Source code of SimJ-Basic

Table des matières

1 UML Diagram

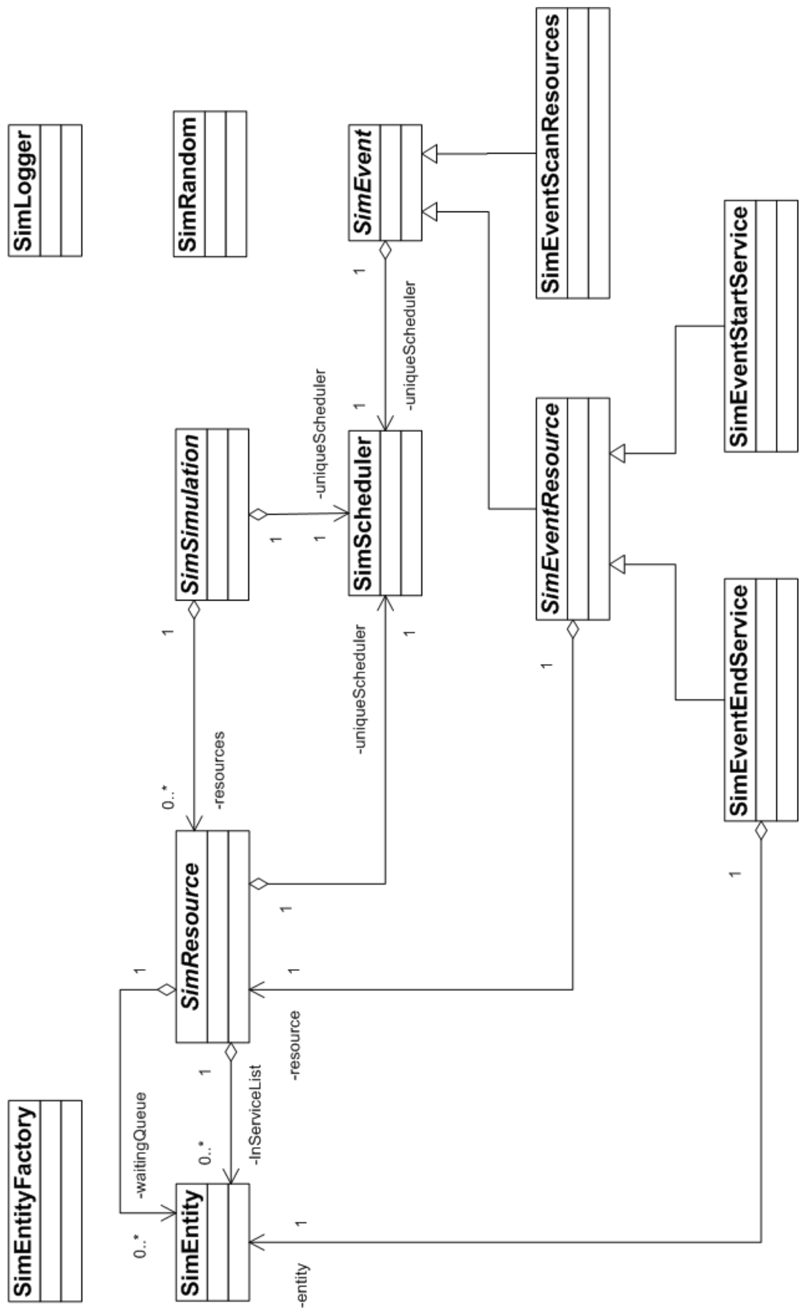

FIGURE 1 – UML diagram of SimJ

2 Package simj

2.1 simj.SimEntity

```
/* SimJ - A framework for discrete event simulation.
 * @(#)SimEntity.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 allows for the creation of the simpliest possible temporary
 * entity, its single feature being its ID. Subclasses must be used in order to
 * work with entities containing more features.
 *
 * @version 1.0
 * @author  <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
 */
public class SimEntity {

    /** The unique ID of this entity. */
    private final int id;

    /**
     * Constructs a SimEntity object.
     *
     * @param pId
     *            An int specifying the unique ID of this entity.
     */
    public SimEntity(final int pId) {
        this.id = pId;
    }

    /**
     * Returns the unique ID value.
     *
     * @return An int representing the unique ID of this entity.
     */
    public final int getId() {
        return this.id;
    }
}
```

2.2 simj.SimEntityFactory

```
/* SimJ - A framework for discrete event simulation.
 * @(#)SimEntityFactory.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 implements an entity factory.
 *
 * @version 1.0
 * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
 */
public class SimEntityFactory {

    private static SimEntityFactory instance = new SimEntityFactory();
    private int counter;

```

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

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

        return doCreateSimEntity(counter);
    }

    /**
     * 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.
     */
    public void reset() {
        counter = 0;
    }

    /**
     * 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 pId The unique ID of the entity that will be created.
     *
     * @return The new entity.
     */
    protected SimEntity doCreateSimEntity(final int pId) {
        return new SimEntity(pId);
    }

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

2.3 simj.SimEvent

```java
/* 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'.
 */

package simj;

/**
 * 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>
 */
public abstract class SimEvent {

    private double firingTime;

    private SimScheduler uniqueScheduler = SimScheduler.getUniqueInstance();
    /**
     * Constructs a SimEvent object.
     *
     * @param pFiringTime
     *                 The firing or execution time of this event.
     */
    public SimEvent(final double pFiringTime) {
        this.schedule(pFiringTime);
    }

    /**
     * Does the actual work of this event.
     * This is a primitive operation and each concrete event has to
     * implement it appropriately.
     *
     */
    public abstract void execute();

    /**
     * Schedules this event at a given time. This visibility of this
     * method is protected to allow subclasses to reschedule themselves
     * if needed.
     *
     * @param pFiringTime
     *                 The firing or execution time of this event.
     */
    protected final void schedule(final double pFiringTime) {
        firingTime = pFiringTime;
        this.uniqueScheduler.insertEvent(this);
    }

    /**
     * Returns the firing or execution time of this event.
     *
     * @return The firing or execution time of this event.
     */
    public double getFiringTime() {
        return firingTime;
    }

    /**
     * Returns the current time of the simulation.
     * This commoditiy method is useful for subclasses.
     *
     * @return The current time of the simulation.
     */
    protected final double getCurrentTime() {
        return this.uniqueScheduler.getCurrentTime();
    }
}
```

2.4 simj.SimEventEndService

```
/* SimJ – A framework for discrete event simulation.
 * @(#)SimEventEndService.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 discrete event, which consists of ending serving an
 * entity at a given resource.
 *
 * @version 1.0
 * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
 */
public final class SimEventEndService extends SimEventResource {

    /** The entity associated with this end service event. */
    private SimEntity entity;

    /**
```

```
     * Constructs a SimEventEndService object.
     *
     * @param pResource
     *                 The resource associated with this end service event.
     * @param pEntity
     *                 The entity associated with this end service event.
     * @param pFiringTime
     *                 The firing or execution time of this event.
     */
    SimEventEndService(final SimResource pResource, final SimEntity pEntity,
                       final double pFiringTime) {
        super(pResource, pFiringTime);
        this.entity = pEntity;
    }

    /**
     * Terminates the service of the associated entity by the associated
     * resource. If there are other entities waiting, it creates the
     * corresponding <code>SimEventStartService</code>.
     */
    public void execute() {
        resource.endServing(entity);

        // Take the next entity which is waiting (if any) and create a
        // corresponding start service event.
        if (resource.hasEntitiesWaitingToBeServed()) {
            new SimEventStartService(resource, getCurrentTime());
        }
    }
}
```

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.
```

```java
 * @(#)SimEventScanResources.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 an event, which schedules itself every
 * <code>interval</code> and prints the status (# being served
 * and # waiting) for each resource.
 *
 * @version 1.0
 * @author  <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
 */
public final class SimEventScanResources extends SimEvent {

    /** The time between two resources scanning events. */
    private final double interval;

    /**
     * Constructs a SimEventScanResources object.
     *
     * @param pInterval
     *            The time between two recource scanning events.
     */
    public SimEventScanResources(final double pInterval) {
        // This event will be scheduled for the first time at pInterval.
        super(pInterval);
        this.interval = pInterval;
    }

    /**
     * Logs the status of all the resources of the simulation and
     * reschedules itself.
     *
     */
    public final void execute() {
        this.displayInfo();
        // This event reschudeles itself...
        this.schedule(getCurrentTime() + this.interval);
    }

    /**
     * Sends the status of each resource to the logger. Information consists of
     * the number of entities being served and the number of entities waiting
     * to be served.
     */
    private void displayInfo() {
        SimSimulation uniqueSimulation = SimSimulation.getInstance();
        final int numberOfRes = uniqueSimulation.getNumberOfResources();
        StringBuffer message  = new StringBuffer(256);

        message.append(
            "\n============================================\nAt time ");
        message.append(getCurrentTime());
        for (int i = 1; i <= numberOfRes; i++) {
            SimResource currentR = uniqueSimulation.getResource(i);

            message.append("\nin resource ");
            message.append(currentR.getResourceName());
            message.append(", there are:\n");
            message.append(currentR.numberOfEntitiesBeingServed());
            message.append(" entities being served and ");
            message.append(currentR.numberOfEntitiesWaitingToBeServed());
            message.append(" waiting.\n");
        }
        message.append("============================================\n");
        SimLogger.getUniqueLogger().info(message.toString());
    }
}
```

2.7 simj.SimEventStartService

```java
/* 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'.
 */

package simj;

/**
 * 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>
 */
public final class SimEventStartService extends SimEventResource {

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

    /**
     * Gets the next entity waiting to be served, tells the associated
     * resource to serve it and creates the corresponding
     * <code>SimEventEndService</code>.
     */
    public void execute() {
        SimEntity nextEntityToServe = resource.getNextEntityToServe();
        resource.startServing(nextEntityToServe);
        double endServiceTime = resource.getEndServiceTime(nextEntityToServe);
        new SimEventEndService(resource, nextEntityToServe, endServiceTime);
    }
}
```

2.8 simj.SimLogger

```java
/* 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'.
 */
package simj;

import simj.util.logging.HTMLFormatter;
import simj.util.logging.SimJFormatter;

import java.io.File;
import java.io.IOException;

import java.text.MessageFormat;
import java.text.SimpleDateFormat;

import java.util.Calendar;
import java.util.logging.ConsoleHandler;
import java.util.logging.FileHandler;
import java.util.logging.Level;
import java.util.logging.Logger;

/**
```

```java
 * This singleton class implements a logging service.
 *
 * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
 * @version 1.0
 */
public class SimLogger {

    private static ConsoleHandler console;
    private static FileHandler detailedLogFile;
    private static String detailedLogFileName = "detailed.html";
    private static SimLogger instance = new SimLogger();
    private static Logger logger;
    private static final String outputFolder = System.getProperty("logging.output.dir");//"output";
    private static FileHandler simpleLogFile;
    private static String simpleLogFileName = "short.html";
    private final String loggerName = "simj";

    /**
     * Private constructor in order to avoid direct instantiation of this
     * singleton class.
     */
    private SimLogger() {
        logger = Logger.getLogger(loggerName);
        logger.setUseParentHandlers(false);
        console = new ConsoleHandler();
        setConsoleHandler();
        logger.addHandler(console);
        logger.setLevel(Level.ALL);
        this.set(false);
    }

    private void closeFileHandlers() {
        logger.removeHandler(simpleLogFile);
        logger.removeHandler(detailedLogFile);

        if (simpleLogFile != null) {
            simpleLogFile.close();
        }

        if (detailedLogFile != null) {
            detailedLogFile.close();
        }
    }

    /**
     * Returns the unique logger managed by this class.
     *
     * @return The unique instance of the logger.
     */
    public static Logger getUniqueLogger() {
        return logger;
    }

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

    /**
     * Enables or disables the detailed logging for the current simulation.
     * This method is invoked at the initialization of each simulation.
     *
     * @param pFineLogging A boolean indicating if detailed logging is enabled
     *                     or no. If <code>false</code> there are just the global
     *                     scan messages displayed to the console. If
     *                     <code>true</code> the global scan messages are
     *                     displayed to the console, logged to a simple log file,
     *                     and detailed messages are logged to a detailed log file.
     */
    public void set(final boolean pFineLogging) {

        // First remove and close the "old" file handlers, if any.
        closeFileHandlers();

        if (pFineLogging) {
            setFileHandlers();
        }
    }
    private void setConsoleHandler() {
        console.setLevel(Level.INFO);
        console.setFormatter(new SimJFormatter());
    }

    private void setFileHandlers() {
        String nowID = new SimpleDateFormat("yyMMdd_HHmmss").format(
                Calendar.getInstance().getTime());

        try {
            Object[] filenameArgs = {outputFolder, File.separator, nowID,
                simpleLogFileName, detailedLogFileName};

            simpleLogFileName = new MessageFormat(
                    "{0}{1}{2}-{3}").format(filenameArgs);
            detailedLogFileName = new MessageFormat(
                    "{0}{1}{2}-{4}").format(filenameArgs);

            simpleLogFile = new FileHandler(simpleLogFileName);
            detailedLogFile = new FileHandler(detailedLogFileName);

            HTMLFormatter htmlFormatter = new HTMLFormatter(logger.getName());

            simpleLogFile.setFormatter(htmlFormatter);
            detailedLogFile.setFormatter(htmlFormatter);

            // send FINE and INFO messages to detailed logfile
            detailedLogFile.setLevel(Level.FINE);
            // send only INFO message to simple logfile
            simpleLogFile.setLevel(Level.INFO);
            logger.addHandler(simpleLogFile);
            logger.addHandler(detailedLogFile);
        } catch (SecurityException e) {
            logger.severe("SecurityException during creation of logger files.");
        } catch (IOException e) {
            logger.severe("IOException during creation of logger files.");
        }
    }
}
```

2.9 simj.SimRandom

```java
/* 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.
     *
     * @return A random number following an exponential law of mean <code>mu</code>
     */
    public double expo(final double mu) {
        return -mu * java.lang.Math.log(nextDouble());
    }

    /**
     * Returns a double random number following a uniform law between
     * <code>minValue</code> and <code>maxValue</code>.
     *
     * @param minValue The minimal possible value of the uniform random variable.
     * @param maxValue The maximal possible value of the uniform random variable.
     *
     * @return A double random number following a uniform law between
     * <code>minValue</code> and <code>maxValue</code>.
     */
    public double uniform(final double minValue, final double maxValue) {
        return minValue + (maxValue - minValue) * nextDouble();
    }

    /**
     * Returns the unique instance of this Singleton class.
     *
     * @return The unique instance of this class.
     */
    public static SimRandom getUniqueInstance() {
        return instance;
    }

    // TODO add other probabilistic laws:
    // cash desk (truncated) normal law with parameters mu and sigma, triangular, ...
    // autres: Erlang, Weibull, Gamma
}
```

2.10 simj.SimResource

```
/* SimJ - A framework for discrete event simulation.
 * @(#)SimResource.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 implements the generic concept of a resource. The
 * resource is initialized with a name and a number of service
 * channels (capacity). It offers features for managing the arrival,
 * the service, the departure and further management of temporary entities.
 * The concrete service time must be reimplemented.
 *
 * @version 1.0
 * @author  <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
 */
public abstract class SimResource {

    private int capacity;
    private String resourceName;
    private List<SimEntity> inServiceList;
    private List<SimEntity> waitingQueue;

    private SimScheduler uniqueScheduler = SimScheduler.getUniqueInstance();
    private Logger uniqueLogger = SimLogger.getUniqueLogger();
    protected SimRandom uniqueRandomizer = SimRandom.getUniqueInstance();


    /**
     * Constructs a resource.
     *
     * @param pCapacity The capacity of this resource, that is the maximum
```

```java
     * number of entities that can be served simultaneously by this resource.
     * @param pResourceName The name of this resource.
     */
    public SimResource(final int pCapacity, final String pResourceName) {
        this.capacity      = pCapacity;
        this.resourceName  = pResourceName;
        this.waitingQueue  = new ArrayList<SimEntity>();
        this.inServiceList = new ArrayList<SimEntity>();
        SimSimulation.getInstance().registerResource(this);
    }

    /**
     * Returns the number of entities being served by this resource.
     *
     * @return The number of entities being served by this resource.
     */
    public int numberOfEntitiesBeingServed() {
        return inServiceList.size();
    }

    /**
     * Returns the number of entities waiting to be served by this resource.
     *
     * @return The number of entities waiting to be served by this resource.
     */
    public int numberOfEntitiesWaitingToBeServed() {
        return waitingQueue.size();
    }

    /**
     * Starts "managing" the entity, for which the resource is requested.
     *
     * @param pEntity The entity requesting this resource.
     */
    public void request(final SimEntity pEntity) {
        displayInfoRequest(pEntity);
        startWaiting(pEntity);

        if (isAvailable()) {
            new SimEventStartService(this, getCurrentTime());
        }
    }

    /**
     * Ends the service of an entity.
     *
     * @param pEntity The entity that is being served by this resource.
     */
    void endServing(final SimEntity pEntity) {
        displayInfoEndService(pEntity);
        inServiceList.remove(pEntity);
        afterEndService(pEntity);
    }

    /**
     * Start serving an entity.
     *
     * @param pEntity The entity this resource is starting to serve.
     */
    void startServing(final SimEntity pEntity) {
        displayInfoStartService(pEntity);
        waitingQueue.remove(pEntity);
        inServiceList.add(pEntity);
    }

    /**
     * Do whatever is appropriate after the end of the service of an entity.
     * By default it does nothing but this is a hook operation and may be
     * overriden by subclasses (for example, "send" the entity to another
     * resource).
     *
     * @param pEntity The entity that has been served by this resource.
     */
    protected void afterEndService(final SimEntity pEntity) {}

    /**
     * Logs default information about the end of service of an entity by this
     * resource.
     * This is a hook operation and may be overridden by subclasses.
     *
     * @param pEntity The entity that has been served by this resource.
     */
    protected void displayInfoEndService(final SimEntity pEntity) {
        StringBuffer message = new StringBuffer(64);

        message.append("The temp entity ");
        message.append(pEntity.getId());
        message.append(" is relinquished by the ");
        message.append(this.getResourceName());
        message.append(" at time ");
        message.append(getCurrentTime());
        uniqueLogger.fine(message.toString());
    }

    /**
     * Logs default information about an entity requesting this resource.
     * This is a hook operation and may be overridden by subclasses.
     *
     * @param pEntity The entity that is requesting this resource.
     */
    protected void displayInfoRequest(final SimEntity pEntity) {
        StringBuffer message = new StringBuffer(64);

        message.append("The temp entity ");
        message.append(pEntity.getId());
        message.append(" requests the ");
        message.append(this.getResourceName());
        message.append(" at time ");
        message.append(getCurrentTime());
        uniqueLogger.fine(message.toString());
    }

    /**
     * Logs default information about the start of service of an entity by
     * this resource.
     * This is a hook operation and may be overridden by subclasses.
     *
     * @param pEntity The entity beginning to be served by this resource.
     */
    protected void displayInfoStartService(final SimEntity pEntity) {
        StringBuffer message = new StringBuffer(64);

        message.append("The temp entity ");
        message.append(pEntity.getId());
        message.append(" starts being served by the ");
        message.append(this.getResourceName());
        message.append(" at time ");
        message.append(getCurrentTime());
        uniqueLogger.fine(message.toString());
    }

    /**
     * Logs default information each time an entity starts waiting in
     * front of this resource.
     * This is a hook operation and may be overridden by subclasses.
     *
     * @param pEntity The entity that starts waiting to be served by
     * this resource.
     */
    protected void displayInfoStartWaiting(final SimEntity pEntity) {
        StringBuffer message = new StringBuffer(64);

        message.append("The temp entity ");
        message.append(pEntity.getId());
        message.append(" starts waiting at time ");
        message.append(getCurrentTime());
        uniqueLogger.fine(message.toString());
    }

    private void startWaiting(final SimEntity pEntity) {
        displayInfoStartWaiting(pEntity);
        waitingQueue.add(pEntity);
    }

    /**
     * Returns the next entity to serve.
     *
     * @return The next entity to serve.
     */
    public SimEntity getNextEntityToServe() {
        return waitingQueue.get(0);
    }

    /**
```

```java
     * Returns how much time is needed to serve the entity.
     * This is a primitive operation, and each concrete resource has to
     * implement it appropriately.
     *
     * @param pEntity The entity this resource will serve.
     *
     * @return The duration the entity will spend in this resource.
     */
    protected abstract double getServiceTime(SimEntity pEntity);

    /**
     * Returns the time at which this resource will relinquish the entity.
     *
     * @param pEntity The entity this resource will serve.
     *
     * @return The time at which the entity will have been finished serving.
     */
    public double getEndServiceTime(SimEntity pEntity) {
        return getCurrentTime() + getServiceTime(pEntity);
    }

    /**
     * Returns the current time of the simulation.
     * This commoditiy method is useful for subclasses.
     *
     * @return The current time of the simulation.
     */
    protected final double getCurrentTime() {
        return uniqueScheduler.getCurrentTime();
    }

    /**
     * Returns the name of this resource.
     *
     * @return The name of this resource.
     */
    protected String getResourceName() {
        return resourceName;
    }

    /**
     * Tells if there are entities waiting to be served by this resource or not.
     *
     * @return <code>true</code> if there are entities waiting to be served;
     *         <code>false</code> otherwise.
     */
     boolean hasEntitiesWaitingToBeServed() {
        return !waitingQueue.isEmpty();
    }

    /**
     * Tells if this resource is available or not.
     *
     * @return <code>true</code> if the resource is available;
     *         <code>false</code> otherwise.
     */
    private boolean isAvailable() {
        return numberOfEntitiesWaitingToBeServed()
                <= capacity - numberOfEntitiesBeingServed();
    }
}
```

2.11 simj.SimScheduler

```java
/* SimJ - A framework for discrete event simulation.
 * @(#)SimScheduler.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.TreeMap;
```

```java
/**
 * This class maintains the current time and has features for inserting new
 * events or for finding the next event in the event list or future event
 * chain (FETCH).
 *
 * @version 1.0
 * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
 */
public final class SimScheduler {

    private static SimScheduler instance                                     =
        new SimScheduler();
    private static final long serialVersionUID                               =
        3834030242716397880L;
    private static TreeMap<Double, ArrayList<SimEvent>> futureEventChain =
        new TreeMap<Double, ArrayList<SimEvent>>();
    private double currentTime;

    /**
     * Private constructor in order to avoid direct instantiation of this
     * singleton class.
     */
    private SimScheduler() {}

    /**
     * Empties the scheduler completely.
     *
     */
    public void empty() {
        futureEventChain.clear();
    }

    /**
     * Insert a given event in the future event chain.
     *
     * @param pSimEvent The event to add to the future event chain.
     */
    public void insertEvent(final SimEvent pSimEvent) {
        Double key                    = Double.valueOf(pSimEvent.getFiringTime());
        ArrayList<SimEvent> events = futureEventChain.get(key);

        if (events == null) {
            events = new ArrayList<SimEvent>();
        }

        events.add(pSimEvent);
        futureEventChain.put(key, events);
    }

    /**
     * Returns the current time of the simulation.
     *
     * @return The current time of the simulation.
     */
    public double getCurrentTime() {
        return this.currentTime;
    }

    /**
     * Returns the unique instance of this Singleton class.
     *
     * @return The unique instance of this class.
     */
    public static SimScheduler getUniqueInstance() {
        return instance;
    }

    /**
     * Returns the next event that has to be executed.
     *
     * @return The next scheduled event.
     */
    public SimEvent getNextEvent() {
        Double key                          = futureEventChain.firstKey();
        ArrayList<SimEvent> theNextEvents = futureEventChain.remove(key);
        SimEvent theNextEvent               = theNextEvents.remove(0);

        if (!theNextEvents.isEmpty()) {
            futureEventChain.put(key, theNextEvents);
        }

        this.currentTime = theNextEvent.getFiringTime();

        return theNextEvent;
    }

    /**
     * Tells if the scheduler is empty or not.
     *
     * @return <code>true</code> if there are no more events in the scheduler;
     *         <code>false</code> otherwise.
     */
    public boolean isEmpty() {
        return futureEventChain.isEmpty();
    }
}
```

2.12 simj.SimSimulation

```java
/* 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 subclaccess to access it.
     */
    protected static SimSimulation instance;
    private SimScheduler uniqueScheduler = SimScheduler.getUniqueInstance();
    private SimLogger simjLogger = SimLogger.getUniqueInstance();
    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);
    }

    /**
```

```java
     * Initializes and starts the simulation.
     * This is a template method and calls several primitive and hook operations.
     */
    public final void startSimulation() {
        createResources();
        createEvents();
        setupSimulation();
        SimEntityFactory.getUniqueInstance().reset();
        new SimEventScanResources(this.scanInterval);
        this.eventLoopManager();
    }

    /**
     * Adds a resource to this simulation.
     *
     * @param pRes The resource to add to this simulation.
     */
    void registerResource(final SimResource pRes) {
        resources.add(pRes);
    }

    /**
     * Creates the bootstrapping events of the simulation.
     * This is a primitive operation and has to be implemented by subclasses.
     */
    protected abstract void createEvents();

    /**
     * Creates the several resources of the simulation.
     * This is a primitive operation and has to be implemented by subclasses.
     */
    protected abstract void createResources();

    /**
     * Does some additional simulation initialization configuration.
     * This is a hook operation and may be overridden by subclasses.
     * By default it does nothing.
     */
    protected void setupSimulation() {
    }

    private void eventLoopManager() {
        SimEvent currentEvent;

        for (currentEvent = uniqueScheduler.getNextEvent();
                !isSimulationFinished();
                currentEvent = uniqueScheduler.getNextEvent()) {
            currentEvent.execute();
        }

        // The end of simulation time is either the firing time of the
        // first not executed event (if the simulation time elapsed)
        // or the firing time of the last executed event (if the
        // scheduler has been emptied)
        uniqueLogger.info("End of simulation at time " + SimScheduler.getUniqueInstance().getCurrentTime() + ".");

        // just in case there are still event in the future event chain
        // or scheduler.
        uniqueScheduler.empty();
    }

    /**
     * Returns the unique instance of this singleton class.
     *
     * @return The unique instance of this singleton class.
     */
    public static SimSimulation getInstance() {
        return instance;
    }

    /**
     * Returns the number of resources of the simulation.
     *
     * @return The number of resources of the simulation.
     */
    public int getNumberOfResources() {
        return resources.size();
    }

    /**
     * Returns the resource at the specified index.
     *
     * @param index An index into the resources of this simulation
     * (between <code>1</code> and <code>getNumberOfResources</code>).
     *
     * @return The resource at the specified index.
     */
    public SimResource getResource(final int index) {
        return resources.get(index - 1);
    }

    private boolean isSimulationFinished() {
        return (uniqueScheduler.isEmpty() || (uniqueScheduler.getCurrentTime() > this.finalEventTime));
    }
}
```

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);
```

```java
        buf.append(getLogEntry(record));

        return buf.toString();
    }

    /**
     * Returns the header string for the HTML file containing the formatted
     * records of this logger.
     * This method is called just after the handler using this formatter is
     * created.
     * This method overrides the empty
     * <code>java.util.logging.Formatter.getHead</code> method.
     *
     * @param handler The target handler.
     *
     * @return The header string for the HTML file containing the formatted
     * records of this logger.
     */
    public String getHead(final Handler handler) {
        String beginTime       = getTime();
        String cssDeclarations =
            "<style type=\"text/css\">\n" + "BODY {\n"
            + "FONT-FAMILY: Verdana, Arial, Helvetica, sans-serif;\n}\n"
            + "TABLE {\n    FONT-SIZE: 75%;\n}\n</style>";
        String htmlHead = "<HEAD>\n<TITLE>SimJ Log: " + loggerName
                          + "</TITLE>\n" + cssDeclarations + "\n</HEAD>";
        String htmlTitle = "<H2>" + loggerName + "<BR>\nLog started: "
                           + beginTime + "</H2>";

        return "<HTML>\n" + htmlHead + "\n<BODY>\n" + htmlTitle
               + "\n<table width=\"100%\" border=\"0\">\n";
    }

    /**
     * Returns the tail string for the HTML file containing the formatted
     * records of this logger.
     * This method is called just after the handler using this formatter is
     * closed.
     * This method overrides the empty
     * <code>java.util.logging.Formatter.getTail</code> method.
     *
     * @param handler The target handler.
     *
     * @return The tail string for the HTML file containing the formatted
     * records of this logger.
     */
    public String getTail(final Handler handler) {
        String endTime = getTime();
        String endLog  = "<H2>Log finished: " + endTime + "</H2>";

        return "</TABLE>\n" + endLog + "\n</BODY>\n</HTML>\n";
    }

    private String getLogEntry(final LogRecord record) {
        StringBuffer buf = new StringBuffer(2048);

        buf.append("<TR>\n");
        buf.append("<TD><DIV ALIGN=\"right\">");
        buf.append(record.getSequenceNumber());
        buf.append("</DIV></TD>");
        buf.append("<TD>");

        // Bold any levels >= INFO, for instance severe error messages.
        if (record.getLevel().intValue() >= Level.INFO.intValue()) {
            buf.append("<b>");

            if (record.getLevel().intValue() >= Level.WARNING.intValue()) {
                buf.append("<i>");
                buf.append(record.getLevel());
                buf.append("</i>");
            } else {
                buf.append(record.getLevel());
            }

            buf.append("</b>");
        } else {
            buf.append(record.getLevel());
        }

        buf.append("</TD>");
        buf.append("<TD>");
        buf.append(record.getMillis());
        buf.append("</TD>");
        buf.append("<TD>");
        buf.append(record.getSourceClassName());
        buf.append("</TD>");
        buf.append("<TD>");
        buf.append(record.getSourceMethodName());
        buf.append("</TD>");
        buf.append("<TD>");
        buf.append("\n</TR>\n");
        buf.append("<TR>\n");
        buf.append("<TD>");
        buf.append("</TD>");
        buf.append("<TD COLSPAN=\"4\">");
        buf.append("<EM>");
        buf.append(cleanUpHTML(formatMessage(record)));
        buf.append("</EM>");
        buf.append("</TD>");
        buf.append("\n</TR>\n");
        buf.append("<TR>");
        buf.append("<TD COLSPAN=\"5\"><HR SIZE=\"1\" NOSHADE></TD>");
        buf.append("</TR>\n\n");

        return buf.toString();
    }

    private String cleanUpHTML(String message) {
        return StringEscapeUtils.escapeHtml(message).replaceAll("\n", "<BR>\n");
    }

    private String getTime() {
        Date now = Calendar.getInstance().getTime();

        return new SimpleDateFormat("dd/MM/yyyy").format(now) + " at "
               + new SimpleDateFormat("HH:MM:ss").format(now);
    }
}
```

4.2 simj.util.logging.SimJFormatter

```java
/* SimJ - A framework for discrete event simulation.
 * @(#)SimJFormatter.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.util.logging.Formatter;
import java.util.logging.LogRecord;

/**
* This class implements a simple text formatter for log records.
* The output is usually sent to the console.
*
* @version 1.0
* @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
*/
public class SimJFormatter extends Formatter {

    /**
     * Formats a given log record in a very simple way.
     *
     * @param record The log record to format.
     *
     * @return A simple string representing the formatted log record.
     */
    public String format(final LogRecord record) {
        return record.getMessage() + '\n';
    }
}
```

5 Package supermarket

5.1 supermarket.Caisse

```java
/* 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());
    }

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

        message.append("Debut 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 displayInfoStartWaiting(final SimEntity pEntity) {
        StringBuffer message = new StringBuffer(64);

        message.append("Le client ");
        message.append(pEntity.getId());
        message.append(" commence Ãă attendre devant la ");
        message.append(this.getResourceName());
        message.append(" au temps ");
        message.append(getCurrentTime());
        uniqueLogger.fine(message.toString());
    }

    /**
     * Returns how much time is needed to serve a client by this cash desk.
     * Here, it depens on a random average time to handle an article and the
     * number of articles the current client has.
     *
     * @param pEntity The client this cash desk will serve.
     *
     * @return The time the client will spend at this cash desk.
     */
    protected double getServiceTime(final SimEntity pEntity) {
        Client pClient = (Client) pEntity;

        return uniqueRandomizer.uniform(minServiceTime, maxServiceTime) * pClient.getNbArticles();
    }
}
```

5.2 supermarket.Client

```java
/* SimJ - A framework for discrete event simulation.
 * @(#)Client.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 simj.SimEntity;
import simj.SimRandom;

/**
 * This client implements a client of the supermarket.
 *
 * @version 1.0
 * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
 */
public class Client extends SimEntity {

    /** A reference to the unique Randomizer instance. */
    private SimRandom uniqueRandomizer = SimRandom.getUniqueInstance();

    private static int maxItems;
    private final int nbItems;

    /**
     * Constructs a new client.
     *
     * @param pNewId The unique ID of this client.
     */
    public Client(final int pNewId) {
        super(pNewId);
        nbItems = Double.valueOf(uniqueRandomizer.uniform(1.0,
                getMaxArticles() + 0.5)).intValue();
    }

    /**
     * Returns the maximun number of items a client can buy.
     *
     * @return The maximum number of item a client can buy.
     */
    public int getMaxArticles() {
        return maxItems;
    }

    /**
     * Returns the actual number of articles this client has bought.
     *
     * @return The actual number of articles this client has bought.
     */
    public int getNbArticles() {
        return this.nbItems;
    }

    /**
     * Sets the maximum number of articles a client can buy.
     *
     * @param pMaxArticles The maximum number of articles a client can buy.
     */
    static void setMaxItems(final int pMaxItems) {
        maxItems = pMaxItems;
    }
}
```

5.3 supermarket.ClientFactory

```java
/* SimJ - A framework for discrete event simulation.
 * @(#)ClientFactory.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 simj.SimEntity;
import simj.SimEntityFactory;

/**
* This class implements a client factory.
*
* @version 1.0
* @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
*/
public class ClientFactory extends SimEntityFactory {

    private static ClientFactory instance = new ClientFactory();

    /**
     * Private constructor to avoid external instantiation of this singleton class.
     */
    private ClientFactory() {}

    /**
     * Overrides <code>SimEntityFactory.doCreateSimEntity</code> method
     * to create supermarket clients instead of generic entities.
     *
     * @param pId The unique ID the client will get.
     *
     * @return A new client with an unique ID.
     */
    @Override
    protected SimEntity doCreateSimEntity(final int pId) {
        return new Client(pId);
    }

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

5.4 supermarket.EvenementNouveauClient

```java
/* SimJ - A framework for discrete event simulation.
 * @(#)EvenementNouveauClient.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 simj.SimEvent;
import simj.SimRandom;
import simj.SimSimulation;

/**
 * This class implements an event which will request the shopping area
 * resource and reschedule this event after a random amount of time.
 *
 * @version 1.0
 * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
 */
public class EvenementNouveauClient extends SimEvent {

    private double interArrivalTime;

    private SuperMarche uniqueSimulation = (SuperMarche)SimSimulation.getInstance();

    /** A reference to the unique Randomizer instance. */
    private SimRandom uniqueRandomizer = SimRandom.getUniqueInstance();

    /**
     * Constructs an event for the arrival of a new client of the supermarket.
     *
     * @param pInterArrivalTime The average time between two client arrivals.
     * @param pFiringTime The firing or execution time of this event.
     */
    EvenementNouveauClient(final double pInterArrivalTime,
                           final double pFiringTime) {
        super(pFiringTime);
        interArrivalTime = pInterArrivalTime;
    }

    /**
     * Does the actual work of this event.
     * That is, send a new client to the shopping area and reschedule this event.
     *
     */
    public void execute() {
        Client entity    =
            (Client) ClientFactory.getUniqueInstance().createSimEntity();
        Magasin magasin = uniqueSimulation.getMagasin();

        magasin.request(entity);
        this.schedule(getNextArrivalTime());
    }

    private double getNextArrivalTime() {
        return getCurrentTime() + uniqueRandomizer.expo(interArrivalTime);
    }
}
```

5.5 supermarket.Magasin

```java
/* SimJ - A framework for discrete event simulation.
 * @(#)Magasin.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;
import simj.SimSimulation;

/**
 * This class implements the supermarket's shopping area resource.
 *
 * @version 1.0
 * @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
 */
public class Magasin extends SimResource {

    private Logger uniqueLogger = SimLogger.getUniqueLogger();
    private SuperMarche uniqueSimulation = (SuperMarche) SimSimulation.getInstance();

    private double maxAchatTime;
    private double minAchatTime;

    /**
     * Constructs a shopping area resource.
     *
     *
     * @param pCapacity
     * @param pResourceName
     * @param pMinAchatTime
     * @param pMaxAchatTime
     */
    public Magasin(final int pCapacity, final String pResourceName,
            final double pMinAchatTime, final double pMaxAchatTime) {
        super(pCapacity, pResourceName);
        minAchatTime = pMinAchatTime;
        maxAchatTime = pMaxAchatTime;
    }

    /**
     * Sends the client who just finished its shopping to the cash desk with
     * the shorter waiting queue.
     *
     * @param pEntity The client that has just finished its shopping.
     */
    @Override
    protected void afterEndService(final SimEntity pEntity) {
        Caisse caisse1 = uniqueSimulation.getCaisse1();
        Caisse caisse2 = uniqueSimulation.getCaisse2();

        if ((caisse2.numberOfEntitiesWaitingToBeServed() + caisse2.numberOfEntitiesBeingServed()) < (caisse1.
            numberOfEntitiesWaitingToBeServed() + caisse1.numberOfEntitiesBeingServed())) {
            caisse2.request(pEntity);
        } else {
            caisse1.request(pEntity);
        }
    }

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

        message.append("Fin d'achats du client ");
        message.append(pEntity.getId());
        message.append(" au temps ");
        message.append(getCurrentTime());
        uniqueLogger.fine(message.toString());
    }

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

        message.append("Le client ");
        message.append(pEntity.getId());
        message.append(" demande le ");
        message.append(this.getResourceName());
        message.append(" au temps ");
        message.append(getCurrentTime());
        uniqueLogger.fine(message.toString());
    }

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

        message.append("Entree du client ");
        message.append(pEntity.getId());
        message.append(" dans le ");
        message.append(this.getResourceName());
        message.append(" au temps ");
        message.append(getCurrentTime());
        uniqueLogger.fine(message.toString());
    }

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

        message.append("Le client ");
        message.append(pEntity.getId());
        message.append(" commence Ãă attendre au temps ");
        message.append(getCurrentTime());
        uniqueLogger.fine(message.toString());
    }

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

5.6 supermarket.SuperMarche

```
/* SimJ - A framework for discrete event simulation.
 * @(#)SuperMarche.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 simj.SimRandom;
import simj.SimSimulation;

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

```java
    private int maxNbItemsOfClient;
    private double tempsAchatMax;
    private double tempsAchatMin;
    private double tempsMoyenArrivee;
    private double tempsServiceMax;
    private double tempsServiceMin;

    /**
     * Constructs an instance of supermarket simulation.
     *
     * @param pTempsMoyenArrivee
     * @param pTempsAchatMin
     * @param pTempsAchatMax
     * @param pTempsServiceMin
     * @param pTempsServiceMax
     * @param pFinalEventTime
     * @param pMaxNbItemsOfClient
     * @param pScanInterval
     * @param pFineLogging
     */
    private SuperMarche(final double pTempsMoyenArrivee,
                        final double pTempsAchatMin,
                        final double pTempsAchatMax,
                        final double pTempsServiceMin,
                        final double pTempsServiceMax,
                        final double pFinalEventTime,
                        final int pMaxNbItemsOfClient,
                        final int pScanInterval, final boolean pFineLogging) {
        super(pFinalEventTime, pScanInterval, pFineLogging);
        tempsMoyenArrivee  = pTempsMoyenArrivee;
        tempsAchatMin      = pTempsAchatMin;
        tempsAchatMax      = pTempsAchatMax;
        tempsServiceMin    = pTempsServiceMin;
        tempsServiceMax    = pTempsServiceMax;
        maxNbItemsOfClient = pMaxNbItemsOfClient;
    }

    /**
     * Creates the event for the arrival of the first supermarket's client.
     */
    protected void createEvents() {

        // Evenement pour la premiere arrivee d'un client
        new EvenementNouveauClient(tempsMoyenArrivee,
                        SimRandom.getUniqueInstance().expo(tempsMoyenArrivee));
    }

    /**
     * Creates the resources of the supermarket simulation. That is, a
     * shopping area and two cash desks.
     */
    protected void createResources() {

        // Creation: Magasin - resource no. 1 (1000 = capacite du magasin)
        new Magasin(10000, "Magasin", tempsAchatMin, tempsAchatMax);

        // Creation: Caisse 1 - resource no. 2 (1 = capacite de la caisse)
        new Caisse(1, "Caisse 1", tempsServiceMin, tempsServiceMax);

        // Creation: Caisse 2 - resource no. 3 (1 = capacite de la caisse)
        new Caisse(1, "Caisse 2", tempsServiceMin, tempsServiceMax);
    }

    /**
     * Does some additional initialisation configurations for the Supermarket.
     */
    @Override
    protected void setupSimulation() {

        // Parametres des clients
        Client.setMaxItems(this.maxNbItemsOfClient);
        ClientFactory.getUniqueInstance().reset();
    }

    /**
     * Returns the first cash desk resource.
     *
     * @return The first Caisse (cash desk) resource.
     */
    public Caisse getCaisse1() {
        return (Caisse) getResource(2);
    }

    /**
     * Returns the second cash desk resource.
     *
     * @return The second Caisse (cash desk) resource.
     */
    public Caisse getCaisse2() {
        return (Caisse) getResource(3);
    }

    /**
     * Returns a new instance of the supermarket simulation: initialised and
     * ready to run.
     *
     * @param pTempsMoyenArrivee
     * @param pTempsAchatMin
     * @param pTempsAchatMax
     * @param pTempsServiceMin
     * @param pTempsServiceMax
     * @param pFinalEventTime
     * @param pMaxNbArticlesOfClient
     * @param pScanInterval
     * @param pFineLogging
     *
     * @return A ready to run supermarket simulation instance.
     */
    public static SimSimulation getInstance(final double pTempsMoyenArrivee,
            final double pTempsAchatMin, final double pTempsAchatMax,
            final double pTempsServiceMin, final double pTempsServiceMax,
            final double pFinalEventTime, final int pMaxNbArticlesOfClient,
            final int pScanInterval, final boolean pFineLogging) {

        // if (instance == null) {
        instance = new SuperMarche(pTempsMoyenArrivee, pTempsAchatMin,
                                   pTempsAchatMax, pTempsServiceMin,
                                   pTempsServiceMax, pFinalEventTime,
                                   pMaxNbArticlesOfClient, pScanInterval,
                                   pFineLogging);

        // }
        return instance;
    }

    /**
     * Returns the shopping area resource.
     *
     * @return The resource Magasin (shopping area).
     */
    public Magasin getMagasin() {
        return (Magasin) getResource(1);
    }
}
```

5.7 supermarket.SuperMarcheFrame

```
/* SimJ - A framework for discrete event simulation.
 * @(#)SuperMarcheFrame.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;

/**
* This class implements a GUI for entering the several parameters needed
* to simulate a supermarket.
*
* @version 1.0
* @author <a href="mailto:patrik.fuhrer@unifr.ch">Patrik Fuhrer</a>
*/
public class SuperMarcheFrame extends javax.swing.JFrame {

```

```java
    private static final long serialVersionUID = 4050477932868875571L;
    private javax.swing.JCheckBox fineLogging;
    private javax.swing.JLabel dureeLabel;
    private javax.swing.JTextField dureeSimulation;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JLabel maxNbArticesPerClient;
    private javax.swing.JTextField maxNbArticlesOfClients;
    private javax.swing.JTextField scanInterval;
    private javax.swing.JLabel scanIntervalLabel;
    private javax.swing.JButton startButton;
    private javax.swing.JTextField tempsAchatMax;
    private javax.swing.JLabel tempsAchatMaxLabel;
    private javax.swing.JTextField tempsAchatMin;
    private javax.swing.JTextField tempsMoyenArrivee;
    private javax.swing.JTextField tempsServiceMax;
    private javax.swing.JLabel tempsServiceMaxLabel;
    private javax.swing.JTextField tempsServiceMin;
    private javax.swing.JLabel tempsServiceMinLabel;
    private javax.swing.JLabel titleLabel;
    private javax.swing.JLabel tmpsAchatMinLabel;
    private javax.swing.JLabel tmpsmoyenLabel;

    /** Creates new form SupermarcheFrame. */
    public SuperMarcheFrame() {
        initComponents();
    }

    /**
     * The main method of this class: it displays the GUI for the supermarket
     * simulation parametrization and creation.
     *
     * @param args A string array with the command line arguments (here they
     * are not used).
     */
    public static void main(String[] args) {
        new SuperMarcheFrame().setVisible(true);
    }

    /** Exit the Application. */
    private void exitForm() {
        System.exit(0);
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    private void initComponents() {
        jPanel1                = new javax.swing.JPanel();
        titleLabel             = new javax.swing.JLabel();
        tmpsmoyenLabel         = new javax.swing.JLabel();
        tempsMoyenArrivee      = new javax.swing.JTextField();
        tmpsAchatMinLabel      = new javax.swing.JLabel();
        tempsAchatMin          = new javax.swing.JTextField();
        tempsAchatMaxLabel     = new javax.swing.JLabel();
        tempsAchatMax          = new javax.swing.JTextField();
        tempsServiceMinLabel   = new javax.swing.JLabel();
        tempsServiceMin        = new javax.swing.JTextField();
        tempsServiceMaxLabel   = new javax.swing.JLabel();
        tempsServiceMax        = new javax.swing.JTextField();
        fineLogging                    = new javax.swing.JCheckBox();
        startButton            = new javax.swing.JButton();
        maxNbArticesPerClient  = new javax.swing.JLabel();
        dureeLabel             = new javax.swing.JLabel();
        dureeSimulation        = new javax.swing.JTextField();
        scanIntervalLabel      = new javax.swing.JLabel();
        scanInterval           = new javax.swing.JTextField();
        maxNbArticlesOfClients = new javax.swing.JTextField();
        getContentPane().setLayout(new java.awt.GridBagLayout());

        java.awt.GridBagConstraints gridBagConstraints1;

        setTitle("Application of SimJ");
        setName("MainSuperMarche");
        setResizable(false);
        addWindowListener(new java.awt.event.WindowAdapter() {

            public void windowClosing(final java.awt.event.WindowEvent evt) {
                exitForm();
            }
        });
        jPanel1.setLayout(new java.awt.GridBagLayout());

        java.awt.GridBagConstraints gridBagConstraints2;

        jPanel1.setPreferredSize(new java.awt.Dimension(500, 400));
        jPanel1.setMinimumSize(new java.awt.Dimension(500, 400));
        titleLabel.setText("Supermarche - Simulation");
        titleLabel.setForeground(new java.awt.Color(51, 51, 255));
        titleLabel.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
        titleLabel.setFont(new java.awt.Font("Arial", 1, 24));
        gridBagConstraints2           = new java.awt.GridBagConstraints();
        gridBagConstraints2.gridx     = 0;
        gridBagConstraints2.gridy     = 0;
        gridBagConstraints2.gridwidth = 2;
        gridBagConstraints2.fill      = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints2.weightx   = 1.0;
        gridBagConstraints2.weighty   = 0.1;
        jPanel1.add(titleLabel, gridBagConstraints2);
        tmpsmoyenLabel.setText("Temps moyen entre deux arriv\u00e9es : ");
        tmpsmoyenLabel.setHorizontalAlignment(javax.swing.SwingConstants.LEFT);
        gridBagConstraints2         = new java.awt.GridBagConstraints();
        gridBagConstraints2.gridx   = 0;
        gridBagConstraints2.gridy   = 1;
        gridBagConstraints2.fill    = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints2.weightx = 0.8;
        gridBagConstraints2.weighty = 0.09;
        jPanel1.add(tmpsmoyenLabel, gridBagConstraints2);
        tempsMoyenArrivee.setColumns(5);
        tempsMoyenArrivee.setText("100");
        gridBagConstraints2         = new java.awt.GridBagConstraints();
        gridBagConstraints2.gridx   = 1;
        gridBagConstraints2.gridy   = 1;
        gridBagConstraints2.fill    = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints2.weightx = 0.2;
        gridBagConstraints2.weighty = 0.09;
        jPanel1.add(tempsMoyenArrivee, gridBagConstraints2);
        tmpsAchatMinLabel.setText("Temps d'achat minimal : ");
        tmpsAchatMinLabel.setHorizontalAlignment(
            javax.swing.SwingConstants.LEFT);
        gridBagConstraints2         = new java.awt.GridBagConstraints();
        gridBagConstraints2.gridx   = 0;
        gridBagConstraints2.gridy   = 2;
        gridBagConstraints2.fill    = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints2.weightx = 0.8;
        gridBagConstraints2.weighty = 0.09;
        jPanel1.add(tmpsAchatMinLabel, gridBagConstraints2);
        tempsAchatMin.setColumns(5);
        tempsAchatMin.setText("300");
        gridBagConstraints2         = new java.awt.GridBagConstraints();
        gridBagConstraints2.gridx   = 1;
        gridBagConstraints2.gridy   = 2;
        gridBagConstraints2.fill    = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints2.weightx = 0.2;
        gridBagConstraints2.weighty = 0.09;
        jPanel1.add(tempsAchatMin, gridBagConstraints2);
        tempsAchatMaxLabel.setText("Temps d'achat maximal : ");
        tempsAchatMaxLabel.setHorizontalAlignment(
            javax.swing.SwingConstants.LEFT);
        gridBagConstraints2         = new java.awt.GridBagConstraints();
        gridBagConstraints2.gridx   = 0;
        gridBagConstraints2.gridy   = 3;
        gridBagConstraints2.fill    = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints2.weightx = 0.8;
        gridBagConstraints2.weighty = 0.09;
        jPanel1.add(tempsAchatMaxLabel, gridBagConstraints2);
        tempsAchatMax.setColumns(5);
        tempsAchatMax.setText("800");
        gridBagConstraints2         = new java.awt.GridBagConstraints();
        gridBagConstraints2.gridx   = 1;
        gridBagConstraints2.gridy   = 3;
        gridBagConstraints2.fill    = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints2.weightx = 0.2;
        gridBagConstraints2.weighty = 0.09;
        jPanel1.add(tempsAchatMax, gridBagConstraints2);
        tempsServiceMinLabel.setText(
            "Temps minimal de service pour un article : ");
        tempsServiceMinLabel.setHorizontalAlignment(
            javax.swing.SwingConstants.LEFT);
        gridBagConstraints2         = new java.awt.GridBagConstraints();
        gridBagConstraints2.gridx   = 0;
        gridBagConstraints2.gridy   = 4;
        gridBagConstraints2.fill     = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints2.weightx  = 0.8;
        gridBagConstraints2.weighty  = 0.09;
        jPanel1.add(tempsServiceMinLabel, gridBagConstraints2);
        tempsServiceMin.setColumns(5);
        tempsServiceMin.setText("20");
        gridBagConstraints2          = new java.awt.GridBagConstraints();
        gridBagConstraints2.gridx    = 1;
        gridBagConstraints2.gridy    = 4;
        gridBagConstraints2.fill     = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints2.weightx  = 0.2;
        gridBagConstraints2.weighty  = 0.09;
        jPanel1.add(tempsServiceMin, gridBagConstraints2);
        tempsServiceMaxLabel.setText(
            "Temps maximal de service pour un article : ");
        tempsServiceMaxLabel.setHorizontalAlignment(
            javax.swing.SwingConstants.LEFT);
        gridBagConstraints2          = new java.awt.GridBagConstraints();
        gridBagConstraints2.gridx    = 0;
        gridBagConstraints2.gridy    = 5;
        gridBagConstraints2.fill     = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints2.weightx  = 0.8;
        gridBagConstraints2.weighty  = 0.09;
        jPanel1.add(tempsServiceMaxLabel, gridBagConstraints2);
        tempsServiceMax.setColumns(5);
        tempsServiceMax.setText("40");
        gridBagConstraints2          = new java.awt.GridBagConstraints();
        gridBagConstraints2.gridx    = 1;
        gridBagConstraints2.gridy    = 5;
        gridBagConstraints2.fill     = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints2.weightx  = 0.2;
        gridBagConstraints2.weighty  = 0.09;
        jPanel1.add(tempsServiceMax, gridBagConstraints2);
        fineLogging.setForeground(new java.awt.Color(255, 51, 51));
        fineLogging.setText("Fine Logging");
        gridBagConstraints2            = new java.awt.GridBagConstraints();
        gridBagConstraints2.gridx      = 0;
        gridBagConstraints2.gridy      = 9;
        gridBagConstraints2.gridwidth  = 2;
        gridBagConstraints2.fill       = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints2.weightx    = 1.0;
        gridBagConstraints2.weighty    = 0.09;
        jPanel1.add(fineLogging, gridBagConstraints2);
        startButton.setForeground(new java.awt.Color(255, 51, 51));
        startButton.setText("Start Simulation");
        startButton.setBackground(java.awt.Color.lightGray);
        startButton.addActionListener(new java.awt.event.ActionListener() {

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

        });
        gridBagConstraints2            = new java.awt.GridBagConstraints();
        gridBagConstraints2.gridx      = 0;
        gridBagConstraints2.gridy      = 10;
        gridBagConstraints2.gridwidth  = 2;
        gridBagConstraints2.fill       = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints2.weightx    = 1.0;
        gridBagConstraints2.weighty    = 0.09;
        jPanel1.add(startButton, gridBagConstraints2);
        maxNbArticesPerClient.setText("Nombre maximal d'articles par client:");
        maxNbArticesPerClient.setHorizontalAlignment(
            javax.swing.SwingConstants.LEFT);
        gridBagConstraints2          = new java.awt.GridBagConstraints();
        gridBagConstraints2.gridx    = 0;
        gridBagConstraints2.gridy    = 6;
        gridBagConstraints2.fill     = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints2.weightx  = 0.8;
        gridBagConstraints2.weighty  = 0.09;
        jPanel1.add(maxNbArticesPerClient, gridBagConstraints2);
        dureeLabel.setText("Dur\u00e9e de la simulation :");
        dureeLabel.setForeground(new java.awt.Color(255, 51, 51));
        dureeLabel.setHorizontalAlignment(javax.swing.SwingConstants.LEFT);
        gridBagConstraints2          = new java.awt.GridBagConstraints();
        gridBagConstraints2.gridx    = 0;
        gridBagConstraints2.gridy    = 7;
        gridBagConstraints2.fill     = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints2.weightx  = 0.8;
        gridBagConstraints2.weighty  = 0.09;
        jPanel1.add(dureeLabel, gridBagConstraints2);
        dureeSimulation.setColumns(5);
```

```java
        dureeSimulation.setForeground(new java.awt.Color(255, 51, 51));
        dureeSimulation.setText("80000");
        gridBagConstraints2         = new java.awt.GridBagConstraints();
        gridBagConstraints2.gridx   = 1;
        gridBagConstraints2.gridy   = 7;
        gridBagConstraints2.fill    = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints2.weightx = 0.2;
        gridBagConstraints2.weighty = 0.09;
        jPanel1.add(dureeSimulation, gridBagConstraints2);
        scanIntervalLabel.setText(
            "Intervalle entre deux \"scan\" des ressources : ");
        scanIntervalLabel.setForeground(new java.awt.Color(255, 51, 51));
        scanIntervalLabel.setHorizontalAlignment(
            javax.swing.SwingConstants.LEFT);
        gridBagConstraints2         = new java.awt.GridBagConstraints();
        gridBagConstraints2.gridx   = 0;
        gridBagConstraints2.gridy   = 8;
        gridBagConstraints2.fill    = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints2.weightx = 0.8;
        gridBagConstraints2.weighty = 0.09;
        jPanel1.add(scanIntervalLabel, gridBagConstraints2);
        scanInterval.setColumns(5);
        scanInterval.setForeground(new java.awt.Color(255, 51, 51));
        scanInterval.setText("4000");
        gridBagConstraints2         = new java.awt.GridBagConstraints();
        gridBagConstraints2.gridx   = 1;
        gridBagConstraints2.gridy   = 8;
        gridBagConstraints2.fill    = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints2.weightx = 0.2;
        gridBagConstraints2.weighty = 0.09;
        jPanel1.add(scanInterval, gridBagConstraints2);
        maxNbArticlesOfClients.setColumns(5);
        maxNbArticlesOfClients.setText("20");
        gridBagConstraints2         = new java.awt.GridBagConstraints();
        gridBagConstraints2.gridx   = 1;
        gridBagConstraints2.gridy   = 6;
        gridBagConstraints2.fill    = java.awt.GridBagConstraints.BOTH;
        gridBagConstraints2.weightx = 0.2;
        gridBagConstraints2.weighty = 0.09;
        jPanel1.add(maxNbArticlesOfClients, gridBagConstraints2);
        gridBagConstraints1         = new java.awt.GridBagConstraints();
        gridBagConstraints1.weightx = 1.0;
        gridBagConstraints1.weighty = 1.0;
        getContentPane().add(jPanel1, gridBagConstraints1);
        pack();
    }

    private void startButtonActionPerformed() {
        SuperMarche supermarche =
            (SuperMarche) SuperMarche.getInstance(
                Double.parseDouble(tempsMoyenArrivee.getText()),
                Double.parseDouble(tempsAchatMin.getText()),
                Double.parseDouble(tempsAchatMax.getText()),
                Double.parseDouble(tempsServiceMin.getText()),
                Double.parseDouble(tempsServiceMax.getText()),
                Double.parseDouble(dureeSimulation.getText()),
                Integer.parseInt(maxNbArticlesOfClients.getText()),
                Integer.parseInt(scanInterval.getText()), fineLogging.isSelected());

        supermarche.startSimulation();
    }
}
```