Object-Oriented Design and Programming 2021 – Coursework

The objective of this coursework is to practice the development of moderately complex class systems and the appropriate use of ownership concepts to avoid memory leaks and memory corruption.

Coffee Roasting

Your task is to build the data backend for a web application supporting a coffee roastery. Coffee roasting is an intricate process that merits a brief discussion. The two key steps are blending and roasting.

Figure 1: The Gene Café Home Coffee Roaster

Blending

When performing a roast, the roaster (the person operating the machine) first selects beans of different types and mixes them. A roast is, thus, comprised of several "Ingredients", i.e., a specific amount of beans from a given origin (we name beans after their origin). You may assume that a roast does not contain two ingredients with beans from the same origin.

Roasting

The roasting process turns a mix of green beans with little taste into the brown beans we like to brew. The roaster starts the process at a specific time (measured in Unix time, i.e., seconds since 1 Jan 1970). In the API, we use the starting time to identify a roast, preheats the roasting machine, adds the ingredients and empirically tracks the process. During the roasting process, different kinds of events can occur arbitrarily many times:

- The roaster sets the temperature to a specific value
- The roaster observes a specific value on the builtin thermometer
- The roaster fills the machine with beans
- The beans turn from green to brown (an event called browning)
- The beans crack (much like the popping of popcorn, though without popping the bean open)
- The roaster drops the beans (removes them from the machine)

While in reality, some of these events (like browning or dropping) can only occur once, the application does not model that fact.

Roasty

Roasty is a web application to track the roasting process. It has a built-in web server and a javascript-based GUI. For this coursework, we have built Roasty and stripped out the data model component.

The model is built around the concept of a roast (which is identified by a timestamp, i.e., no two events have the same timestamp). A roast contains ingredients that, in turn, refer to a bean and have a set amount. A bean merely has a name. Note, while an ingredient can be removed from a roast, the bean type remains in the database (wink, wink, nudge, nudge). The roast further contains events that have a type (simply a string), a timestamp and an optional event value (which merely contains a numeric value).

Your task

Your task is to provide that data model. On the one hand, you need to analyze the problem and decompose it into a class structure. On the other, you need to implement the required behavior for the classes you designed. To guide you in the right direction, Roasty comes with an extensive suite of tests (little functions that are run to verify individual pieces of behavior). The tests define the required API – separation of concerns at its best. Once you pass all the tests, you can run the Roasty server, open your web browser and start tracking your coffee roasts :-).

Because Roasty is a comparatively simple application, the data model will live in two files (we do not separate code into one file per class). Those files are pre-created in your project repository: `Source/Model/RoastyModel.hpp` and `Source/Model/RoastyModel.cpp`. Do not modify any other files (unless advised by me, i.e., Holger).

Getting started

To get started, log in to a lab machine of your choice and run the following sequence of commands:

```
git clone ${your repository URL from LabTS}
cd ${your repository name}
```

You should set up a separate directory underneath the project binary for the compiled binaries. Here is how you do that:

```
mkdir Build
cd Build
cmake -DCMAKE_BUILD_TYPE=Debug ..
```

If you prefer the clang compiler instead of gcc you can (on the lab machines) replace the last line with the following (clang tends to produce better error messages):

```
CXX=clang++ CC=clang cmake -DCMAKE_BUILD_TYPE=Debug ..
```

You can compile the project by typing

```
cmake --build .
```

Note that the first build of the project will take a long time (and produce a lot of output) since it also builds dependencies.

Testing

To run the tests, simply run

```
./Tests
```

a successful run output should look like this (pass -? for more options)

```
===============================================================
All tests passed (101 assertions in 12 test cases)
```

(if you have fewer assertions, the framework probably skipped some because others have failed)

Running the server

To run the roasty server, run the following sequence

```
./Roasty
```

the output should look like this:

```
open http://localhost:8000 to access the application
press ctrl+c to quit
```

At that point, you can open `http://localhost:8000` in your browser and enjoy the result of your labor.

Submission

The coursework will be submitted using LabTS. Important are two files mentioned files:

- `Source/Model/RoastyModel.hpp` and
- `Source/Model/RoastyModel.cpp`.

Make sure all your implementation is in those files and no other files are modified. The markers will not take anything else into account.

Use what you have learned in class so far. Do not use templates or inheritance.

Marking

The marks are distributed as follows:

- Correct implementation of the Roasty API (note that passing tests are required but not necessarily sufficient for a correct implementation): 30%
- Memory-correctness of the tests: 30%
- Reasonable performance (no silly performance bugs, appropriate use of references to avoid copies, etc.): 20%
- Clean code with appropriate documentation (remember Goldilocks: not too little, not too much – do not document *what* you have done, but *why*): 20%

Note that passing tests on LabTS are required. It is not enough if they pass on your machine.