Scheduler Simulator

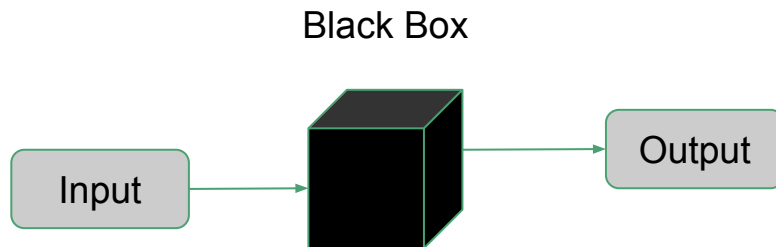Vicente Adolfo Bolea Sánchez <vicente.bolea@gmail.com>

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.*

Black Box

Input → Output

Technical requirements

1. No more than 8 hours.
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.
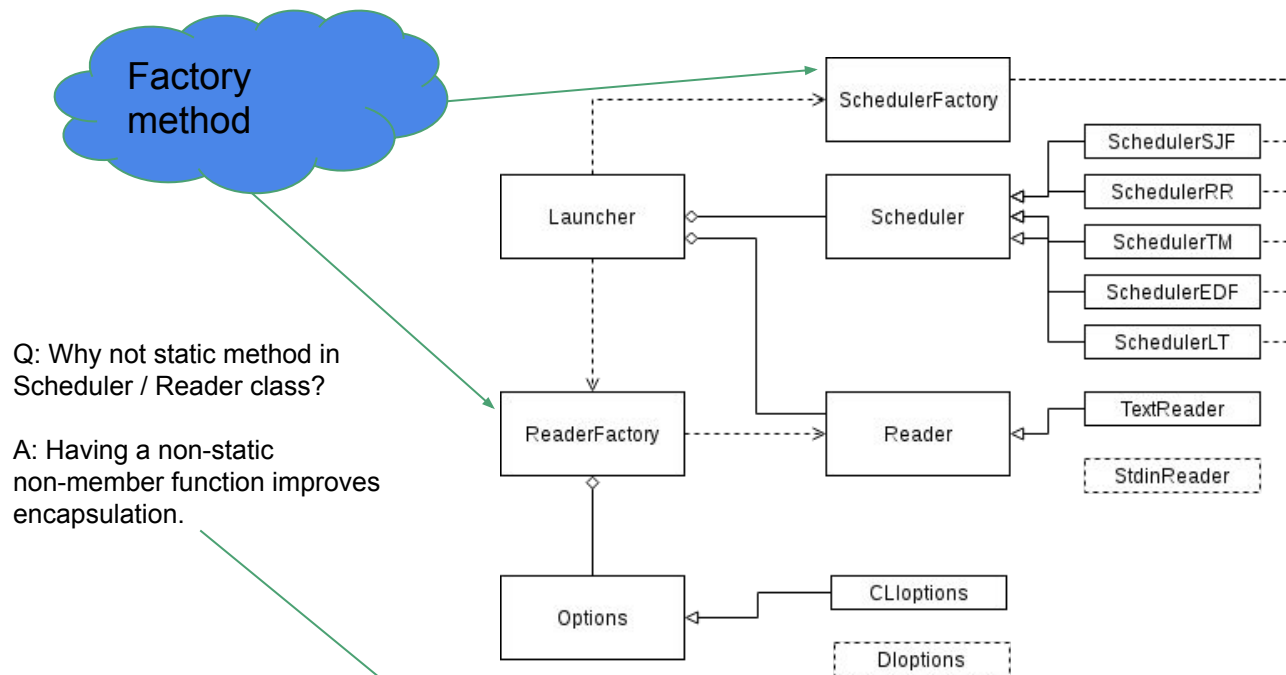   d. Discipline to test and refactor.

Class diagram

Class diagram

Strategy pattern
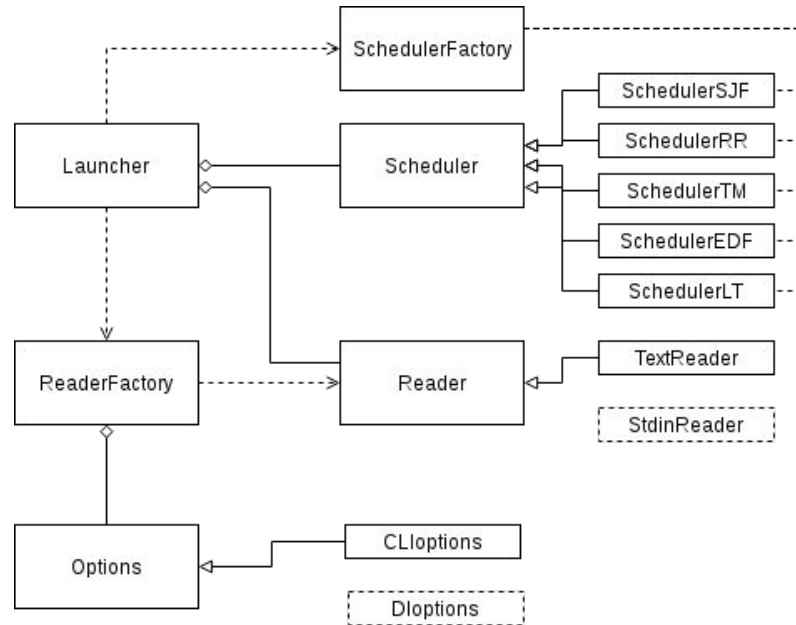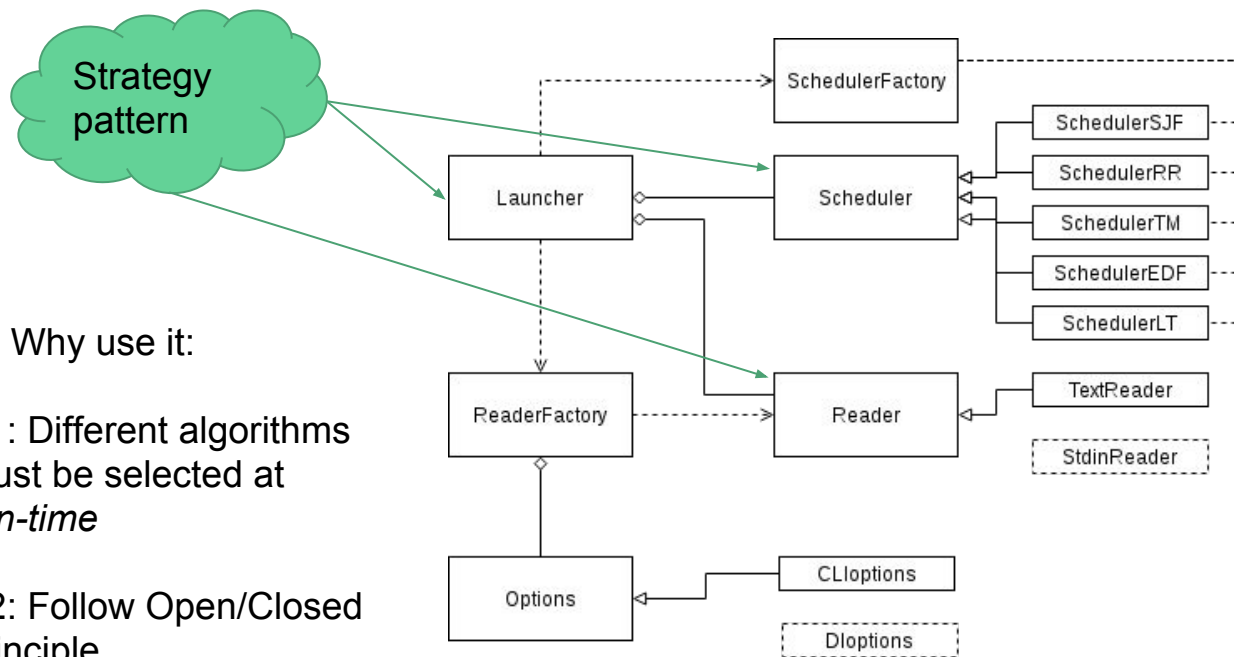
Q: Why use it:

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

A2: Follow Open/Closed principle

SchedulerFactory

SchedulerSJF

SchedulerRR

SchedulerTM

SchedulerEDF

SchedulerLT

Launcher

Scheduler

ReaderFactory

Reader

TextReader

StdinReader

Options

CLIoptions

DIoptions

Class diagram

Factory method

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

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

In diagram:
- SchedulerFactory
- SchedulerSJF
- SchedulerRR
- SchedulerTM
- SchedulerEDF
- SchedulerLT
- Launcher
- Scheduler
- ReaderFactory
- Reader
- TextReader
- StdinReader
- Options
- CLIoptions
- Dloptions

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
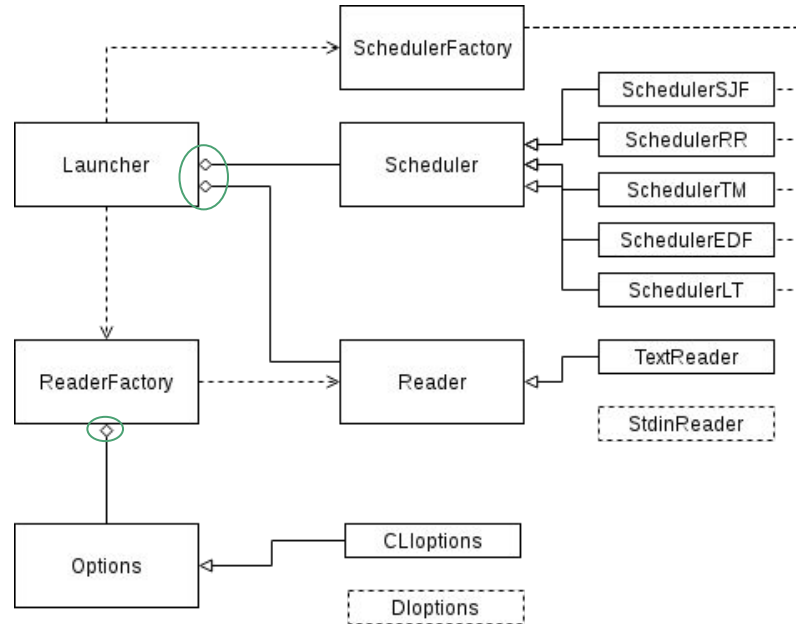
How the main file should look like?

```cpp
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).

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.

Extra features

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

Demonstration

Thank you!