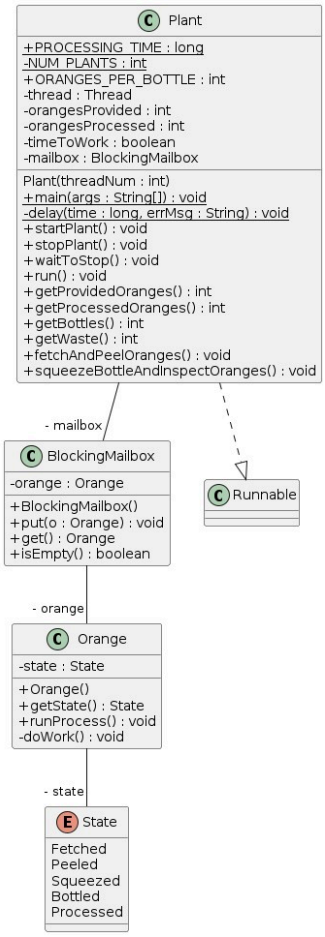
UML Diagram



Description

As implemented in the Moodle example, oranges are picked up and take a finite amount of time to fetch an orange, peel it, juice it, bottle it, and complete.

Two plants each run in parallel and each plant has two workers, one handling fetching and peeling and another handling juicing, bottling and completion.

There are seven threads, the main thread which starts 2 plants, which themselves start 2 worker threads each.

Challenges

The first challenge I faced was determining how to make one new thread handle one set of tasks and another handle the other. I solved this using this thread creation code:

```
fetchAndPeeler = new Thread(this::fetchAndPeelOranges, "Fetcher");
```

which assigns the thread a function to run when it's called.

That handled task parallelization but I ran into issues when the threads were supposed to stop, since `BlockingMailbox` blocks indefinitely on `put()` and `get()`. I handled this by daemonizing the threads and not waiting for them to `join()`.