

# Design Pattern Assignment - Source Code

## Background

It's been a while since your last work with Permanent Assurance Company, and the FEAA/ERP system you helped them with has not been treated well. The company has taken over the Very Big Corporation of America, and in the process the system has seen significant change and growth. Many consultants have come and gone on the system, and there have been several "rush" jobs that focused on shipping code that worked and not on code that worked (or was designed) well.

The end result is a system that works... sort of. All of the functional requirements are met, and the system mostly delivers what the newly named Crimson Permanent Assurance (CPA) front-end operators need. However, it is very slow in some places (causing non-functional tests to fail), and any time CPA asks someone to add a new feature or fix the lag issues they run away in terror.

CPA have now asked you to come in and fix one of their key components, but since money is rather tight, they have placed restrictions on your work:

- Several 'module borders' in the system exist – you may not change anything beyond the module border. These modules are pretend abstractions of legacy software the organisation has lost the source code for and does not have the time or money to redevelop.
- CPA has the existing API for these modules and has given you a 'pretend' version of each module for each to test with.
- Your solution will be run through a test suite (provided to you as well) to ensure the module interactivity remains intact at the end, and to ensure your delivered code still responds in precisely the same functional way as before.

Thankfully, the module you will be redeveloping was behind a façade just like your original design – so besides a few public interfaces you can change anything you like about its internal workings without worrying about the view not being able to handle the changes.

You should **use proper red-green-refactor practices when refactoring** this code. The code currently passes function-based testing and should continue to do so the entire time - no breaking changes!

## Work Required

You must dig into and correct the design and implementation of the FEAA package along with all involved classes. CPA has observed the following key issues they would like you to investigate and solve:

- The system **uses a LOT of RAM**. Analysis has indicated this is due to the **Report class**, which stores a lot of data. CPA would like you to **solve this RAM issue** somehow, without breaking the existing use of the Report interface. **ReportImpl** has been included in your module scope to assist with this, but ReportDatabase is a fake façade on a remote database that you cannot change.
- There are several types of accounting service Orders. The current solution for these orders is to create a new class for each type (based on work type e.g. **audits** or **day-to-day** work, **whether the order is for priority client**, and whether the order is a **one off or regularly scheduled** work). The full system has  $66 * 2 * 2$  of these classes (**264 order classes**), with 8 of these ( $2*2*2$ ) provided to you as an example – CPA would like you to find a way to **reduce this class load** without breaking the **existing Order interface**.
- The current method of **handling client contact methods** is quite bulky – CPA would like you to **streamline** this somehow.
- Any time **Clients are loaded from the database**, the system lags for a long time. The database issues themselves have been deemed too expensive to fix, but perhaps you can **partially mitigate this** with the software somehow?
- Because the Report object captures data without any consistent primary key, and because people have duplicated object names and versions, any time reports need to be **compared for equality** we have to remember to **check many fields**. CPA would like you to **make this process simpler**.
- The **Order creation** process involves a lot of **slow database operations**. CPA would like you to **simplify this process** (especially the database lag while the employee is entering data) without breaking the **Order interface**.
- The current system is mostly **single-threaded**. There has been some work on the database side to allow multithreading, but as yet the FEAA module does not have any threading besides the main one. CPA would like you to **use multithreading** to allow employees to use the system for other things while slow database processes happen in the background.

As well as these they would like you to **clean up the code and document** where you believe it is necessary (for both the existing code and your changes).

## Detailed Allowed Scope

- The ONLY package you are allowed to modify is `au.edu.sydney.cpa.erp.feaa` and contents (**`feaa.ordering` and `spfeaa.reports`**) - these are the 'in-scope' classes. No, it doesn't make much sense for ordering and reports to be inside feaa - this is so it is easier for you to know what you are and aren't allowed to change.

- You may not modify feaa.reports.ReportDatabase (it's not the real database and modifying this won't help CPA).
- You may not replace the uses of auth, contact, database, ordering, or view packages - the test suite will enforce some of this
- You may add, remove, merge, change, etc any of the in-scope classes, so long as the **public api of the feaa package remains the same** - e.g. FEAAFacade must still exist in the same way so the view can call its methods, etc.

## Assessment Notes

- You will be assessed on your code quality, your use of design patterns, and keeping to the requirements of the client.
- The most important thing to remember is *don't break production code* - **If your code submission does not pass the test suite provided the maximum mark will be 50% if all other requirements are perfect.** Run the tests after each small change you make (that's what they're there for) and **use version control** to ensure you are able to roll back any breaking changes you can't fix.
- There will be a first demo due at the end of Week 11 for you to show a minimum of 1 of the above problems having been solved, and the code still passing the provided test suite – remember Red-Green-Refactor, you should not make sweeping changes to existing code without checking it passes tests as you go.
- There will be a final demo in partway through Week 13 for you to show however many problems you have solved and again with your code passing the test suite - at this point you may not edit the code further and should focus on your written Report.
- There are many ways to solve the given problems – for best marks you should prioritise methods that use the design patterns you have been taught in this unit.
- Marking will be focused on your ability to solve these problems with good, clean, patterned code, over and above solving 'all' of the problems. **You will get much better marks for solving 50% of the problems completely and with good solutions than you will for solving 100% of the problems with poor quality code.** Ensure you **focus your efforts!**
  - The multi-threading issue in particular you **should not attempt until all other issues have been solved adequately**
  - There will be times when the modules you can't change force you to do things that **break OOP design principles** – this is perfectly normal. Make sure it is actually required, then **document where and why you have done so.** Patterns may help you limit the negative effects this has on the rest of your code.
- Ensure you use versioning software so you can roll-back any changes that break tests when it comes time to demo and submit.

## Resources

[CPACodebase.zip](#)

## Submission Requirements

- You must submit a **zip folder** containing your complete FEAA application as a **Gradle Project**.
- To achieve the compilation & run marks your project **MUST** compile and test properly with 'gradle build' without requiring any modification
  - This means any locally stored dependencies you decide to add must be included in the zip folder and given a relative reference in your build.gradle - in general avoid this as **work the libraries do for you will not earn you marks**
  - Code which does not compile and run is likely to be penalised in other areas as well (e.g. you haven't used design patterns to solve issues well if your code does not compile).
- When running your code through the test suite for marking the 'out of scope' classes such as TestDatabase or the test suite itself will be swapped out with known original sources - be careful not to edit these accidentally as your edits will NOT carry over and this may cause your code to fail testing.
- You must **include a readme file** (a simple text file, either .txt or .md) listing what CPA issues you have solved and where in the code you did so. If you have made any **assumptions** in your work, or if your code **requires** anything in particular (like external dependencies/libraries) you should **list them** here also.

## Final Notes

- This is a new codebase with some complexity. As with any such code, **there is a high chance of functional bugs existing in the codebase or test suite**. If you do notice a functional bug, please post this on Ed - either a fix will be given and the code updated, or you will be allowed to ignore it (non-functional bugs are intended, and you should fix these yourself).
- The test suite is there to guide you, not guarantee correctness. It is entirely possible to code something that directly hits the limited test cases specified but would fail if those test cases were changed to anything else - this will not yield a good result. Treat the combination of the test suite, the existing code, and the (sparse) comments as the specification for this code - if anything is unclear you should ask on Ed before changing any behaviours you might notice through the facade.

Design Pattern Assignment - Report

## Requirements


CPA needs about 100 similar modules redeveloped, so they would like you to write up a document they can use to develop a training workshop for their own developers. This document needs to cover the following:

- What **issues** did you identify in the existing code that caused CPA's problems?
- What **solutions** did you use?
- Where? What is the '**before and after**' of your solution?
- What do those solutions provide to the module overall (positive and negative)? Both in terms of **utility** (speed, etc.) and in terms of **OOP design**.
- **How** did you develop them? Particularly how did you ensure the exact overall function was maintained?

This document should be professionally written as a report to CPA's CIO John Goldstone **using the template provided**. You should repeat the required sections of the template for each problem you solved. For any problems you did NOT solve, you should still discuss these as best you can (what were the causes, etc?). You should include **code style & commenting, general OO principles** and anything else you have done in your report, not just design patterns. What you must NOT do is treat the report as a 'what if' - focus only on what you have actually done, and with your **examples stick to what was actually submitted**.

You should ensure you cover these problems to the **appropriate level of detail**, but without copying out textbook definitions of patterns, OO principles, etc. Stick to the specific questions being asked - so if you used e.g. the AbstractFactory pattern to solve an issue, *don't* just write about what AbstractFactory is and what general benefits/drawbacks it has. **Talk specifically about what problem you are solving with AbstractFactory, how you maintained** the overall function, and **(most important) what benefits/drawbacks** your use of AbstractFactory provides to CPA's own code.

**A page (~4-500 words) per problem** plus a page each for **overall OO principles** and **refactoring method** would likely be the right length, *if* no unnecessary information is included. You may also wish to include diagrams like UML class diagrams or charts such as performance tests (time, total RAM used, etc) to support your writing.

[A3 Report template.docx](#)  You do not need to use this file directly (for example if you would prefer to use latex), but you **MUST** follow the exact headings as given.