LEADERBOARD APPLICATION

Table of Contents

Scope

This is a demo application demonstrating a modular and flexible approach to deal with a hypothetical need of an online game. We are assuming that the game facilitates multiple servers across continents in order to provide access to the players and each one of these servers exposes a leaderboard in some format. Our application's target is to retrieve all the available leaderboards, merge the data and perform an action based on the merger outcome.

The design of the application is modular and provides extra flexibility. It is able to support multiple input sources (disk file, http, web service), multiple parsers (we assume different formatting between the American and European servers for example), multiple merger actions (e.g. based on player points or player kills) and multiple final actions to publish the results (email, twitter).

The modular/flexible design is achieved by adhering to some basic OO design principles and patterns. First and foremost is the *Single Responsibility* principle by ensuring that each class has one and only one responsibility. This also makes easier the appliance of the *Open-Closed* principle by braking down the algorithm into multiple steps where each step is executed through an interface, so a different implementation - defined in the configuration file – can be used on each execution (*Strategy pattern*). This way each new requirement can be fulfilled by writing new implementations for existing interfaces and changing the configuration and not by applying changes to the existing code. This achieves better code readability and fewer chances to introduce bugs in the existing code.

Finally, an option is provided to utilize multiple threads in order to execute the algorithm.

Algorithm Description

The steps of the algorithm are discrete and can be described as follows

- **Step 1** - Read from an input source
- **Step 2** - Parse the input provided by Step 1 and generate a list of TopPlayer objects
- **Step 3** - Repeat steps 1,2 above for as many times as different input sources have been defined
- **Step 4** - Merge the results from the previous steps
- **Step 5** - Perform an action based on the final results from the merger

Implementation Details

The package structure has as follows

- **conf**
  Services responsible for handling XML file configuration
- **leaderboard**
  This is the heart of the application. The algorithm steps are executed here and each step uses services that reside in the following sub-packages which contain an interface and the available implementations.
  - **inputsourcehandler**
    All implementations for opening connections to different input sources (http, file etc)
  - **inputanalyzer**
    All implementations for parsing the input source stream opened by an InputSourceHandler.
  - **merger**
    All implementations for managing the results produced from the inputAnalyzer implementation(s)
  - **publisher**
    All implementations for publishing the results of the merger (email, twitter, etc)

- **model**
  All model objects used through the lifetime of the application like TopPlayer
  - **xml**
    Contains all objects generated by JAXB that map leaderboard_conf.xml to objects.
- **tools**
  Utility classes that act as support tools for development and testing purposes
- **utils**
  Utility classes providing generic functionality needed by other services
- **web**
  Everything concerning the simple front-end page for testing the application is located in here
  - **validator**
    Contains available validators for input in the interface. Currently contains only an email validator
- **webservice**
  This includes a simple REST API that returns a fixed list of TopPlayer objects simulating in this way an input source.

The heart of the application lies in the implementations of the `LeaderboardHandler.java` interface (inside the `leaderboard` package) where the algorithm steps described in Algorithm Description are actually executed. The *leaderboard_conf.xml* is the XML file holding the predefined configurations for the leaderboards. Each time a single configuration gets executed.

There are 2 available implementation of the `LeaderboardHandler.java` interface. The default implementation of the algorithm which is single threaded can be found in *SingleThreadLeaderboardHandler.java*. The multithreaded version is inside *MultiThreadedLeaderboardHandler.java*. For the multithreaded version the *ExecutorService* and *Future* constructs from *java.util.concurrent* package are utilized. The actual portion of the algorithm which is executed in parallel is Step 2. Each one of the input sources defined in leaderboard_conf.xml is handled by a different thread. Once all data from all threads are gathered Steps 4,5 continue execution.

Notes:

- The front-end interface provided is just for demonstrating that the application is actually functional.
- In a production version most probably a database would have been used instead of an XML along with a front-end that will allow the user to customize the configuration. Since the spotlight is elsewhere a decision was made to not include database dependencies in the project and use XML instead.

Application Demo

The application executable (war file) has been deployed on the cloud for demonstration purposes. One can test the application by visiting the following URL http://leaderboard.zisist.cloudbees.net/ .
Delay of a few seconds might be encountered upon first visit due to sleep application mode applied by cloudbees service.
The UI is as simple as it can get since it is used only for demonstration purposes. There is a drop down menu called *Configuration* where the user can select one of the predefined configurations. Upon selection a description of the selected configuration will immediately appear below the drop-down menu along with any needed input for the selected configuration (e.g. email to publish the results). By clicking on *Publish* the algorithm described in Algorithm Description gets executed according to the selected configuration.
One email and one twitter account have been setup for the demo purposes. The twitter account is setup appropriately and used by the 2nd predefined configuration. The account details follow

Email : www.gmail.com
Username : leaderboard.zt@gmail.com
Password : leader123!@#
Twitter : www.twitter.com
Username : leaderboard.zt@gmail.com
Password : leader123!@#

Finally, the following sources provide lists of Top Players for the demo purposes

- a web service returning a fixed-length list of top players in JSON format at http://leaderboard.zisist.cloudbees.net/api/players/leaders

- A file containing a fixed length list of top players in text format located at http://fttz.myqnapnas.com/leaders_part2.txt
- A file containing 120.000 top players located at http://leaderboard.net63.net/leaders
- Another file containing 120.000 top players located at http://leaderboard-2.site50.net/leaders-2

Technologies Used

- Java
- JUnit
- Spring
- XML
- JSON
- JSF
- RichFaces
- RESTful web services