SOFT8023

Tutorial Document 01

Dr. Larkin Cunningham

September 15, 2021

1 Introduction

This series of tutorial documents, to be done each week during your lab time, will take a step-by-step approach, slowly introducing new concepts and providing you with a scaffold on which to build other distributed systems. Solutions will be provided each week, but you are advised to code along because active learning tends to garner the best results for learners. These documents are like a portal into the thoughts of a software development practitioner - some decisions made early on will seem wrong in later documents. This is perfectly normal; modern software development practitioners tend to get into coding their understanding of the requirements early and *refactor* later, continuously improving both the design and the code. Don't be surprised, therefore, to see entire sections of code substantially modified or even discarded.

These documents will be accompanied by videos that provide additional discussion, insight, etc. Multiple modes of learning are best - hearing the expert talk, watching the expert in action, reading the explanations, seeing the steps taken and learning why they are taken, manually coding the solution yourself.

2 The App

Darts. What a wonderful sport. I'm biased, but that's okay. Often associated with the pub (and large-bellied men), the game is in fact played by people of all ages and abilities. Some play competitively, while others simply use it as a way to pass the time in a mindful way. As women have begun to take the game more seriously, they are now competing with the men and getting their fair share of victories on the big stage (note Fallon Sherrock beating two ranked men in the 2020 PDC world championship).

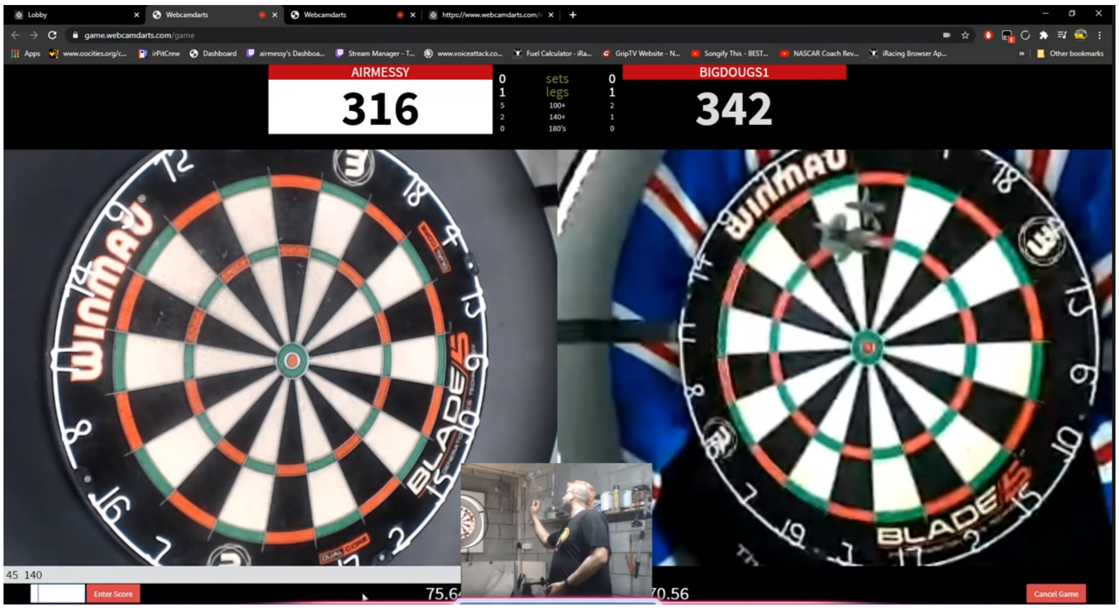


Figure 1: An example of webcam darts.

During the coronavirus lockdown, webcam darts became very popular. Two players, each with a webcam pointed at their dartboard, take turns to visit the board, throw their three darts, and record the score. This, of course, requires software for both players to record their score and have the score updated for both (see Figure 1 for an example). This is the app we will build during these tutorials.

If you already play darts, well done, you have great taste in sport (yes it is a sport!). You can play along and test the software. If you don't play darts, I recommend you do, but if you definitely have no intention (or don't have the room for a dartboard), then you can simply enter in made-up scores (or watch a match on YouTube and enter the scores the players get).

3 The Technology

While I have previously taught distributed systems using Java, I have decided to opt for Python this year. I think Python is more accessible, will be less frustrating, easier to run / test / debug, etc. We will be using Python 3. I will be using version 3.6, but 3.7 or 3.8 will do fine also. There should be no issue developing and running your application on Windows, Mac or Linux. In fact, I recommend a Linux VM if you are using Windows, e.g. Ubuntu 20.04.

With respect to an IDE for development, you could use the free version of PyCharm, but you can also use an advanced text editor, such as Notepad++ or Sublime Text.

For the most part, I will be taking an object-oriented approach. Sometimes these will be distributed objects that communicate using some form of messaging technology - and I mean messaging in a loose sense, i.e. sending data / requests from a producer to a consumer or client to server (and vice versa).

4 Getting Set Up

Week 1 is spent making sure you are ready to move forward and learn new concepts. We'll make sure you have what you need installed and that you can run a simple Python application.

4.1 Is Python 3 Installed?

Using the command prompt / terminal, enter the following:

```
python --version
```

You should see output similar to:

```
Python 3.6.3
```

Note: on Linux, you may need to enter:

```
python3 --version
```

4.2 Is git Installed?

Using the command prompt / terminal, enter the following:

```
git --version
```

You should see output similar to:

```
git version 2.17.1
```

4.3 Clone My Test Repository

I have an early iteration of the application in a public GitHub repository. We are just going to use it to test your environment (Python, git, IDE, etc.).

If you don't already have a directory for cloning git repositories, create one. For example, you might create the directory D:\Repos or /home/myself/git.

You have a couple of options when it comes to using git with your projects. The first option is to manage the project yourself using the command line (or a GUI git tool).

The second is to use the git capabilities of an IDE, such as PyCharm. The following two subsections show both options.

4.3.1 Manually Creating a git Repository

Change into that directory and enter the following command:

```
git clone git@github.com:dlarkinc/soft8023-darts.git
```

The output should look like this:

```
Cloning into 'soft8023-darts'...
remote: Enumerating objects: 17, done.
remote: Counting objects: 100% (17/17), done.
remote: Compressing objects: 100% (16/16), done.
remote: Total 17 (delta 2), reused 13 (delta 1), pack-reused 0
Receiving objects: 100% (17/17), done.
Resolving deltas: 100% (2/2), done.
```

4.3.2 Using the git Capabilities of PyCharm

In PyCharm, select "Get from Version Control" from the VCS menu. Figure 2 shows where you enter the GitHub url from Section 4.3.1. Select an appropriate directory where you will store git repositories and then click "Clone".

4.4 Use pipenv Instead of pip

The pip command installs packages on a system-wide basis, whereas the pipenv command does so only on a project-by-project basis. Install it with the following:

```
pip install pipenv
```

Chances are, this won't add `pipenv` to the PATH, so look in the output from the above command for a path to the Scripts directory. For example:

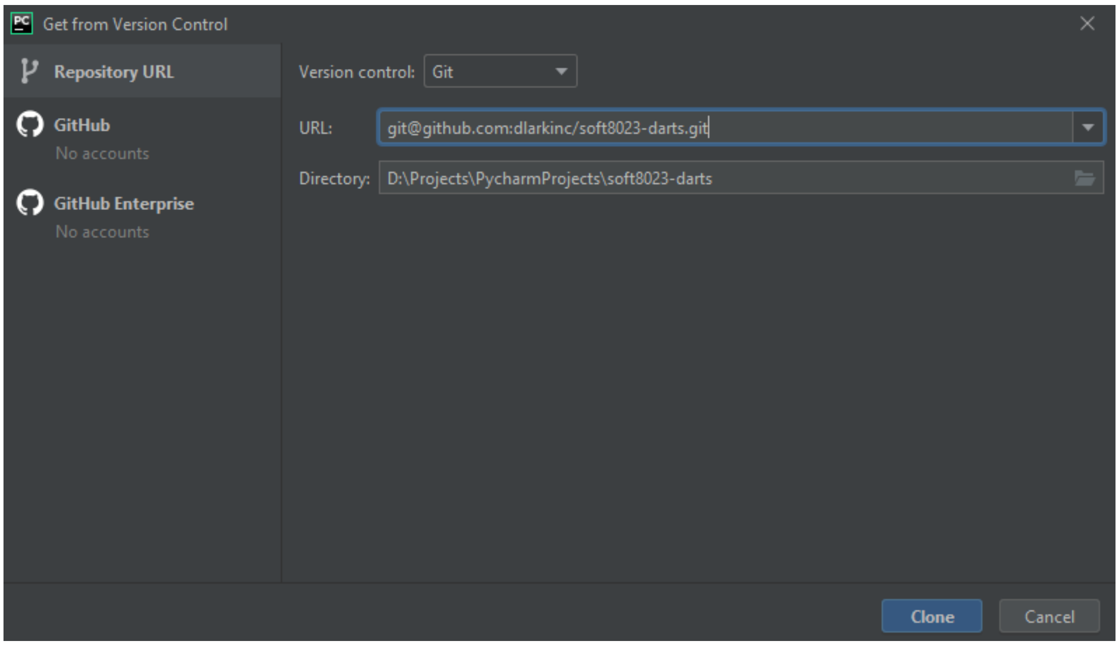
```
The script virtualenv.exe is installed in 'C:\Users\larki\AppData\Roaming\Pyt
```

Update the PATH environment variable on your system to include the Scripts directory and restart your command prompt / terminal.

Now change to the project directory and enter the following command:

```
pipenv install tinydb
```

`pipenv install` creates a Pipfile, which allows you to manage the dependencies for your project. This is important when sharing projects, for example when submitting an assignment to your lecturer. It ensures everyone can easily install the same set of



Version control: Git

URL: git@github.com:dlarkinc/soft8023-darts.git

Directory: D:\Projects\PycharmProjects\soft8023-darts

Figure 2: Getting a project from GitHub.

dependencies and are more or less working within the same environment. You should see output similar to Figure 3.

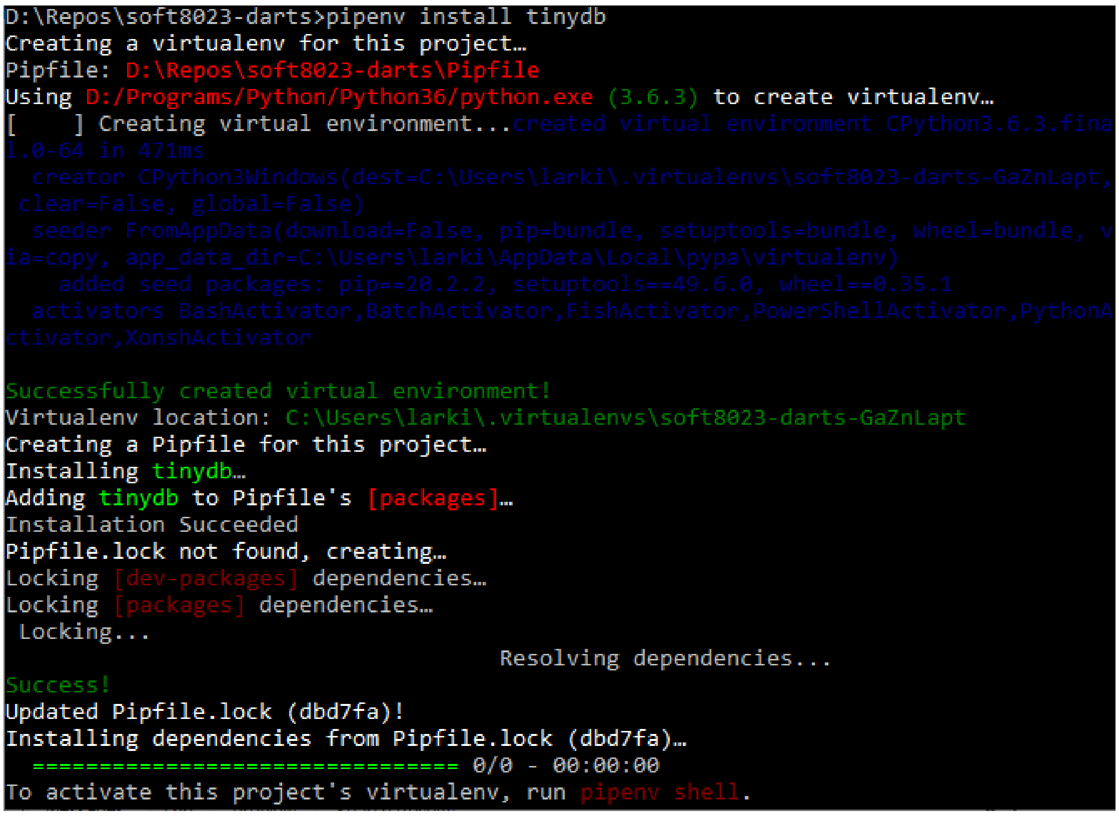
This should create a Pipfile very similar to Listing 1. The only relevant line for now is line 9 where you see tinydb listed with "*" meaning latest version.

Listing 1: Initial Pipfile with tinydb dependency.

```
1  [[source]]
2  name = "pypi"
3  url = "https://pypi.org/simple"
4  verify_ssl = true
5  
6  [dev-packages]
7  
8  [packages]
9  tinydb = "*"
10 
11 [requires]
12 python_version = "3.6"
```

`tinydb` is a package for a basic document store database that we will make a lot of use of throughout the remaining tutorial documents.

It should be noted that Pycharm supports pipenv and can be selected when creating a new project. This is an alternative way to create the Pipfile.



```
D:\Repos\soft8023-darts>pipenv install tinydb
Creating a virtualenv for this project…
Pipfile: D:\Repos\soft8023-darts\Pipfile
Using D:/Programs/Python/Python36/python.exe (3.6.3) to create virtualenv…
[    ] Creating virtual environment...created virtual environment CPython3.6.3.fina
l.0-64 in 471ms
  creator CPython3Windows(dest=C:\Users\larki\.virtualenvs\soft8023-darts-GaZnLapt,
 clear=False, global=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundle, v
ia=copy, app_data_dir=C:\Users\larki\AppData\Local\pypa\virtualenv)
    added seed packages: pip==20.2.2, setuptools==49.6.0, wheel==0.35.1
  activators BashActivator,BatchActivator,FishActivator,PowerShellActivator,PythonA
ctivator,XonshActivator

Successfully created virtual environment!
Virtualenv location: C:\Users\larki\.virtualenvs\soft8023-darts-GaZnLapt
Creating a Pipfile for this project…
Installing tinydb…
Adding tinydb to Pipfile's [packages]…
Installation Succeeded
Pipfile.lock not found, creating…
Locking [dev-packages] dependencies…
Locking [packages] dependencies…
 Locking...
                                     Resolving dependencies...
Success!
Updated Pipfile.lock (dbd7fa)!
Installing dependencies from Pipfile.lock (dbd7fa)…
  ================================ 0/0 - 00:00:00
To activate this project's virtualenv, run pipenv shell.
```

Figure 3: Installing a package using pipenv.

Additionally, it should be noted that you will more often see a requirements.txt to be used with pip than a Pipfile with pipenv. Both options are available with PyCharm: `https://www.jetbrains.com/help/pycharm/managing-project-dependencies.html`.

4.5 Run Using pipenv

To run a script within the project's own pipenv environment, enter the following:

```
pipenv run python test.py
```

You should something similar to the following output after a few seconds:

```
Insert attempted on Dupe      0.3661381316390109
```

Don't worry about this output now (or what random number is included). If you see it, it just means that things are working as expected. Note: If you configured PyCharm to use pipenv for the project, this will happen automatically when you run a script.

5 A Quick Run Through the Python Code

This module does not teach beginner Python coding. It is expected that you have done Python coding to at least an intermediate level. However, let's just look at the code in that repository you cloned to remind ourselves of a few things about Python.

5.1 The DartsMatchDao Class

Python supports the object-oriented programming paradigm, though it also supports non-OO procedural programming. If we look at the file darts_match_dao.py in the dao folder (see Listing 2), we can make a number of observations:

1. We import the classes TinyDB and Query from the tinydb module (line 1). This module is not part of the core Python install and had to be specified as a dependency in our Pipfile file. Tinydb isn't a database server like MySQL, MongoDB or Redis. It simply manages standard files on the filesystem using JSON format to store documents (like you see in MongoDB, for example). It is not suited to a multi-user distributed system—paradoxically, this makes it ideal for us when learning about distributed systems, because we have to manually implement some of the concepts / mechanisms we take for granted in a multi-user database management system like MongoDB, such as ensuring we don't have two users updating the same piece of data at the same time resulting in a lost update for one of them.
2. We import the entire threading and time modules (lines 3 and 4). These are built-in and we do not need to specify them in Pipfile. For an explanation of the difference

between `from...import` and `import...` see `https://stackoverflow.com/questions/9439480/from-import-vs-import`.

3. We define a class DartsMatchDao (line 6)—here I am using the convention of UpperCamelCase names for classes. If I was inheriting from another class, I would have included the base class as a parameter to this class is brackets, e.g. `class DartsMarchDao(BaseDao):`. The Dao bit at the end stands for Data Access Object (DAO). This is one of the *design patterns* we will take a look at in the early stages of the module and is just a conventional way of writing classes that only do database operations on a particular type of entity / table / document. In this example we provide one or more database operations on the DartsMatch document (tinydb is a document store, remember, so we talk in terms of documents, but these are very similar to tables / relations in a relational database).

4. I include an initialisation method (line 8), similar to a constructor method in Java. We pass a reference to the instance (self) as a parameter. We then create two instance variables, db and lock, providing initial hard-coded values. These are class-level variables—the lock defined here is a class-level lock, meaning we can have all methods check that one lock to see if the document store is currently available to the current thread of execution (more on all those concepts later). We can also pass in values as parameters to __init__ just like with Java constructors. You can also implement the concept of encapsulation with private properties in Python—see `https://www.datacamp.com/community/tutorials/property-getters-setters`.

5. I define a public method (no double-underscores) called add (line 12). When there is no current class-level lock, I will acquire it (and wait otherwise). I simulate some heavy activity by sleeping for 4 seconds before processing the query to add a new match to the document store (assuming it doesn't already exist). When my work is done, I release the lock so some other thread of execution can come into the class to perform a data store operation. Again, more on these concepts later.

Listing 2: dao/darts_match_dao.py

```
1  from tinydb import TinyDB, Query
2
3  import random
4  import threading
5  import time
6
7
8  class DartsMatchDao:
9
10     def __init__(self):
11         self.db = TinyDB('db.json')
12         self.lock = threading.Lock()
13         self.rand = random.random()
14
15     def add(self, match):
        self.lock.acquire()

        time.sleep(4)

        Match = Query()
        if not self.db.contains(Match.player1 == match.player1):
            self.db.insert({'type': match.type, 'player1': match.player1, 'player2': match.player2})

        print('Insert attempted on ' + match.player1 + '    ' + str(self.rand))

        self.lock.release()
```

5.2 The JSON Store

Have a look at the db.json file. It contains some initial documents, for example:

`{"type": "501", "player1": "Ali", "player2": "Jose"}`

We can also see that it looks like a key-value store with values like "1", "2", etc. as keys to values that are documents, e.g. the key "1" is associated with the value `{"type": "501", "player1": "Ali", "player2": "Jose"}`.

5.3 The Test Script

Adhoc Python test scripts are often written in a non-OO way (you could also use a dedicated test library for test-driven development, such as unittest). In the file test.py (see Listing 3), we import our custom classes from the modules (the Python scripts) in the dao and domain folders (lines 1 and 2). We create an instance of DartsMatchDao (line 4) and use its add function (on line 8) to add a new match (created on line 6) to the JSON store.

Listing 3: dao/darts_match_dao.py

```
1 from dao import darts_match_dao
2 from domain import darts_match
3
4 dao = darts_match_dao.DartsMatchDao()
5
6 match = darts_match.DartsMatch('501', 'Dupe', 'Dup1')
7
8 dao.add(match)
```

6 Exercises

1. In test.py add the following code to the end:

   ```
   dao2 = darts_match_dao.DartsMatchDao()
   match = darts_match.DartsMatch('501', 'Dupe2', 'Dup2')
   dao2.add(match)
   ```

   Run test.py. Is the timing of the output what you expected?

   Well, this is a single thread of execution, so the use of the lock works fine even though we have 2 different instances of the DAO class (the 2 random numbers tell us this). But what would happen if we had multiple threads of execution working in parallel? Let's find out in the next exercise.

2. Look at the file test_threads.py (see Listing 5). Run it. What do you notice about the timings now? What does it tell us might have gone wrong? Now we have 2 different instances of the DAO class (again the 2 randoms numbers tell us this) *and* two separate threads. Don't worry about the thread stuff too much now—basically it is almost like they are each running independently on their own separate little CPUs.

3. Let's fix the issue above. We don't want 2 separate threads getting into our data store at the same time because all kinds of chaos could ensue with lost updates, and so on. We only want one thread at a time to access the JSON file. The solution in darts_match_dao_thread_safe_singleton.py does two new things:

   a) It implements the singleton pattern to ensure that we always deal with one single instance of the DAO—we call the static method get_instance which only creates a new instance using the constructor if one does not already exist. The get_instance method is an example of a *factory* method because instead of using *new* or a constructor directly, we pass on that responsibility to a factory method and have the complexities of how the instance is created hidden from us. This gives us nice clean code.

   b) We again use locking in get_instance to ensure multiple threads can't access the creation of the instance at the same time, which could result in erratic behaviour.

4. Now when you run test_threads.py you should see timings that indicate that the second thread had to wait for the first thread to release the lock on the DAO add method. You should also see the random number matches in each of the insertion output messages, e.g.:

   ```
   $ pipenv run python test_threads.py
   Starting Thread-1
   ```

```
Starting Thread-2

Exiting main thread.

Insert attempted on Dupe 0.17329372035885504

Exiting Thread-1

Insert attempted on Dupe 0.17329372035885504

Exiting Thread-2
```

Listing 4: darts_match_dao_thread_safe_singleton.py

```python
from tinydb import TinyDB, Query

import random
import threading
import time


class DartsMatchDao:
    __instance = None

    @staticmethod
    def get_instance():
        if DartsMatchDao.__instance is None:
            with threading.Lock():
                if DartsMatchDao.__instance is None:  # Double locking mechanism
                    DartsMatchDao()
        return DartsMatchDao.__instance

    def __init__(self):
        if DartsMatchDao.__instance is not None:
            raise Exception("This is a singleton!")
        else:
            DartsMatchDao.__instance = self
        self.db = TinyDB('db.json')
        self.lock = threading.Lock()
        self.rand = random.random()

    def add(self, match):
        self.lock.acquire()

        time.sleep(4)

        Match = Query()
        if not self.db.contains(Match.player1 == match.player1):
            self.db.insert({'type': match.type, 'player1': match.player1, 'player2': match.player2})

        print('Insert attempted on ' + match.player1 + '     ' + str(self.rand))

        self.lock.release()

    def update(self, match):
        self.lock.acquire()

        self.lock.release()
```

Listing 5: test_threads.py: Two separate threads of execution

```
from dao import darts_match_dao_thread_safe_singleton, darts_match_dao
from domain import darts_match

import threading


class AddThread(threading.Thread):
    def __init__(self, thread_id, name, match):
        threading.Thread.__init__(self)
        self.threadID = thread_id
        self.name = name
        self.match = match
        self.dao = darts_match_dao.DartsMatchDao.get_instance()

    def run(self):
        print("Starting " + self.name)
        self.dao.add(self.match)
        print("Exiting " + self.name)


dart_match1 = darts_match.DartsMatch('501', 'Dupe', 'Dup2')
thread1 = AddThread(1, "Thread-1", dart_match1)

dart_match2 = darts_match.DartsMatch('501', 'Dupe', 'Dup2')
thread2 = AddThread(2, "Thread-2", dart_match2)

thread1.start()
thread2.start()

print("Exiting main thread.")
```

7 More Python

That's enough exploration of Python for now. There are many excellent online tutorials and books available as references. For example, `https://www.tutorialspoint.com/python/index.htm`.