Self-Directed Bootcamp Tutorial

Prerequisites

- Install maven 3
- Configure maven settings as shown here.
- On Mac, install Ruby 1.9 (for running services from the command line)
- Configure git

```
git config --global user.name "FULL NAME"
git config --global user.email "EMAIL@proofpoint.com"
```

Clone the Bootcamp projects

```
$ git clone https://github.com/psummers/bootcamp.git
Cloning into bootcamp...
remote: Counting objects: 89, done.
remote: Compressing objects: 100% (51/51), done.
remote: Total 89 (delta 23), reused 87 (delta 21)
Unpacking objects: 100% (89/89), done.
$
```

Build the project

```
$ cd bootcamp/person
$ mvn install
[INFO] Scanning for projects...
...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building person 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------

[...SNIP downloading the entire internet...]

[... project builds ...]

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1:29.156s
[INFO] Finished at: Mon May 07 07:39:21 PDT 2012
[INFO] Final Memory: 16M/81M
[INFO] ------------------------------------------------------------------------
$
```

Run the sample application

At this point the instructions are different for windows/non windows users since the application cannot be launched from command line on windows.

Run From Command Line (Mac/Linux users)

Inspect the build artifacts

```
$ cd target
$ ls
archive-tmp/
classes/
generated-sources/
maven-archiver/
person-1.0-SNAPSHOT-distribution.tar.gz
person-1.0-SNAPSHOT-sources.jar
person-1.0-SNAPSHOT.jar
surefire/
surefire-reports/
test-classes/
```

- The `.jar` file contains the compiled classes from this project.
- The `-sources.jar` file contains the java source code for this project.
- The `.tar.gz` is the final server binary that we use in production.

Unpack the application

```
$ tar xzf person-1.0-SNAPSHOT-distribution.tar.gz
```

Add configuration

```
$ cp -R -p ../etc person-1.0-SNAPSHOT
$ touch person-1.0-SNAPSHOT/etc/jvm.config
$ ls person-1.0-SNAPSHOT/etc/
config.properties
jvm.config
log.properties
```

Run server

```
$ cd person-1.0-SNAPSHOT
$ bin/launcher run
```

Output of run (via intellij or launcher)

```
2011-11-15T21:21:37.253-0800  INFO   main    com.proofpoint.log.Logging  Logging to stderr
2011-11-15T21:21:37.266-0800  INFO   main    com.proofpoint.bootstrap.Bootstrap  Loading
configuration
2011-11-15T21:21:37.351-0800  INFO   main    com.proofpoint.bootstrap.Bootstrap  Initializing
logging
...
2011-11-15T21:21:39.820-0800  INFO   main    com.proofpoint.bootstrap.LifeCycleManager   Life
cycle starting...
2011-11-15T21:21:39.821-0800  INFO   main    com.proofpoint.bootstrap.LifeCycleManager   Life
cycle startup complete. System ready.
2011-11-15T21:21:39.822-0800  ERROR   Announcer-0 com.proofpoint.discovery.client.Announcer
Cannot connect to discovery server for announce: No discovery servers are available
```

You can ignore the ERROR from the com.proofpoint.discovery.client.Announcer, this is only indicating that we don't have a discovery server to point to yet. We'll cover that later on.

Stop the service with ctrl-C.

Run the server from within IntelliJ (Mac, Linux, and Windows)

Open project from within IntelliJ

1. Launch IntelliJ
2. Select Open Project
3. Navigate to the directory where the project sources are located (i.e. the "person" directory) and open the file named pom.xml

Run Service from within IntelliJ

1. Open Main class by choosing the "Go To" -> "Class..." menu and entering "Main"
2. Open "Main"->"Edit Configurations" and add "-Dconfig=etc/config.properties" under "VM parameters"
3. Right click on the main() method and select "Run"

Test from the command line:

With the service running either in IntelliJ or from the command line, in another terminal window type the following commands:

```
$ curl -X PUT -H "content-type: application/json" --data '{ "name": "Matt", "email":
"mstephenson@proofpoint.com"}' "http://localhost:8222/v1/person/mattstep"
$ curl "http://localhost:8222/v1/person?pretty"
[ {
  "email" : "mstephenson@proofpoint.com",
  "name" : "Matt"
} ]
$ curl "http://localhost:8222/v1/person/mattstep?pretty"
{
  "email" : "mstephenson@proofpoint.com",
  "name" : "Matt",
  "self" : "http://localhost:8222/v1/person/mattstep?pretty"
}
```

Add the project to your bootcamp git repository

NOTE: Configuration management for external Bootcamps is TBD. We'll investigate how this can be managed via Github so that we can conduct code reviews with remote students.

Add a feature to the server

In the current version of the sample server, the "person" entries contain just a name and email, but no id. This makes it hard for clients to know what url to call to DELETE or GET specific entries. We want to add an "id" field and expose it as part of the http API, as in the following example:

```
{
  "id": "alice",
  "name": "Alice",
  "email": "alice@proofpoint.com"
}
```

The API needs to validate that the id specified in the json structure matches the id that's part of the url and return a "400 Bad Request" error otherwise.

So, this call should fail:

```
$ curl -X PUT -H "content-type: application/json" --data '{ "id": "alice", "name": "Alice",
"email": "alice@proofpoint.com"}' "http://localhost:8222/v1/person/bob"

On Windows:
    >curl -X PUT -H "content-type: application/json" --data "{ \"id\": \"alice\", \"name\":
\"Alice\", \"email\": \"alice@proofpoint.com\"}" "http://localhost:8222/v1/person/bob"
```

As part of this task you should:

1. Implement the change and unit tests
2. Commit it to your local repository
3. If using a remote git server, push it to origin.