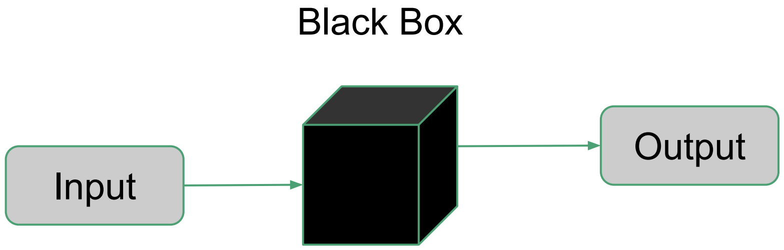
Scheduler Simulator

Vicente Adolfo Bolea Sánchez <vicente.bolea@gmail.com>

Index

(Business) Requirements

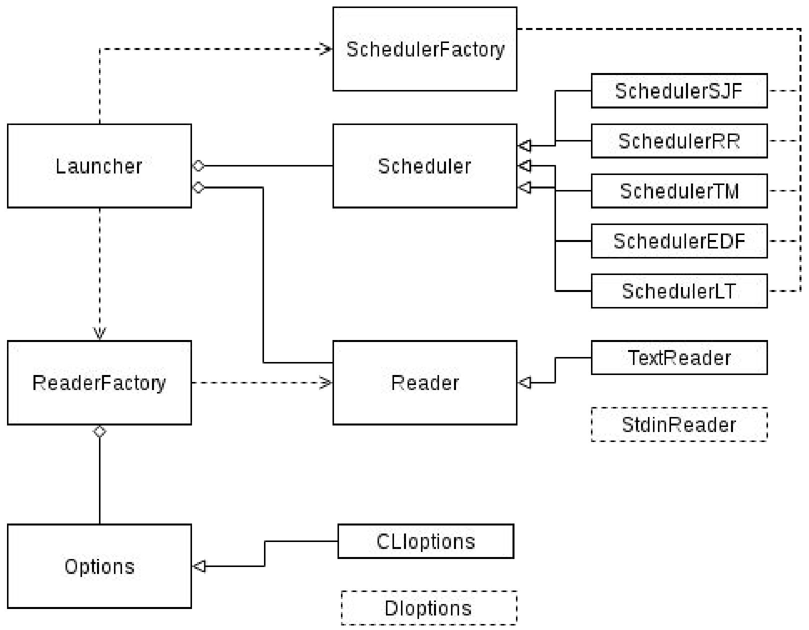
- Should read an input file in **three different** formats.
- Should select a scheduling algorithm in ***run-time***.
- Should accept **multiple flags** through the command line interface.
- Should output the output to *stdout.*



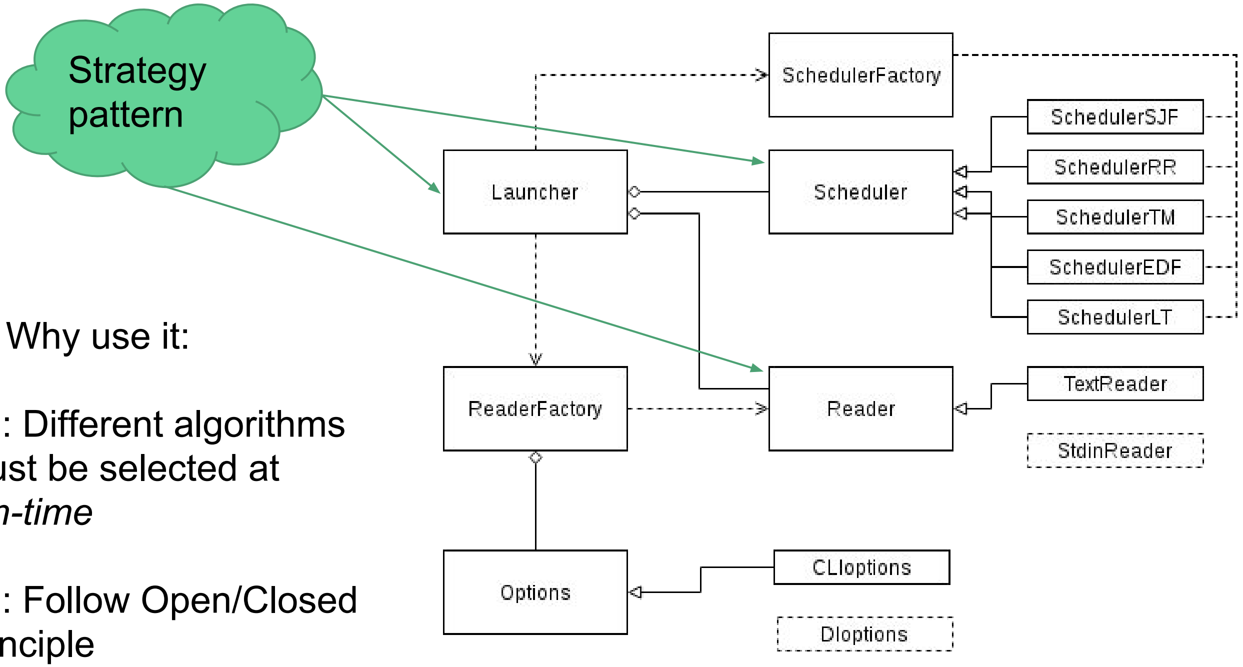
Technical requirements

1. No more than 8 hours (w/o debugging).
2. Generic code, it might be useful for my personal projects.
3. Linux (UNIX) platform.
4. Within the C++14 standard library.
5. Pseudo-XP workflow (Top-down variant)
   a. Iterative development.
   b. Test driven development.
   c. Feedback from integration tests.
   d. Discipline to test and refactor.

Class diagram



Class diagram

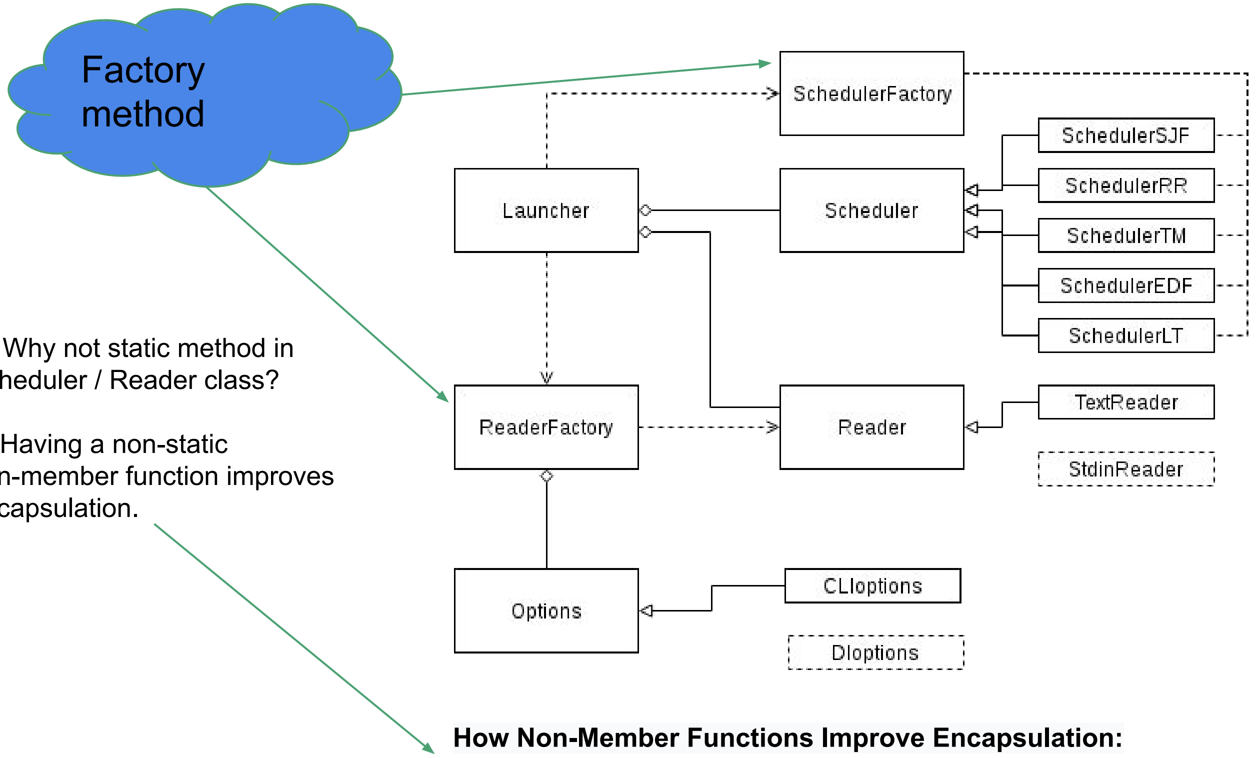


Q: Why use it:

A1: Different algorithms must be selected at *run-time*

A2: Follow Open/Closed principle

Class diagram



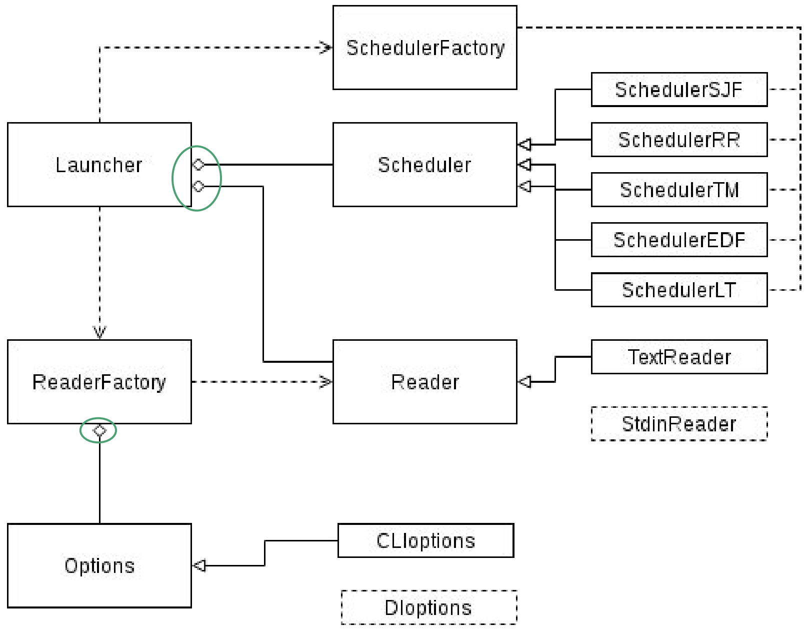
Q: Why not static method in Scheduler / Reader class?

A: Having a non-static non-member function improves encapsulation.

How Non-Member Functions Improve Encapsulation:

http://www.drdobbs.com/cpp/how-non-member-functions-improve-encapsu/184401197

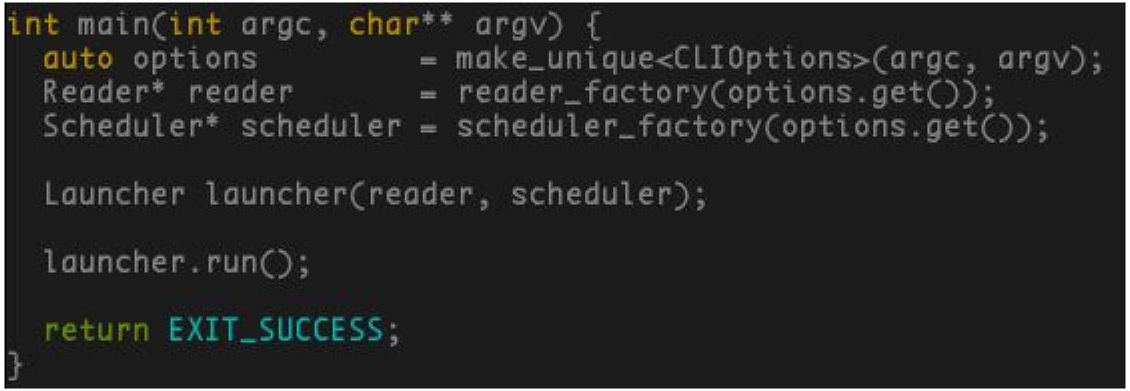
Class diagram



Inversion of Control

Dependency injection

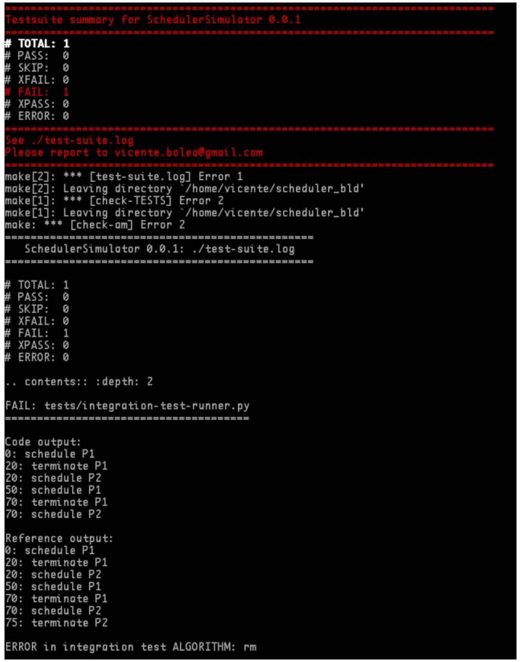
How the main file should look like?



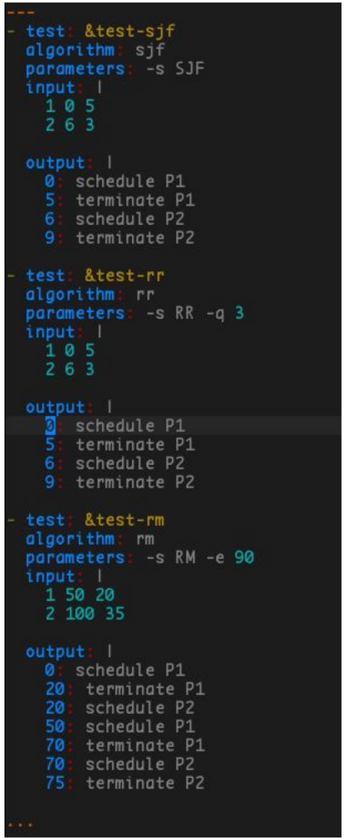
```
int main(int argc, char** argv) {
  auto options          = make_unique<CLIOptions>(argc, argv);
  Reader* reader        = reader_factory(options.get());
  Scheduler* scheduler = scheduler_factory(options.get());

  Launcher launcher(reader, scheduler);

  launcher.run();

  return EXIT_SUCCESS;
}
```

Test Driven Development

- Write first the test, then implement the feature.
- Work best when you have specific test cases that the software must satisfy.
- I used integration tests in this case (The project is small, <1K lines).



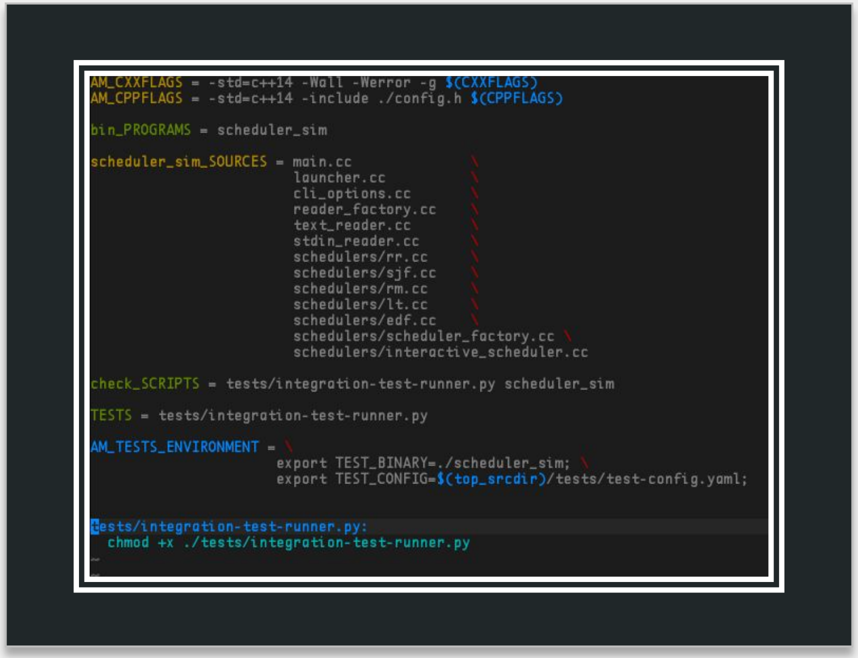
```
============================================================================
Testsuite summary for SchedulerSimulator 0.0.1
============================================================================
# TOTAL: 1
# PASS:  0
# SKIP:  0
# XFAIL: 0
# FAIL:  1
# XPASS: 0
# ERROR: 0
============================================================================
See ./test-suite.log
Please report to vicente.bolea@gmail.com
============================================================================
make[2]: *** [test-suite.log] Error 1
make[2]: Leaving directory `/home/vicente/scheduler_bld'
make[1]: *** [check-TESTS] Error 2
make[1]: Leaving directory `/home/vicente/scheduler_bld'
make: *** [check-am] Error 2
================================================
   SchedulerSimulator 0.0.1: ./test-suite.log
================================================

# TOTAL: 1
# PASS:  0
# SKIP:  0
# XFAIL: 0
# FAIL:  1
# XPASS: 0
# ERROR: 0

.. contents:: :depth: 2

FAIL: tests/integration-test-runner.py
======================================

Code output:
0: schedule P1
20: terminate P1
20: schedule P2
50: schedule P1
70: terminate P1
70: schedule P2

Reference output:
0: schedule P1
20: terminate P1
20: schedule P2
50: schedule P1
70: terminate P1
70: schedule P2
75: terminate P2

ERROR in integration test ALGORITHM: rm
```



```
---
- test: &test-sjf
  algorithm: sjf
  parameters: -s SJF
  input: |
    1 0 5
    2 6 3

  output: |
    0: schedule P1
    5: terminate P1
    6: schedule P2
    9: terminate P2

- test: &test-rr
  algorithm: rr
  parameters: -s RR -q 3
  input: |
    1 0 5
    2 6 3

  output: |
    0: schedule P1
    5: terminate P1
    6: schedule P2
    9: terminate P2

- test: &test-rm
  algorithm: rm
  parameters: -s RM -e 90
  input: |
    1 50 20
    2 100 35

  output: |
    0: schedule P1
    20: terminate P1
    20: schedule P2
    50: schedule P1
    70: terminate P1
    70: schedule P2
    75: terminate P2

...
```

Building system: GNU/Autotools

1. Standardize the installation process of the system (make install)
2. Contains a test harness to enable TDD (make check)
3. Relatively simple for small projects.
4. No need to be platform independent.



```
AM_CXXFLAGS = -std=c++14 -Wall -Werror -g $(CXXFLAGS)
AM_CPPFLAGS = -std=c++14 -include ./config.h $(CPPFLAGS)

bin_PROGRAMS = scheduler_sim

scheduler_sim_SOURCES = main.cc                    \
                        launcher.cc                \
                        cli_options.cc             \
                        reader_factory.cc          \
                        text_reader.cc             \
                        stdin_reader.cc            \
                        schedulers/rr.cc           \
                        schedulers/sjf.cc          \
                        schedulers/rm.cc           \
                        schedulers/lt.cc           \
                        schedulers/edf.cc          \
                        schedulers/scheduler_factory.cc \
                        schedulers/interactive_scheduler.cc

check_SCRIPTS = tests/integration-test-runner.py scheduler_sim

TESTS = tests/integration-test-runner.py

AM_TESTS_ENVIRONMENT = \
                      export TEST_BINARY=./scheduler_sim; \
                      export TEST_CONFIG=$(top_srcdir)/tests/test-config.yaml;


tests/integration-test-runner.py:
  chmod +x ./tests/integration-test-runner.py
~
~
```

Extra features

1. StdinReader to ease the integration tests
2. All the abstractions to enable future changes to the project
3. Extra scheduling algorithm not chosen yet :(

Demonstration

Thank you!