```
import swarm.Globals;
import swarm.defobj.Zone;
import swarm.objectbase.SwarmObjectImpl;

import swarm.collections.ListImpl;

/**
 * OrderDistiller.java
 *
 *
 * Created: Wed May 08 15:29:12 2002
 *
 * @author<br/>
 * Cristian Barreca<a href="mailto:dgbarrec@libero.it"></a><br/>
 * Elena Bonessa<a href="mailto:elena.bonessa@infinito.it"></a><br/>
 * Antonella Borra<a href="mailto:anborra@libero.it"></a>
 * @version 0.6.2.b
 */

/**
 * This class is used to read data from two worksheets. <br/>The
first one contains the
 * list of recipes of
 * our virtual enterprise.<br/>The second one contains a sequence of
orders to be launched,
 * shift by shift, in order to make the daily production activities.
*/
public class OrderDistiller extends OrderGenerator{

    // INSTANCE VARIABLES
    /**A flag to check if the orderSequences worksheet file is open
     */
    public boolean worksheetOrderSequenceFileOpen = false,
worksheetRecipeFileOpen = false;

    /**The arrays containing: the sequence of orders to be done in
the related shift and the respective quantity
     */
    int[] orderSequence1, orderSequence2;

    /**The variable referring to the orderSequences worksheet file
     */
    ExcelReader orderSequenceWorksheet = null, recipeWorksheet =
null;
    /** The name of the recipe
     */
    String recipeName;

    /**Flags to operate checks during the reading
     */
    public String semicolon = ";", checkTheCell, gate = "#", p = "p",
sec = "s", min = "m", end = "e", slash = "/", backslash = "\\", or =
"||";

    /**An object to record the array containing the steps of the
recipes in a List
     */
    Recipe aRecipe;

    /** used to record the number of generated orders
     */
    public int orderCount = 0;

    /** a specific order
```

```
     */
    public Order anOrder;

    /** the list containig the operating units
     */
    public ListImpl unitList;

    /** the list containig the end units
     */
    public ListImpl endUnitList;

    /** the list containig all the orders
     */
    public ListImpl orderList;

    /** the list containig all the orders
     */
    public ListImpl recipeList;

    /**The array containing the steps of the recipe and the steps of
procurement
     */
    int[] orderRecipe;

    // units
    Unit aUnit;
    EndUnit anEndUnit;

    public boolean unitNotFound;
    int j, row, length, choice;

    // CONSTRUCTOR
    public OrderDistiller (Zone aZone, int msn, int msl, ListImpl ul,
                      ListImpl eul, ListImpl ol, VEFrameModelSwarm mo)
{

      super(aZone, msn, msl, ul, eul, ol, mo);

      unitList=ul;
      endUnitList=eul;
        orderList=ol;

    }

    /**This method is used to collect the names of the units and of
the end units, so that it can operate the check of
     * corrispondency between the production phases required by
recipes and the phases of production the units can do.
     */
    public void setDictionary(){

      super.setDictionary();

      // It will contain the ID codes of the recipes worksheet file
      orderSequence1 = new int[dictionaryLength];
      // It will contain the quantity of each recipe to be done
      orderSequence2 = new int[dictionaryLength];
      for( int i = 0; i < dictionaryLength; i++){
          orderSequence1[i] = 0;
          orderSequence2[i] = 0;
      }

      setRecipeContainers();
      readRecipes();
```

```
    }

    /**This method is used to set the length of each recipe
     */
    public void setRecipeContainers(){

      recipeList = new ListImpl(getZone());

      if(! worksheetRecipeFileOpen){
          recipeWorksheet = new ExcelReader
("recipeData/recipes.xls");
          worksheetRecipeFileOpen = true;
      }

      checkTheCell = recipeWorksheet.getStrValue();

      row = 0;
      while(! recipeWorksheet.eof()){
          checkForComments(checkTheCell);
          calculateLength(checkTheCell);
      }

      worksheetRecipeFileOpen = false;
    }

    /** This is the method needed to read and store the recipes. The
procedure follows this steps:<br/> It opens the worksheet file
recipes.xls, in which are stored the sequences of steps of all the
recipes;<br/>It makes some check of the routines; <br/>It search the
object containig the same code of the recipe we are considering;<br/>
     * Finally it substitutes the values of the array of that object
with the new ones.
     */
    public void readRecipes(){
      //local variables
      int code = 0, i;
      boolean recipeCodeNotFound;

      if(! worksheetRecipeFileOpen){
          recipeWorksheet = new ExcelReader
("recipeData/recipes.xls");
          worksheetRecipeFileOpen = true;
      }
      row = 0;
      checkTheCell = recipeWorksheet.getStrValue();
      while(! recipeWorksheet.eof()){

          checkForComments(checkTheCell);
          code = errorIsNotAnInteger(recipeWorksheet);

          recipeCodeNotFound=true;
          for (i = 0; i < recipeList.getCount() &&
recipeCodeNotFound; i++)
            {
                aRecipe = (Recipe) recipeList.atOffset(i);
                if(aRecipe.getCodeNumber() == code)recipeCodeNotFound
= false;
            }

          aRecipe.setSteps(checkTheCell, recipeWorksheet);
          checkTheCell = aRecipe.getCheckTheCell();
      }
    }
```

```
    /**This is the method containing the iterator needed to launch
the daily production of recipes.
     * It take a look at the orderSequence arrays to determine which
recipes must be done and how many times.
     * A request for which unit can do the first production phase of
each recipe will be done to units or endUnits.
     */
    public void distill(){
      //local counters
      int i, ii, iii, k, code, quantity;
      boolean recipeCodeNotFound;
      i=0;

      readOrderSequence();
      while(i < dictionaryLength && orderSequence1[i] != 0){
          code =  getOrderSequence1(i);
          quantity = getOrderSequence2(i);

          recipeCodeNotFound = true;
          for (ii = 0; ii < recipeList.getCount() &&
recipeCodeNotFound; ii++)
            {
                aRecipe = (Recipe) recipeList.atOffset(ii);
                if(aRecipe.getCodeNumber() == code)recipeCodeNotFound
= false;
            }
          for(k = 0; k < quantity; k++){

            orderCount++;
            // creating an order
            anOrder = new Order(getZone(), orderCount,
                            Globals.env.getCurrentTime(),
                            aRecipe.getLength(),
aRecipe.getOrderRecipe(),
                            vEFrameModelSwarm, endUnitList);

            anOrder.setRecipeName(aRecipe.getRecipeName());

            // add the active orders to the general order list (they
will be
            // eliminated when dropped in a unit, being finished);
this
            // list has been introduced for accounting purposes [may
be it would
            // be better subtitute it with a get to the units to know
their
            // waiting lists]
            orderList.addLast(anOrder);

            /**
             * sending the order to the first production unit
             * (we are acting as the Front End of the VE)
             */
            unitNotFound = true;
            for (ii = 0; ii < unitList.getCount() && unitNotFound;
ii++)
                {
                  aUnit=(Unit) unitList.atOffset(ii);
                  for (iii = 0; iii < aUnit.getNOfPhasesToDealWith()
&& unitNotFound; iii++)
                      {
                        if(anOrder.getNextStep() ==
aUnit.getProductionPhase(iii))
                            {
```

```
                                unitNotFound = false;
                                aUnit.setWaitingList(anOrder);
                              }
                        }
                    }

            // considering also endUnits
            for (ii = 0; ii < endUnitList.getCount() && unitNotFound;
ii++)
                {
                  anEndUnit = (EndUnit) endUnitList.atOffset(ii);
                  if(anOrder.getNextStep() == anEndUnit.getUnitNumber
())
                        {
                          unitNotFound=false;
                          anEndUnit.setTemporaryList(anOrder);
                        }
                }

            if(unitNotFound){
                System.out.println("OrderDistiller internal error:
receiving "+
                                "unit does not exixts");
                System.exit(1);
            }


          }

          i++;
      }


    }


    /**This method reads from the worksheet containing, shift by
shift, the sequence of orders to be launched and fills
     * in the orderSequence1 with the ID codes of recipes and the
orderSequence2 with the quantities of each recipe.
     */
    public void readOrderSequence(){
      //local variables
      int i, ii,numberOfShift;

      i = 0;
      if(! worksheetOrderSequenceFileOpen){
          orderSequenceWorksheet = new ExcelReader
("recipeData/orderSequences.xls");
          worksheetOrderSequenceFileOpen = true;
      }

      numberOfShift = orderSequenceWorksheet.getIntValue();
      if(StartVEFrame.verbose)
          System.out.println("The shift #" + numberOfShift + "has
begun ");

      while(! orderSequenceWorksheet.eol()){

          orderSequence1[i] = errorIsNotAnInteger
(orderSequenceWorksheet);

          errorIsNotAString(orderSequenceWorksheet);
          orderSequenceWorksheet.getStrValue();
```

```
          orderSequence2[i] = errorIsNotAnInteger
(orderSequenceWorksheet);
          i++;
      }

      errorIsNotAString(orderSequenceWorksheet);
      orderSequenceWorksheet.getStrValue();



      if(orderSequenceWorksheet.eof()){
          //   System.out.println("The simulation finishes here. It
is not the right and normal way, but is only for explanation. Thank
you for your attention and interest.");
          // System.exit(1);
          worksheetOrderSequenceFileOpen = false;
      }
    }


    /**This method is used to check if there are comments in the
cells
     */
    public void checkForComments(String cTC){

      checkTheCell = cTC;
      while(checkTheCell.equals(gate)){
          row++;
          if(StartVEFrame.verbose)
            System.out.println("The row # " + row + " contains a
comment");

          recipeWorksheet.getStrValue();
          checkTheCell = recipeWorksheet.getStrValue();
          if(! checkTheCell.equals(semicolon))errorIsNotAString
(recipeWorksheet);
          checkTheCell = recipeWorksheet.getStrValue();
      }
    }

    /**This method is used to calculate the length of each row after
a strong check of the elements
     */
    public void calculateLength(String cTC){

      checkTheCell = cTC;
      recipeName = checkTheCell;
      code = errorIsNotAnInteger(recipeWorksheet);
      row++;
      length = 0;
      checkTheCell = recipeWorksheet.getStrValue();


      while(! checkTheCell.equals(semicolon)){
          if(checkTheCell.equals(p))   choice = 1;
          if(checkTheCell.equals(or))  choice = 2;
          if(checkTheCell.equals(end)) choice = 3;
          if(recipeWorksheet.checkForLabelCell() && !
checkTheCell.equals(semicolon))   choice = 4;

          switch(choice){
          case 1:
            if(StartVEFrame.verbose)
```

```
            System.out.println("The choice is for a Procurement");
            procurement(recipeWorksheet);
            break;
          case 2:
            if(StartVEFrame.verbose)
            System.out.println("The choice is for an Or ");
            oR(recipeWorksheet);
            break;
          case 3:
            if(StartVEFrame.verbose)
            System.out.println("The choice is for an End ");
            end(recipeWorksheet);
            break;
          case 4:
            if(StartVEFrame.verbose)
            System.out.println("The choice is for a Number");
            number(recipeWorksheet);
            break;
          default:
      if(StartVEFrame.verbose)
            System.out.println("No matches were found reading the
worksheet, check for errors inside it");
            System.exit(1);


          }
      }

      if(! recipeWorksheet.eof())
          checkTheCell = recipeWorksheet.getStrValue();



      aRecipe = new Recipe(getZone(), code, length, recipeName);
      if( StartVEFrame.verbose)
          System.out.println("OrderDistiller: a Recipe named "+
recipeName+ " with code " + code + "was born");
      recipeList.addLast(aRecipe);
    }

    /**This method is used to obtain the elements of the
orderSequence1 array
     */
    public int getOrderSequence1(int j){

      return orderSequence1[j];

    }

    /**This method is used to obtain the elements of the
orderSequence2 array
     */
    public int getOrderSequence2(int j){

      return orderSequence2[j];

    }

    /**This method is used to check the corrispondence with the
dictionary of production phases
     */
    public int checkTheExistence(int c){
      int i, check;
      boolean dictionaryVoice = false;
```

```
      check = c;

      for(i = 0; i < dictionaryLength; i++){
          if(check == dictionary[i])dictionaryVoice = true;
      }

      if(dictionaryVoice == false){
            System.out.println("The value found is not a valid number
of production");
            System.exit(1);
      }

      return check;
    }

    /**This method is used to check if a type error occurs
     */
    public int errorIsNotAnInteger(ExcelReader e){

      if(e.checkForLabelCell())
       {
           System.out.println("The cell should contain an Integer
Value");
           System.exit(1);
       }
      return e.getIntValue();
    }

    /**This method is used to check if a type error occurs
     */
    public void  errorIsNotAString(ExcelReader e){

      if(e.checkForLabelCell());
      else {
          System.out.println("The cell should contain a String
Value");
          System.exit(1);
      }
    }

    /**This method dial with the procurement choice
     */
    public void procurement(ExcelReader e){
      int numberOfStepsForProcurement = 0;

      numberOfStepsForProcurement = e.getIntValue();
      length += (2 + numberOfStepsForProcurement);
      for(int h = 0; h < numberOfStepsForProcurement; h++)
checkTheCell = e.getStrValue();
      checkTheCell = e.getStrValue();

    }

    /**This method dial with the or choice
     */
    public void oR(ExcelReader e){

      length +=2;
      checkTheCell = e.getStrValue();

    }

    /**This method dial with the end choice
```

```
     */
    public void end(ExcelReader e){

      length++;
      checkTheCell = e.getStrValue();

    }

    /**This method dial with normal or batch choice
     */
    public void number(ExcelReader e){

      checkTheCell = e.getStrValue();
      if(checkTheCell.equals(sec)) second(e);
      else if(checkTheCell.equals(min))minute(e);
      else{
          if(StartVEFrame.verbose)
          System.out.println("Time is not expressed in minutes or
seconds. Check the worksheet");
          System.exit(1);
      }
      j++;
    }


    public void second(ExcelReader e){
      int numberOfSteps = 0;

          numberOfSteps = e.getIntValue();
          checkTheCell = e.getStrValue();

          if(checkTheCell.equals(slash) || checkTheCell.equals
(backslash)){

            length +=4;
            e.getStrValue();
            checkTheCell = e.getStrValue();

          }

          else {
            length += numberOfSteps;
          }

    }

   public void minute(ExcelReader e){
      int numberOfSteps = 0;

          numberOfSteps = (e.getIntValue() * 60);
          checkTheCell = e.getStrValue();

          if(checkTheCell.equals(slash) || checkTheCell.equals
(backslash)){

            length +=4;
            e.getStrValue();
            checkTheCell = e.getStrValue();

          }

          else {
            length += numberOfSteps;
          }
```

```
    }

} // OrderDistiller




import swarm.defobj.Zone;
import swarm.objectbase.SwarmObjectImpl;

/**
 * Recipe.java
 *
 *
 * Created: Wed May  13 15:29:12 2002
 *
 * @author Cristian Barreca<a href="mailto:dgbarrec@libero.it"></a>
<br/>Elena Bonessa<a href="mailto:elena.bonessa@infinito.it"></a>
<br/>Antonella Borra<a href="mailto:anborra@libero.it"></a>
 * @version 0.6.2.a
 */

/**This class is used to record the recipes and their referring
number (ID), so we can assign them to a List*/
public class Recipe extends SwarmObjectImpl{
    //INSTANCE VARIABLES
    /**the array containing the recipe*/
    int[] orderRecipe;

    /**the referring number of the recipe in the worksheet file*/
    int code;

    /**The length of the array
     */
    int length;

    /**Flags to operate checks during the reading
     */
    public String semicolon = ";", checkTheCell, gate = "#", p = "p",
end = "e", sec = "s", min = "m", slash = "/", backslash = "\\", or =
"||";
    /** The name of the recipe
     */
    String recipeName;

    /** the dictionary to be used to choose the steps to be included
in
     *    a recipe
     */
    int[] dictionary;

    /** length of the dictionary */
    int dictionaryLength;

    OrderDistiller orderDistiller;

    int j, choice, step;
    //CONSTRUCTOR
    public Recipe(Zone aZone, int c, int l, String rN){

      super(aZone);
```

```
      code = c;
      orderRecipe = new int[l];
      for(int j = 0; j < l; j++)orderRecipe[j] = 0;
      length = l;
      recipeName = rN;
    }

    public int getCodeNumber(){

      return code;

    }

    public int[] getOrderRecipe(){

      return orderRecipe;

    }

    public int getLength(){

      return length;

    }

    public String getRecipeName(){

      return recipeName;

    }

    public void setSteps(String cTC, ExcelReader recipeWorksheet){


      checkTheCell = cTC;

      checkTheCell = recipeWorksheet.getStrValue();
      j = 0;
      if(StartVEFrame.verbose)
      System.out.println(checkTheCell +" The length of the recipe n°
" + getCodeNumber() + " is " + getLength());

      while(! checkTheCell.equals(semicolon)){
          if(checkTheCell.equals(p))   choice = 1;
          if(checkTheCell.equals(or))  choice = 2;
          if(checkTheCell.equals(end)) choice = 3;
          if(recipeWorksheet.checkForLabelCell() && !
checkTheCell.equals(semicolon))   choice = 4;

          switch(choice){
          case 1:
            if(StartVEFrame.verbose)
            System.out.println("The choice is for a Procurement");
            procurement(recipeWorksheet);
            break;
          case 2:
            if(StartVEFrame.verbose)
            System.out.println("The choice is for an Or ");
            oR(recipeWorksheet);
            break;
          case 3:
            if(StartVEFrame.verbose)
            System.out.println("The choice is for an End ");
```

```
            end(recipeWorksheet);
            break;
          case 4:
            if(StartVEFrame.verbose)
            System.out.println("The choice is for a Number");
            number(recipeWorksheet);
            break;
          default:
      if(StartVEFrame.verbose)
            System.out.println("No matches were found reading the
worksheet, check for errors inside it");
            System.exit(1);


          }
      }

      for(int h = 0; h < length; h++)
          if(StartVEFrame.verbose)
          System.out.println("The recipe contains " + orderRecipe[h]
+ " in position " + h);
      if(! recipeWorksheet.eof())
          checkTheCell = recipeWorksheet.getStrValue();

    }

    public String getCheckTheCell(){

      return checkTheCell;

    }

    /**This method is used to check if a type error occur
     */
    public int errorIsNotAnInteger(ExcelReader e){

      if(e.checkForLabelCell())
          {
            if(StartVEFrame.verbose)
            System.out.println("The cell should contain an Integer
Value");
            System.exit(1);
          }
      return e.getIntValue();
    }

    /**This method is used to check if a type error occur
     */
    public void errorIsNotAString(ExcelReader e){

      if(e.checkForLabelCell());
      {
          if(StartVEFrame.verbose)
          System.out.println("The cell should contain a String
Value");
          System.exit(1);
      }
    }
    /**This method dial with the procurement choice
     */
    public void procurement(ExcelReader e){
      int numberOfStepsForProcurement = 0;

      orderRecipe[j] = -1;
```

```
      numberOfStepsForProcurement = e.getIntValue();
      orderRecipe[++j] = numberOfStepsForProcurement;
      for(int h = 0; h < numberOfStepsForProcurement; h++)
          orderRecipe[++j] = e.getIntValue();
      j++;
      checkTheCell = e.getStrValue();

    }

    /**This method dial with the or choice
     */
    public void oR(ExcelReader e){

      orderRecipe[j] = -10;
      orderRecipe[++j] = e.getIntValue();
      j++;
      checkTheCell = e.getStrValue();

    }

    /**This method dial with the end choice
     */
    public void end(ExcelReader e){

      orderRecipe[j] = e.getIntValue();
      checkTheCell = e.getStrValue();

    }

    /**This method dial with normal or batch choice
     */
    public void number(ExcelReader e){

      step = Integer.parseInt(checkTheCell);
      checkTheCell = e.getStrValue();
      if(checkTheCell.equals(sec)) second(e, step);
      else if(checkTheCell.equals(min))minute(e, step);
      else{
          if(StartVEFrame.verbose)
          System.out.println("Time is not expressed in minutes or
seconds. Check the worksheet");
          System.exit(1);
      }
      j++;
    }


    public void second(ExcelReader e, int step){
      int numberOfSteps = 0;

          numberOfSteps = e.getIntValue();
          checkTheCell = e.getStrValue();

          if(checkTheCell.equals(slash) || checkTheCell.equals
(backslash)){

            if(checkTheCell.equals(slash)){
                orderRecipe[j] = -2;
                orderRecipe[++j] = numberOfSteps;
                orderRecipe[++j] = e.getIntValue();
                orderRecipe[++j] = step;
            }

            else if(checkTheCell.equals(backslash)){
```

```
                orderRecipe[j] = -3;
                orderRecipe[++j] = numberOfSteps;
                orderRecipe[++j] = e.getIntValue();
                orderRecipe[++j] = step;
            }
            checkTheCell = e.getStrValue();
        }

        else {
            orderRecipe[j] = step;
            for(int jj = 0; jj < numberOfSteps - 1; jj++)
                orderRecipe[++j] = step;
        }

    }

   public void minute(ExcelReader e, int step){
       int numberOfSteps = 0;

        numberOfSteps = e.getIntValue();
        numberOfSteps = numberOfSteps * 60;
        checkTheCell = e.getStrValue();

        if(checkTheCell.equals(slash) || checkTheCell.equals
(backslash)){

            if(checkTheCell.equals(slash)){
                orderRecipe[j] = -2;
                orderRecipe[++j] = numberOfSteps;
                orderRecipe[++j] = e.getIntValue();
                orderRecipe[++j] = step;
            }

            else if(checkTheCell.equals(backslash)){
                orderRecipe[j] = -3;
                orderRecipe[++j] = numberOfSteps;
                orderRecipe[++j] = e.getIntValue();
                orderRecipe[++j] = step;
            }
            checkTheCell = e.getStrValue();
        }

        else {
            orderRecipe[j] = step;
            for(int jj = 0; jj < numberOfSteps - 1; jj++)
                orderRecipe[++j] = step;
        }

    }

}//Recipe.java
```