBagConstraints2.gridy = 4;

```

```

189 gridBagConstraints2.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints2.weightx = 0.8;
191 gridBagConstraints2.weighty = 0.09;
jPanel1.add(tempsServiceMinLabel, gridBagConstraints2);
193 tempsServiceMin.setColumns(5);
tempsServiceMin.setText("20");
195 gridBagConstraints2 = new java.awt.GridBagConstraints();
gridBagConstraints2.gridx = 1;
197 gridBagConstraints2.gridy = 4;
gridBagConstraints2.fill = java.awt.GridBagConstraints.BOTH;
199 gridBagConstraints2.weightx = 0.2;
gridBagConstraints2.weighty = 0.09;
201 jPanel1.add(tempsServiceMin, gridBagConstraints2);
tempsServiceMaxLabel.setText(
203 "Temps maximal de service pour un article : ");
tempsServiceMaxLabel.setHorizontalAlignment(
205 javax.swing.SwingConstants.LEFT);
gridBagConstraints2 = new java.awt.GridBagConstraints();
207 gridBagConstraints2.gridx = 0;
gridBagConstraints2.gridy = 5;
209 gridBagConstraints2.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints2.weightx = 0.8;
211 gridBagConstraints2.weighty = 0.09;
jPanel1.add(tempsServiceMaxLabel, gridBagConstraints2);
213 tempsServiceMax.setColumns(5);
tempsServiceMax.setText("40");
215 gridBagConstraints2 = new java.awt.GridBagConstraints();
gridBagConstraints2.gridx = 1;
217 gridBagConstraints2.gridy = 5;
gridBagConstraints2.fill = java.awt.GridBagConstraints.BOTH;
219 gridBagConstraints2.weightx = 0.2;
gridBagConstraints2.weighty = 0.09;
221 jPanel1.add(tempsServiceMax, gridBagConstraints2);
fineLogging.setForeground(new java.awt.Color(255, 51, 51));
223 fineLogging.setText("Fine Logging");
gridBagConstraints2 = new java.awt.GridBagConstraints();
225 gridBagConstraints2.gridx = 0;
gridBagConstraints2.gridy = 9;
227 gridBagConstraints2.gridwidth = 2;
gridBagConstraints2.fill = java.awt.GridBagConstraints.BOTH;
229 gridBagConstraints2.weightx = 1.0;
gridBagConstraints2.weighty = 0.09;
231 jPanel1.add(fineLogging, gridBagConstraints2);
startButton.setForeground(new java.awt.Color(255, 51, 51));
233 startButton.setText("Start Simulation");
startButton.setBackground(java.awt.Color.lightGray);
235 startButton.addActionListener(new java.awt.event.ActionListener() {

237     public void actionPerformed(final java.awt.event.ActionEvent evt) {
        startButtonActionPerformed();
239     }

241 });
gridBagConstraints2 = new java.awt.GridBagConstraints();
243 gridBagConstraints2.gridx = 0;
gridBagConstraints2.gridy = 10;
245 gridBagConstraints2.gridwidth = 2;
gridBagConstraints2.fill = java.awt.GridBagConstraints.BOTH;
247 gridBagConstraints2.weightx = 1.0;
gridBagConstraints2.weighty = 0.09;
249 jPanel1.add(startButton, gridBagConstraints2);
maxNbArticlesPerClient.setText("Nombre maximal d'articles par client:");
251 maxNbArticlesPerClient.setHorizontalAlignment(
    javax.swing.SwingConstants.LEFT);
253 gridBagConstraints2 = new java.awt.GridBagConstraints();
gridBagConstraints2.gridx = 0;
255 gridBagConstraints2.gridy = 6;
gridBagConstraints2.fill = java.awt.GridBagConstraints.BOTH;
257 gridBagConstraints2.weightx = 0.8;
gridBagConstraints2.weighty = 0.09;
259 jPanel1.add(maxNbArticlesPerClient, gridBagConstraints2);
dureeLabel.setText("Durée de la simulation :");
261 dureeLabel.setForeground(new java.awt.Color(255, 51, 51));
dureeLabel.setHorizontalAlignment(javax.swing.SwingConstants.LEFT);
263 gridBagConstraints2 = new java.awt.GridBagConstraints();
gridBagConstraints2.gridx = 0;
265 gridBagConstraints2.gridy = 7;
gridBagConstraints2.fill = java.awt.GridBagConstraints.BOTH;
267 gridBagConstraints2.weightx = 0.8;
gridBagConstraints2.weighty = 0.09;
269 jPanel1.add(dureeLabel, gridBagConstraints2);
dureeSimulation.setColumns(5);

```

```

271     dureeSimulation.setForeground(new java.awt.Color(255, 51, 51));
272     dureeSimulation.setText("80000");
273     gridBagConstraints2 = new java.awt.GridBagConstraints();
274     gridBagConstraints2.gridx = 1;
275     gridBagConstraints2.gridy = 7;
276     gridBagConstraints2.fill = java.awt.GridBagConstraints.BOTH;
277     gridBagConstraints2.weightx = 0.2;
278     gridBagConstraints2.weighty = 0.09;
279     jPanel1.add(dureeSimulation, gridBagConstraints2);
280     scanIntervalLabel.setText(
281         "Intervalle entre deux \"scan\" des ressources : ");
282     scanIntervalLabel.setForeground(new java.awt.Color(255, 51, 51));
283     scanIntervalLabel.setHorizontalAlignment(
284         javax.swing.SwingConstants.LEFT);
285     gridBagConstraints2 = new java.awt.GridBagConstraints();
286     gridBagConstraints2.gridx = 0;
287     gridBagConstraints2.gridy = 8;
288     gridBagConstraints2.fill = java.awt.GridBagConstraints.BOTH;
289     gridBagConstraints2.weightx = 0.8;
290     gridBagConstraints2.weighty = 0.09;
291     jPanel1.add(scanIntervalLabel, gridBagConstraints2);
292     scanInterval.setColumns(5);
293     scanInterval.setForeground(new java.awt.Color(255, 51, 51));
294     scanInterval.setText("4000");
295     gridBagConstraints2 = new java.awt.GridBagConstraints();
296     gridBagConstraints2.gridx = 1;
297     gridBagConstraints2.gridy = 8;
298     gridBagConstraints2.fill = java.awt.GridBagConstraints.BOTH;
299     gridBagConstraints2.weightx = 0.2;
300     gridBagConstraints2.weighty = 0.09;
301     jPanel1.add(scanInterval, gridBagConstraints2);
302     maxNbArticlesOfClients.setColumns(5);
303     maxNbArticlesOfClients.setText("20");
304     gridBagConstraints2 = new java.awt.GridBagConstraints();
305     gridBagConstraints2.gridx = 1;
306     gridBagConstraints2.gridy = 6;
307     gridBagConstraints2.fill = java.awt.GridBagConstraints.BOTH;
308     gridBagConstraints2.weightx = 0.2;
309     gridBagConstraints2.weighty = 0.09;
310     jPanel1.add(maxNbArticlesOfClients, gridBagConstraints2);
311     gridBagConstraints1 = new java.awt.GridBagConstraints();
312     gridBagConstraints1.weightx = 1.0;
313     gridBagConstraints1.weighty = 1.0;
314     getContentPane().add(jPanel1, gridBagConstraints1);
315     pack();
316 }
317
318 private void startButtonActionPerformed() {
319     SuperMarche supermarche =
320         (SuperMarche) SuperMarche.getInstance(
321             Double.parseDouble(tempsMoyenArrivee.getText()),
322             Double.parseDouble(tempsAchatMin.getText()),
323             Double.parseDouble(tempsAchatMax.getText()),
324             Double.parseDouble(tempsServiceMin.getText()),
325             Double.parseDouble(tempsServiceMax.getText()),
326             Double.parseDouble(dureeSimulation.getText()),
327             Integer.parseInt(maxNbArticlesOfClients.getText()),
328             Integer.parseInt(scanInterval.getText()), fineLogging.isSelected());
329
330     supermarche.startSimulation();
331 }

```