# Conference Track Management Problem Design Notes

Doug Halperin
15Nov2013

These are notes on the design and implementation of my solution to the Conference Track Management Problem Two as outlined in the ThoughtWorks Code Assignment.

In the problem, a large number of Sessions must be scheduled into Tracks during the Conference. The list of Sessions and their time lengths are received in an input file.

The solution was developed using the Eclipse IDE, Java 1.6, and JUnit 4 on a Mac.

Several key elements of the design that I followed include:

- Use of a BlockFormat prototype class to define the different time periods of a conference track. This allows for rapid adoption of the Scheduler for conference configurations beyond the sessions-lunch-sessions-networking configuration of the sample problem. By the addition of a TimeOffset class that includes a Day Number as well as Hour and minute, the tracks can be set up to extend over multiple days.
- As the Scheduler operates, additional Tracks are created as necessary. These are instantiated following the prototype of a Track defined by an array of BlockFormats.
- By adding an optional input parameter specifying the encoding of the input file as , the software can be rapidly extended to support non-UTF-8 character sets. In this implementation only UTF-8 format is supported.
- While the sample set of Sessions was small and the problem did not state any criteria for optimizing a solution, in general, this kind of scheduling optimization problem is quite complex. As a result, a strategy pattern is used to allow for the rapid deployment (and testing) of alternative scheduling strategies by creating sub-classes of the SchedulingStrategy abstract class. This class defines the methods that must be implemented as well as housing utility methods for common use of sub-classes. In this implementation a single BruteForceStrategy was developed that in the scheduling process attempts to fit Sessions into existing Tracks but simply creates additional Tracks as needed without making any attempt to rearrange already placed Sessions. This gets the job done and is within 2 times of the optimal packing.
- Unit testing was key to creating a robust solution. JUnit was used extensively and unit level test cases were created in this implementation for a majority of the classes and methods.

- A Config class was created to separate out constant (read arbitrary), default, and configuration values. This consolidates into a single location changes that would need to be made to operate for conferences with different Track configurations, different conventions, etc.
- Javadoc document was written and produced for the package classes so that others can extend this solution. The produced output was not included in the submittal.

In this implementation the following assumptions were made beyond the information specified in the problem definition:

- A conference can have an arbitrary number of tracks
- There are no sub-groupings of Sessions into tracks that are preferred. For instance no attempt is made to place all Sessions with the word "Ruby" in the name into the same Track.
- Session names do not start with the # character. Lines in the input file beginning with # are ignored. This facilitates comment lines in the input file. Blank input lines are also ignored.
- Session names must contain non-whitespace and are left and right trimmed of whitespace.
- Session durations must be non-zero in length.
- The additional time formatting of "[0-9]+(hr|hour)s?" and "[0-9]+(hr|hour)s?[0-9]+(min|minute)s?" are supported.
- The only supported import mechanism was input file as specified by a command line property.
- The Scheduler operates in a single Thread. No attempt to support synchronization was made.
- The convention of <ClassName>Test has been followed for all JUnit class level test classes.
- Output of the results is made to System.out.

An example run from the command-line of the software would be by invoking the following line:

*java -Dfile=sessions.txt com.doughalperin.scheduler.Scheduler*

The properties supported in this implementation are:

- file – this is the name of the input file. This is required
- encoding – this is the character encoding of the file. This is optional. The only currently supported value is UTF-8.

Assuming paths are properly set up, the Suite of all JUnit test cases can be involved via the following:

*java org.junit.runner.JUnitCore com.doughalperin.scheduler.TestSuite*

As witness to the successful operation of the Scheduler, here is copy of output from Scheduler for the sample input provided with the problem.

```
Welcome to the Conference Track Manager!

Input file is sample/SampleInput.txt with encoding UTF-8
Scheduling Strategy is BruteForceStrategy

Track 1
09:00AM Writing Fast Tests Against Enterprise Rails [60min]
10:00AM Communicating Over Distance [60min]
11:00AM Rails Magic [60min]
12:00PM Lunch
01:00PM Ruby on Rails: Why We Should Move On [60min]
02:00PM Ruby on Rails Legacy App Maintenance [60min]
03:00PM Overdoing it in Python [45min]
03:45PM Ruby Errors from Mismatched Gem Versions [45min]
04:30PM User Interface CSS in Rails Apps [30min]
05:00PM Networking Event

Track 2
09:00AM Common Ruby Errors [45min]
09:45AM Accounting-Driven Development [45min]
10:30AM Pair Programming vs Noise [45min]
11:15AM Clojure Ate Scala (on my project) [45min]
12:00PM Lunch
01:00PM Lua for the Masses [30min]
01:30PM Woah [30min]
02:00PM Sit Down and Write [30min]
02:30PM Programming in the Boondocks of Seattle [30min]
03:00PM Ruby vs. Clojure for Back-End Development [30min]
03:30PM A World Without HackerNews [30min]
04:00PM Rails for Python Developers [5min]
05:00PM Networking Event
```