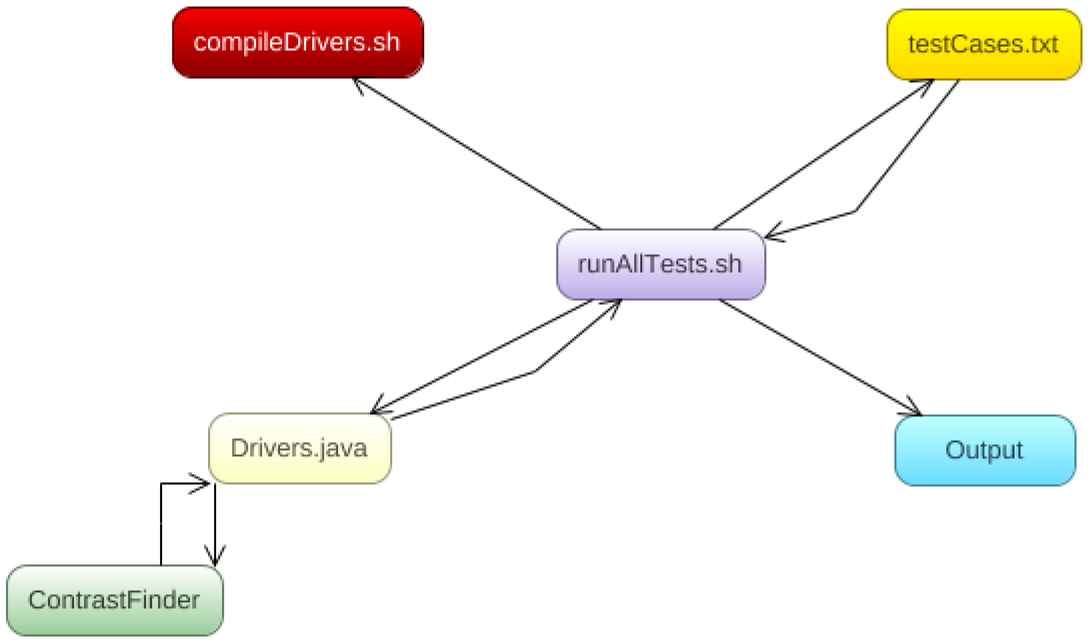
Chapter 3

Layout of our file structure, with diagram

```
http://localhost
├── DeliverableReports
│   ├── CargoPants_deliverable1.pdf
│   └── CargoPants_deliverable2.pdf
├── README.md
├── TestAutomation
│   ├── docs
│   │   ├── INSTRUCTIONS.md
│   │   ├── README.txt
│   │   └── tomcat
│   │       ├── README.md
│   │       ├── tomcat.service.bak
│   │       └── udo systemctl enable tomcat
│   ├── oracles
│   │   └── testCase1Oracle.txt
│   ├── project
│   │   ├── bin
│   │   └── src
│   │       ├── compiling.txt
│   │       ├── org
│   │       │   └── opens
│   │       │      ├── colorfinder
│   │       │      │   ├── AbstractColorFinder.java
│   │       │      │   ├── ColorFinder.java
│   │       │      │   ├── factory
│   │       │      │   │   ├── ColorFinderFactory.java
│   │       │      │   │   └── ColorFinderFactoryImpl.java
│   │       │      │   └── result
│   │       │      │      ├── ColorCombinaison.java
│   │       │      │      ├── ColorCombinaisonImpl.java
│   │       │      │      ├── ColorResult.java
│   │       │      │      ├── ColorResultImpl.java
│   │       │      │      └── factory
│   │       │      │           ├── ColorCombinaisonFactory.java
│   │       │      │           ├── ColorCombinaisonFactoryImpl.java
│   │       │      │           ├── ColorResultFactory.java
│   │       │      │           └── ColorResultFactoryImpl.java
│   │       │      └── utils
│   │       │           ├── colorconvertor
│   │       │           │   └── ColorConverter.java
│   │       │           ├── contrastchecker
│   │       │           │   └── ContrastChecker.java
│   │       │           └── distancecalculator
│   │       │                └── DistanceCalculator.java
│   │       └── test
│   │              ├── CalculateDriver.java
│   │              ├── ComputeContrastDriver.java
│   │              ├── Hex2RgbDriver.java
│   │             ├── Rgb2HexDriver.java
│   │             └── offsetRgbColorDriver.java
│   ├── reports
│   │   └── testReport.txt
│   ├── scripts
│   │   ├── compileDrivers.sh
│   │   ├── myList.sh
│   │   ├── runAllTests.sh
│   │   ├── runSingleTest.sh
│   │   └── testCase1Example.sh
│   ├── testCases
│   │   ├── testCase01.txt
│   │   ├── testCase02.txt
│   │   ├── testCase03.txt
│   │   ├── testCase04.txt
│   │   ├── testCase05.txt
│   │   ├── testCase06.txt
│   │   ├── testCase07.txt
│   │   ├── testCase08.txt
│   │   ├── testCase09.txt
│   │   ├── testCase10.txt
│   │   ├── testCase11.txt
│   │   ├── testCase12.txt
│   │   ├── testCase13.txt
│   │   ├── testCase14.txt
│   │   ├── testCase15.txt
│   │   ├── testCase16.txt
│   │   ├── testCase17.txt
│   │   ├── testCase18.txt
│   │   ├── testCase19.txt
│   │   ├── testCase20.txt
│   │   ├── testCase21.txt
│   │   ├── testCase22.txt
│   │   ├── testCase23.txt
│   │   ├── testCase24.txt
│   │   ├── testCase25.txt
│   │   └── testTemplate.txt
│   └── testCasesExecutables
│     └── MyClass.java
├── Wiki
│   └── Assets
│     ├── myList1.png
│     ├── myList2.png
│     └── myList3.png
├── myList.html
└── myList.sh
30 directories, 72 files
```



Driver Rationale

- **Rgb2HexDriver.java** - takes three arguments (rgb) and parses as ints, constructs new color object to be inserted into rgb2Hex method and prints output.
- **Hex2RgbDriver.java** - takes one argument (hex value) and uses Color.decode to construct a new color object to be inserted into hex2Rgb method and prints output.
- **ComputeContrastDriver.java** - takes two arguments (doubles) and parses as doubles, inserts the two arguments into the computeContrast method and prints output.
- **OffsetRgbColorDriver.java** - takes six arguments (rgb and offsets for each) and parses all as ints, constructs a new color object from the rgb values and inserts that as well as the three offsets into the offsetRgbColor method and prints output.
- **CalculateDriver.java** - takes either two or six arguments depending on if using hex values or rgb values to construct two new color objects to insert into the method Calculate

Scripts

- **compileDrivers.sh** – reads all of the drivers in our test folder and compiles them using the default Java compiler
- **runSingleTest.sh** – takes a filename as input, reads through the file, and tests a method as specified by our test case template with a detailed pass/fail output
- **runAllTests.sh** – runs compileDrivers.sh, forms an array that represents each defined test case file, and passes each individual filename to runSingleTest.sh

Experiences

Compiling the source code from the command line was difficult to understand, as maven has its package structure defined by various pom.xml files in separate modules. We had to manually combine these modules together for the base Java compiler to read and compile properly.

Once the source code could compile, we needed to figure out how to make our test drivers read from their source code and test the necessary methods. We had trouble finding out how to make our code play nicely with the Java package structure, but eventually we tried something which ended up compiling properly. We were extremely relieved with this accomplishment.

After this, we decided to move into Linux and start building our scripts. Collin studied some Bash scripting tutorials and ended up building several scripts which automate the tasks required to test our drivers. It took time, but was fairly intuitive.

After building a demo driver to run a single test case, we encountered some unexpected behavior when scaling it for different drivers. The test files were not being read correctly, despite following the template properly. After some research into how Bash works, we discovered this was due to line ending differences between Windows (where our test cases were written) and Linux (where we were running the tests). A workaround was to rewrite these test cases in our virtual machines, but in the future we can use git's built-in setting "autocrlf" which converts between the two conventions for us.

We made some additional errors while working on automation. Originally we had stored all of our testing and automation in a folder called "TeamAutomation" instead of "TestAutomation", so we had to refactor our file structure, causing some merge conflicts. We also had issues understanding where our scripts would run from, but eventually made it so they would only run from the TestAutomation folder, and nowhere else.

Finally, we have been unsure of how exceptions should be handled in our oracles. After some debate, we decided to use try/catch blocks in our drivers, and use Exception.getMessage() as the output to compare against the oracle. This has not been implemented to each driver yet.