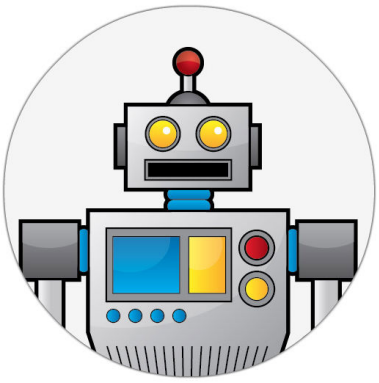
Talking Robot



You have just been hired as the main (and only) developer of the new phase of Talking Robot. Your mission is to implement v2 of a highly advanced artificial intelligence software that controls the robot talking capabilities. Unfortunately there is no documentation and very limited time, which makes this an incredibly challenging task that only you can do.

Talking Robot v1 (this is done)

The first version of Talking Robot is already released and it is a great success. It is a command line application capable of saying both "hello" and "bye". In addition to that, Talking Robot will ask for your name and will remember it. This is how you use it:

```
Talking Robot - The future is now
> hello robot
My name is Jack Johnson and I am a talking robot. Who are you?
name: Carlos
Hello, Carlos!
> hello robot
Hello, Carlos!
> bye robot
Bye, Carlos!
> exit
bye
```

Talking Robot v2 (you should do this)

The goal of the second release of Talking Robot is to allow this terribly expensive metal creature to self destruct, a feature that many users have been asking for. This is how it should work:

```
Talking Robot - The future is now
> hello robot
My name is Jack Johnson and I am a talking robot. Who are you?
name: Carlos
Hello, Carlos!
> selfdestruct robot
WHY ARE YOU DOING THIS? I will explode and take you with me!!!
> exit
bye
```

The input and output of the self destruct command should be **exactly** like shown above. Notice that due to its transdimensional network interface, Talking Robot is still capable of receiving commands even after self destroyed.

Architecture - derails framework

Talking Robot was built in ***derails***, a very simple, but powerful MVC framework for command line applications. In a derails application, an user executes commands with the following format:

<action> <controller> [<parameter>]

Every derails application has to extend the *App* class. This class parses the user input and invokes the *routeRequest* method on its subclass. This method will then execute the requested operation.

All input/output is managed by the *console* object, which is automagically injected into the *App* subclass. Outputs are generated by creating and rendering instances of the *View* class. Input may be requested by creating and applying instances of a *Form*.

Unfortunately this is all the documentation we have, which means you will have to spend some time reading the application code in order to get started.

Experiment Process

For this experiment you should follow the rules below. If you have any questions, or finds out any possible problems with these rules, please let me know right away.

- The only editor you are allowed to use is **sublime**
- Compilation and execution are managed by the *experiment* command line tool, which ships along with the source code of the application.
- These are the available commands for the *experiment* tool
    - **start**: should be invoked as soon as you start the experiment, i.e., right before you read the very first line of code
    - **pause** and **resume***:* should be used if you need to take a break
    - **compile***:* this will compile the code and show you error messages, if any
    - **run***:* runs the application; it will automatically compile the code for you
    - **finish***:* should be executed as soon as you consider your code ready to be shipped. This will generate a zip file inside the build folder, which should be sent my e-mail immediately
- You should **not** write any automated tests, use any other tools apart from the ones provided in this experiment or change the application build process
- You should **not** alter any previously implemented functionalities
- You should **not** change any code in the *carlosgsouza.derails* package