Algorithms For Scheduling (AFS) version JAVA

Short User Guide

1 Introduction

AFS (Algorithms For Scheduling) is a Java framework intended to help the development of Lekin plug-ins. AFS provides the services to read the Lekin input files (_usr.mch and _usr.job files) which contain the machine and job information and to write schedules back (_usr.seq file) to Lekin. The input information is stored in the AFS Java ® data structures. The user can then extract and modify the machine and job information and write his/her own algorithms and to display nice Gantt charts back on Lekin.

AFS was developed for a Scheduling course at the Masters level. This course has been taught by the IE Department of the Universidad de los Andes in Colombia (South America). The first versions were developed in C++. Later the code was enhanced and converted into Java, given the popularity of this programming language.

The software was developed under the GNU GPL licenses and therefore the code can be modified as wished. The author assumes no responsibility for errors and/or the malfunction of the program.

The author would thank any reports on bugs, errors or failures. Please write to gmejia@uniandes.edu.co

2 Prerequisites.

The user of AFS should be familiar with (i) the Java language, (ii) the Lekin Scheduling Software ® and with the Lekin plug-ins. For more information on Java programming, please visit http://java.sun.com/.

Lekin can be downloaded free of charge from http://www.stern.nyu.edu/om/software/lekin/.

3 Installation:

3.1 Pre-requisites:

- Java SDK (Standard Development Kit) or JRE (Java Runtime Environment) version 1.4.2x or later. Visit http://www.java.com and obtain the appropriate version.
- Java IDE (Integrated Development Environment) such as Eclipse ®, or Borland Builder®. Visit www.eclipse.org and www.borland.com to obtain free versions of these IDEs[1].
- Lekin Scheduling System
- Microsoft Windows XP or Vista operating system.

3.2 Installing AFS

Depending on the IDE platform the procedure may vary. There are two ways of installing AFS Java: The first alternative consists of unzipping the file AFSjava.zip

[1] AFS should work on other IDEs and operating systems but has not been tested yet. The user may try to install it and test it. The author thanks any feedback on these items.

directly in the workspace (easiest but may not work on all IDEs). The second alternative consists of creating a project from scratch and manually put the files in there. The steps for the two alternatives are described below:

3.2.1 Importing the AFS Project

This procedure shows the steps provided you are running the Eclipse ® IDE.
Go to **File/Import/General** and select **"Existing Projects into Workspace"** as shown in figure 1.
Click on the **"Next"** button.

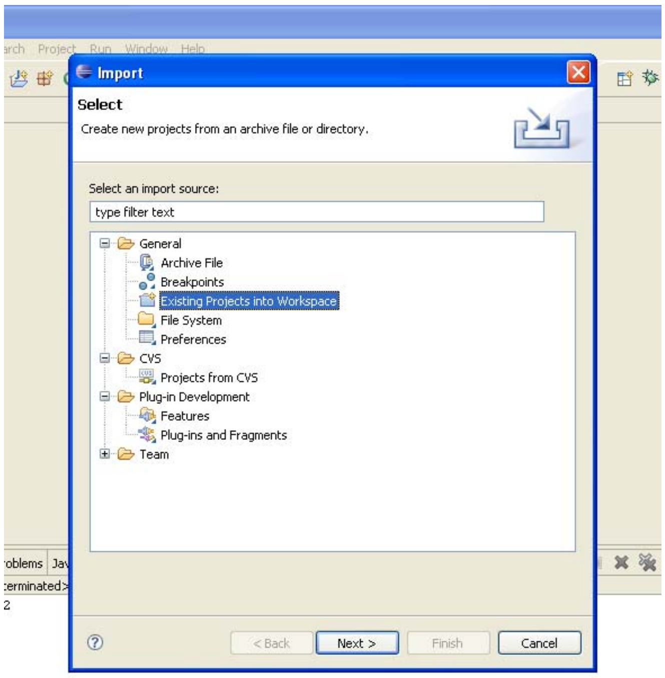

Import
Select
Create new projects from an archive file or directory.
Select an import source:
type filter text

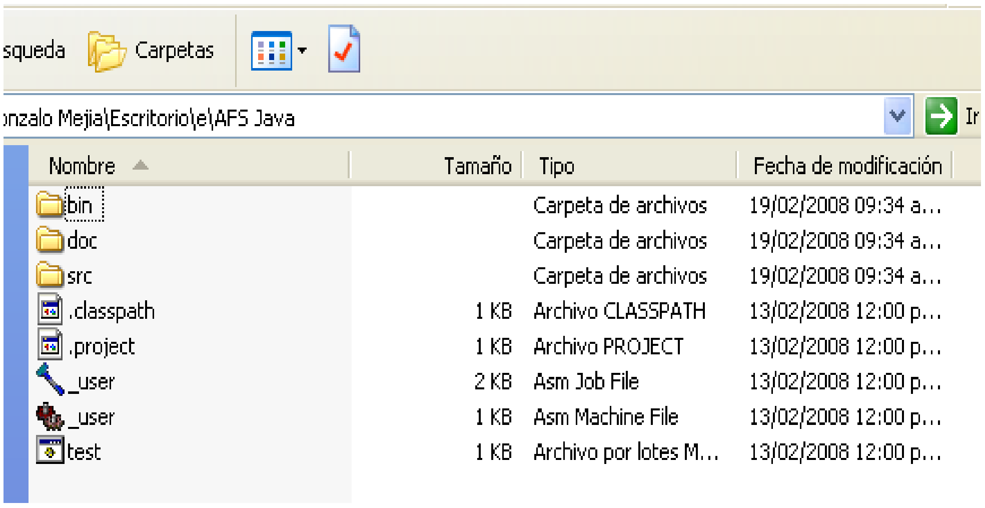
Figure 1. Installing AFS Java

Click on the "Select archive file" option button and browse the location of the AFSJava.zip file.
Select the AFS Java project (check the appropriate box and uncheck the other project boxes, if any) and click on the "Finish" button.
AFS Java is ready to work!

3.2.2 Creating a Blank Project and manually adding the AFS Java files

Again, depending on the IDE platform the procedure may vary. The steps would be:

1. Create a Blank Project
2. Make sure that the "src" and "bin" folders are in the project folder. If these directories are not present when the project is created, then create them manually.
3. Unzip the contents of AFS Java.zip into the hard disk folder of your choice. Figure 2 shows the unzipped project.

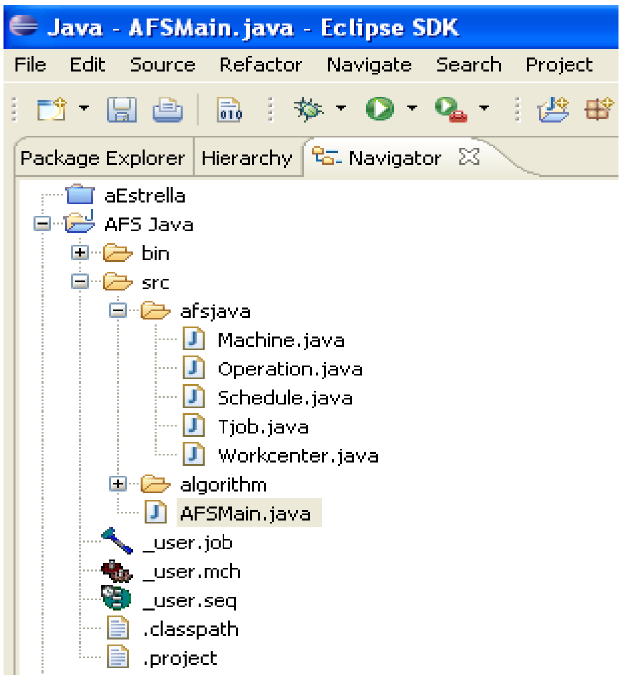


Nombre	Tamaño	Tipo	Fecha de modificación
bin		Carpeta de archivos	19/02/2008 09:34 a...
doc		Carpeta de archivos	19/02/2008 09:34 a...
src		Carpeta de archivos	19/02/2008 09:34 a...
.classpath	1 KB	Archivo CLASSPATH	13/02/2008 12:00 p...
.project	1 KB	Archivo PROJECT	13/02/2008 12:00 p...
_user	2 KB	Asm Job File	13/02/2008 12:00 p...
_user	1 KB	Asm Machine File	13/02/2008 12:00 p...
test	1 KB	Archivo por lotes M...	13/02/2008 12:00 p...

Figure 2. AFS Java Unzipped

4. Copy all files and folders in the root folder of your project.
5. Refresh the project to account for the above changes (See the documentation of the IDE you are using).
6. Compile and run the project (See the documentation of the IDE you are using).

Example: The following figure shows how the project looks like after creating the project (named AFS Java) and copying the AFS java folders and files. The _user.mch and _user.job files provided are for test purposes only. These correspond to a "Single Machine Scheduling Problem". This screen shot was taken from the Eclipse IDE. Make sure you have the Navigator Perspective on.



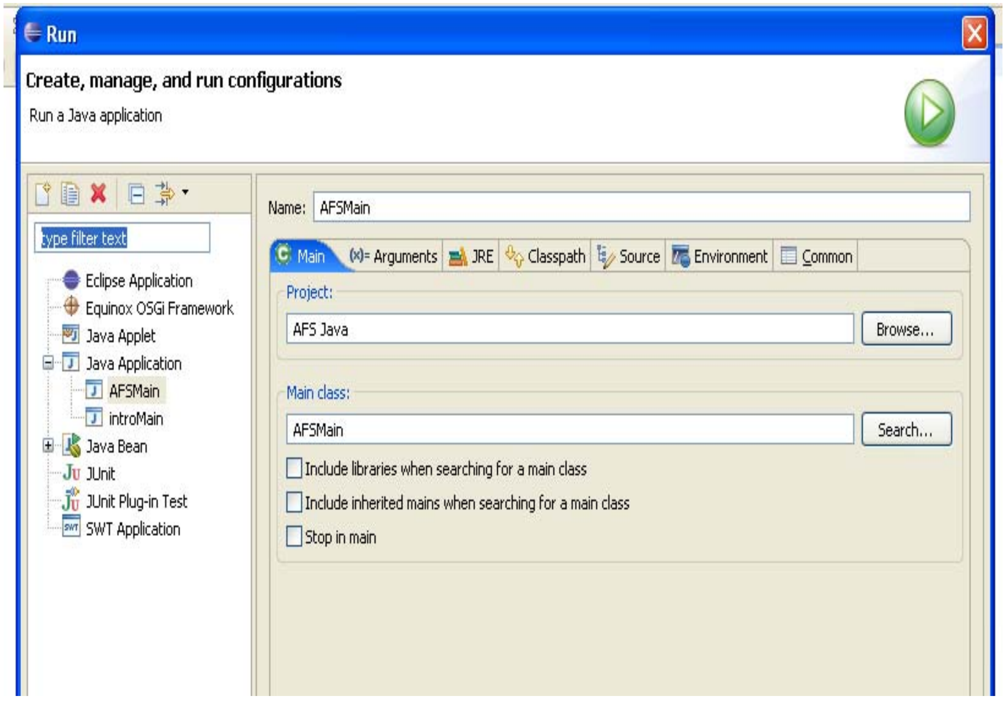
```
aEstrella
AFS Java
    bin
    src
        afsjava
            Machine.java
            Operation.java
            Schedule.java
            Tjob.java
            Workcenter.java
        algorithm
        AFSMain.java
    _user.job
    _user.mch
    _user.seq
    .classpath
    .project
```

Figure 3. AFS Java Installed

4 Running AFS Java

4.1.1 Running AFS Java from your IDE

The main class of AFS Java is “AFSMain”. When you run the program, make sure you are running the right configuration. If you are running Eclipse ® go to **Run/Run… /Java Application** and select AFSMain as shown in figure 4.


Run

Create, manage, and run configurations

Run a Java application

type filter text

- Eclipse Application
- Equinox OSGi Framework
- Java Applet
- Java Application
  - AFSMain
  - introMain
- Java Bean
- JUnit
- JUnit Plug-in Test
- SWT Application

Name: AFSMain

Main | Arguments | JRE | Classpath | Source | Environment | Common

Project: AFS Java [Browse...]

Main class: AFSMain [Search...]

- [ ] Include libraries when searching for a main class
- [ ] Include inherited mains when searching for a main class
- [ ] Stop in main

Figure 4. Running AFS Java from the Eclipse IDE

The default scheduling method is the classical WSPT (Weighted Shortest Processing Time) dispatching rule that runs on a Single Machine framework having 10 jobs. The job information is read from the _user.job file and the machine information from the _user.mch file. You can open these files directly from your IDE to see what it is in there. Be sure you open them with a text editor or directly from your IDE.

The schedule is shown in the _user.seq file. Open it with a text editor.

```
Number of Jobs 10
Number of workstations 1

Number of Jobs 10
Job000
Release Date 0
Due Date 0
Weight 1
Operation: Wkc000: 5 Status: A

Job001
Release Date 2
Due Date 0
Weight 1
Operation: Wkc000: 4 Status: A

Job002
Release Date 3
```

```
Due Date 0
Weight 1
Operation: Wkc000: 7 Status: A

Job003
Release Date 14
Due Date 0
Weight 1
Operation: Wkc000: 5 Status: A

Job004
Release Date 5
Due Date 24
Weight 1
Operation: Wkc000: 5 Status: A

Job005
Release Date 5
Due Date 24
Weight 1
Operation: Wkc000: 8 Status: A

Job006
Release Date 0
Due Date 20
Weight 1
Operation: Wkc000: 9 Status: A

Job007
Release Date 0
Due Date 16
Weight 1
Operation: Wkc000: 7 Status: A

Job008
Release Date 0
Due Date 9
Weight 1
Operation: Wkc000: 3 Status: A

Job009
Release Date 0
Due Date 16
Weight 1
Operation: Wkc000: 6 Status: A

Number of Workcenters 1
WorkCenter Wkc000
Machine 0 Available date 0 Initial Status A

Lekin files successfully read.

A schedule was generated. See the _user.seq file.
```

Note: If you want to test your algorithms without running Lekin we recommend that you create the _user.mch and _user.job from Lekin and copy them from the Temp directory of your computer into the root folder of the AFS Java project.

4.2 *Running AFS Java from Lekin*

Since Java does not create executable files, a Lekin plug-in written in Java requires a batch file (*.bat) which invokes the required Java commands.

4.2.1 Creating a Batch File

The example file “test.bat” is included with the AFS project.

Open the file with a standard text editor and see what the file has:

```
@echo off

SET CLASSTEMP=%CLASSPATH%;

REM The directory of the .class files goes here
SET CLASSPATH=%CLASSPATH%; hardDiskUnit/Javadirectory/AFS Java/bin

REM %0 is parameter #1 %1 is parameter #1, etc.
java AFSMain %0 %1 %2 %3 %4

SET CLASSPATH=CLASSTEMP;

echo Program Finished

EXIT
```

Comments:

1. The CLASSPATH command allows you to run the test.bat from anywhere. Otherwise the test.bat must be located in the same location of the AFSMain.class file.
2. You need to replace the command:

   ```
   SET CLASSPATH=%CLASSPATH%; hardDiskUnit/Javadirectory\AFS Java\bin
   ```
   with the location of the AFS *.class files

   The location of the *.class files may look like this:
   ```
   C:\Documents and Settings\your computer name\workspace\AFS Java\bin
   ```

3. The command:
   ```
   java AFSMain %1 %2 %3 %4
   ```

invokes the AFSMain class and the command line arguments (parameters). %1 corresponds to the parameter #1 (objective function. See Lekin), %2 is the second parameter which is the maximum running time (see Lekin). Parameters %3, %4, etc. are the additional user-defined parameters.

A note on command line arguments:
Lekin sends three command line arguments by default: (i) The working directory, (ii) the objective function and (iii) the maximum running time. You may use other arguments if you need them. Such arguments are stored as string variables (not as integers!) in the args[] variable.
If you need to retrieve the command line arguments in your program keep in mind that args[0] will store the argument 0 (working directory), args[1] the objective function and args[2] the maximum running time.

The following example of java code illustrates the retrieval of the objective function argument:

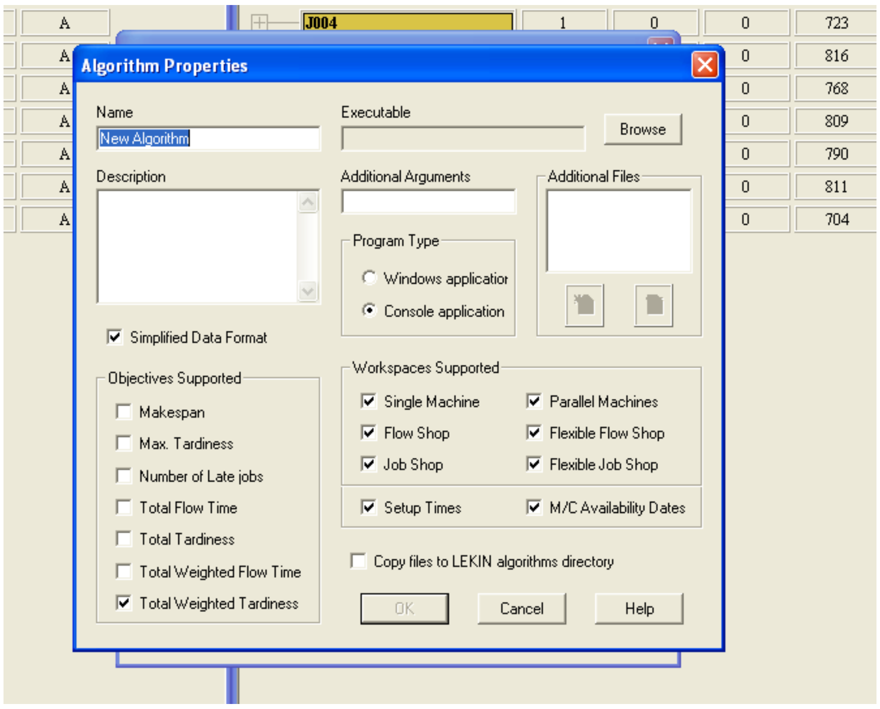
```
    String objectiveFunction = "";
      if (args[1].equals("0")){
      objectiveFunction = "makespan";
      }
      else if (args[1].equals("1")){
      objectiveFunction = "max tardiness";
      }
      else if (args[1].equals("2")){
      objectiveFunction = "max tardiness";
      }
      else if (args[1].equals("3")){
      objectiveFunction = "Number of late jobs";
      }
      else if (args[1].equals("4")){
      objectiveFunction = "Total Flow Time";
      }
      else if (args[1].equals("5")){
      objectiveFunction = "Total Tardiness";
      }

      else if (args[1].equals("6")){
      objectiveFunction = "Total Weighted Flow Time";
      }
      else if (args[1].equals("7")){
      objectiveFunction = "Total Weighted Tardiness";
      }
      else {
            objectiveFunction = "Invalid";
```

4.2.2 Loading the AFS Plug-in:

See the Lekin documentation on how to load plug-ins. The steps to load the batch file that invokes the Java commands are:

a) Create a new plug-in
b) In the **"Executable"** box (see below), browse the location of your batch file and put the name in there. See figure 5.
c) Add a name (e.g. AFS Java) and a description.
d) Check the appropriate boxes of the objective functions that your program supports.
e) Click the OK button
f) Check if the AFS Java plug-in appears in the **Schedule/Heuristic** menu.



A	J004	1	0	0	723
A				0	816
A				0	768
A				0	809
A				0	790
A				0	811
A				0	704

Algorithm Properties

Name: New Algorithm

Executable: [Browse]

Description:

Additional Arguments:

Additional Files:

Program Type: ( ) Windows applicatior (•) Console application

[✓] Simplified Data Format

Objectives Supported:
- [ ] Makespan
- [ ] Max. Tardiness
- [ ] Number of Late jobs
- [ ] Total Flow Time
- [ ] Total Tardiness
- [ ] Total Weighted Flow Time
- [✓] Total Weighted Tardiness

Workspaces Supported:
- [✓] Single Machine
- [✓] Parallel Machines
- [✓] Flow Shop
- [✓] Flexible Flow Shop
- [✓] Job Shop
- [✓] Flexible Job Shop
- [✓] Setup Times
- [✓] M/C Availability Dates

[ ] Copy files to LEKIN algorithms directory

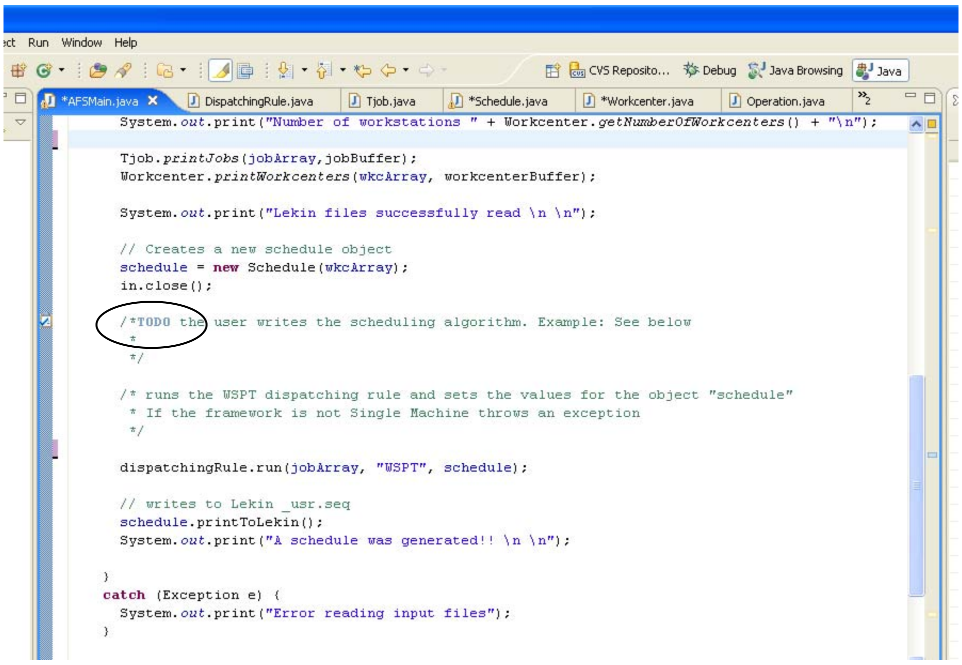
OK | Cancel | Help

Figure 5. Loading the batch file into Lekin

5 Functionality of AFS Java

The concept of AFS Java is simple: AFS reads the Lekin _user.mch and _user.job files, you write an algorithm to generate a schedule and AFS writes such a schedule to the _user.seq file. The best way to start is taking a look at the AFSMain.java file.

When you start writing your code, delete the line: `dispatchingrule.run(jobArray, "WSPT",schedule)` and add your code below the TODO text line.



ect Run Window Help
CVS Reposito... Debug Java Browsing Java
*AFSMain.java DispatchingRule.java Tjob.java *Schedule.java *Workcenter.java Operation.java

```
        System.out.print("Number of workstations " + Workcenter.getNumberOfWorkcenters() + "\n");

        Tjob.printJobs(jobArray,jobBuffer);
        Workcenter.printWorkcenters(wkcArray, workcenterBuffer);

        System.out.print("Lekin files successfully read \n \n");

        // Creates a new schedule object
        schedule = new Schedule(wkcArray);
        in.close();

        /*TODO the user writes the scheduling algorithm. Example: See below
         *
         */

        /* runs the WSPT dispatching rule and sets the values for the object "schedule"
         * If the framework is not Single Machine throws an exception
         */

        dispatchingRule.run(jobArray, "WSPT", schedule);

        // writes to Lekin _usr.seq
        schedule.printToLekin();
        System.out.print("A schedule was generated!! \n \n");

      }
      catch (Exception e) {
        System.out.print("Error reading input files");
      }
```

Figure 6. Inserting your code in AFS java

The following paragraphs describe the classes of AFS Java.

Classes

Operation: The class Operation defines one of the processing steps of a job. The members of the class Operation are: processing time, workcenter used on the operation and the “status” (type) of the operation. Note: The status definition is identical to one given in Lekin. The methods of Operation are “getters and setters” (i.e. methods to respectively retrieve the Operation information and to change the Operation parameters).

Important: You do not want to change the job and/or machine parameters (i.e. don’t use the set methods). If you must, use these methods carefully. If you change the parameters of the job or machine instances, such changes will be not reflected in Lekin.

Tjob: The class `Tjob` contains the parameters of a job as defined in Lekin and the methods to get and set such parameters. The members of a `Tjob` object correspond to the job parameters such as the job number, job ID, release date, due date, and route. The member “route” is an array of Operations.

The class `Tjob` also provides the method `readJobFile` that reads from the `_user.mch` file and stores the information on an array of `"Tjobs"` named `"jobArray"`.

Examples:

The user wants to retrieve the release date of the first job.
```
jobArray[0].getReleaseDate();
```

The user wants to retrieve the due date of the job # 5. Note: Jobs are numbered from 0 to n-1 where n is the number of jobs.
```
jobArray[5].getDueDate();
```

The user wants to retrieve the processing time of the first operation of the job # 5. Note: Operations are numbered from 0 up to m-1 where m is the number of workcenters.
```
jobArray[5].getProcessingTime(0);
```

The user wants to set the due date for job # 3 as 15.
```
jobArray[3].setDueDate (15);
```

The user wants to set the status of the first operation of the job # 4 as 'A'.
```
jobArray[4].setStatus ('A',0);
```

The user wants to retrieve the total number of jobs in the system. This is done by invoking the static method `getNumberOfJobs()` as follows:
```
 Tjob.getNumberOfJobs();
```

Machine: The class `Machine` contains members of a single resource (machine) and the methods to set and get such members. The members of the `Machine` class are the available date (release date), the initial status, and the machine ID and number.

Workcenter: A `Workcenter` is essentially an array of `Machines`. In addition the `Workcenter` class defines the “framework” (Single Machine”, “Flow Shop”, etc.). In addition to the getter and setter methods, this class provides the `readMachineFile` method which reads from the _user.mch file and stores the information on an array of `Workcenters` named `wkcArray.`

Examples:
The user wants to retrieve the number of machines in the workcenter #2. `wkcArray[2].getNumberOfMachines();`

The user wants to retrieve the setup matrix of the workcenter #4. `jobArray[4].getSetup();`

The user wants to retrieve the total number of workcenters in the system. This is done by invoking the static method `getNumberOfWorkcenters()` as follows. `Workcenter.getNumberOfWorkcenters();`

Important: `wkcArray` and `jobArray` are fixed length arrays whose sizes are equal to the Lekin defaults for maximum number of workcenters and jobs respectively. Therefore the instructions `wkcArray.length()` or `jobArray.length()` will return the default length of these arrays and not what you may expect (i.e. the number of workcenters and jobs in the system). If you want to retrieve the actual number of workcenters use the instruction:

```
Workcenter.getNumberOfWorkcenters()
```

Likewise if you want to retrieve the actual number of jobs use the instruction:

```
Tjob.getNumberOfJobs()
```

Schedule: The `Schedule` class has the methods to store a resulting schedule (the output of an algorithm) and write to the _user.seq file required by Lekin. An object of the `Schedule` class is a three-dimensional array of integers. The position *i, j, k* of the array contains the job number at the *k-th* position at the *j-th* machine of the *i-th* workcenter. You need basically three methods: the Schedule constructor, and the methods "`addtoSchedule`" and "`printToLekin`".

Description:
Constructor: `Schedule (Workcenter [] wkcArray);` Constructs a `Schedule` instance based on the array `wkcArray.` Make sure the method "`readMachineFile`" of the class `Workcenter` is called first.

Method: `addToSchedule(int jobNumber, int workcenterNumber, int machineNumber)`
Puts the job #jobNumber at the end of the schedule of the machine # machineNumber of the wokcenter # workcenterNumber.

Method: `printToLekin()`
Prints to the _user.seq file the sequence of each machine. `printToLekin` creates (or overwrites) the file _user.seq.

Example 1: Consider a Single Machine framework in which five jobs are processed in the following order 1,3,4,0,2. Let “schedule1” an instance of the Schedule class. Since

there is only one workcenter with one machine, everything will be assigned to the workcenter# 0 and the machine# 0.
The code would be:

```
Schedule    schedule1    =    new    Schedule    (wkcArray);
schedule1.addToSchedule(1,0,0);    //    adds    job    #1
schedule1.addToSchedule(3,0,0);    //    adds    job    #3
schedule1.addToSchedule(4,0,0);    //    adds    job    #4
schedule1.addToSchedule(0,0,0);    //    adds    job    #0
schedule1.addToSchedule(2,0,0);    //    adds    job    #2
schedule1.printToLekin();
```

Example 2: Consider a Parallel Machine framework in which six jobs are processed in two machines: Jobs 1, 2, 5 are processed in that order on the first machine of the workcenter and jobs 0, 3, 4 are processed in that order on the second machine. Again, let "schedule1" be an instance of the Schedule class. Since there is only one workcenter with two machines, jobs will be assigned to either to the machine #0 or the machine #1 of the workcenter# 0.
The code would be:

```
Schedule schedule1 = new Schedule (wkcArray);
schedule1.addToSchedule(1,0,0); // adds job #1 to mach #0
schedule1.addToSchedule(2,0,0); // adds job #2 to mach #0
schedule1.addToSchedule(5,0,0); // adds job #5 to mach #0
schedule1.addToSchedule(0,0,1); // adds job #0 to mach #1
schedule1.addToSchedule(3,0,1); // adds job #3 to mach #1
schedule1.addToSchedule(4,0,1); // adds job #4 to mach #1
schedule1.printToLekin();
```

Example 3: Consider a two-machine Flow Shop processing 5 jobs. Jobs are processed in the order 1,2,0,3,4 on both machines Again, let "schedule1" be an instance of the Schedule class. Since there are two workcenters with one machine each, jobs will be assigned to the machine #0 of both workcenters # 0 and #1.
The code would be:

```
Schedule    schedule1    =    new    Schedule    (wkcArray);
schedule1.addToSchedule(1,0,0);  //  adds  job  #1  to  wkc  #0
schedule1.addToSchedule(2,0,0);  //  adds  job  #2  to  wkc  #0
schedule1.addToSchedule(0,0,0);  //  adds  job  #0  to  wkc  #0
schedule1.addToSchedule(3,0,0);  //  adds  job  #3  to  wkc  #0
schedule1.addToSchedule(4,0,0);  //  adds  job  #4  to  wkc  #0
schedule1.addToSchedule(1,1,0);  //  adds  job  #1  to  wkc  #1
schedule1.addToSchedule(2,1,0);  //  adds  job  #2  to  wkc  #1
schedule1.addToSchedule(0,1,0);  //  adds  job  #0  to  wkc  #1
schedule1.addToSchedule(3,1,0);  //  adds  job  #3  to  wkc  #1
schedule1.addToSchedule(4,1,0);  //  adds  job  #4  to  wkc  #1
schedule1.printToLekin();
```

Note: AFS does not validate the schedule. If you put twice the same job on the schedule of a machine, AFS will not notice. This verification is performed by Lekin®.

TimedSchedule: The `TimedSchedule` class is an enhancement of the `Schedule` class. The `TimedSchedule` class not only stores the sequence of jobs on each machine but also returns the start and end times of each operation and the performance indicators (makespan, number of tardy jobs, total flow time, total tardiness, etc. See Lekin). A `TimedSchedule` is a three-dimensional array of objects of the `TimedOperation` class. As in the `Schedule` class, the position *i, j, k* of the array contains the operation scheduled at the *k-th* position at the *j-th* machine of the *i-th* workcenter. An object of the `TimedOperation` class contains the job number, the start and end time of the operation and the start and end of the setup time. Also contains the Lekin performance indicators of the schedule, even if it is a partial schedule.

The most used methods are the `TimedSchedule` constructor, the `addToTimedSchedule method` to add operations to a `TimedSchedule and the printToLekin method.`

Description:
Constructor: `TimedSchedule (Workcenter[] wkcArray, Tjob[] jobArray, Schedule schedule)` constructs a `Timedschedule` from a `Schedule` object. If an invalid schedule is passed as parameter, an exception is thrown and the program terminates. For example, if a job is put twice on the same machine or if a "cycle" (See the definition in the Pinedo (1999) book) is formed.

Method: `addToTimedSchedule(int jobNumber, int wksNumber, int mchNumber, Tjob[] jobArray)` adds the job #jobNumber to the machine #mchNumber of the workstation #wksNumber. All predecessors must have been scheduled. If not, an exception is thrown and the program terminates.

Method: `printToLekin()`
Overloaded method. Prints to the _user.seq file the sequence of each machine. `printToLekin()` creates (or overwrites) the file _user.seq.

Recommendations

Do not modify the code under the afsjava package (Tjob, Schedule, Operation, etc. classes). This created many problems to the first users (students of the Scheduling course, class 2006). If you feel, the code needs modifications, create new classes and/or methods. For example, if you want to create nice windows for AFS Java create intermediate classes (facades) so you do not need to make changes to the afsjava package.

Put your algorithms into new packages (e.g. the "algorithm" package). In this way you separate our code from your code and modifications will be easier. See the package "algorithm" that implements the WSPT dispatching rule.

Enjoy AFS Java and write your comments to
Gonzalo Mejía
Universidad de Los Andes
Departamento de Ingeniería Industrial

Carrera 1E #19A-40
Room ML 703
Bogotá, Colombia
E-mail: gmejia@uniandes.edu.co