CS5722: Software Architecture

Team-Based Project

Spring Semester 2021 – 2

J.J. Collins

14th February 2022 (Week 4)

Version 1.0

1. Objectives:

1. Critique the use of architecture and design patterns as a mechanism for supporting quality attributes.
2. Implement the interceptor architectural pattern and a subset of the design patterns in the Gang of Four text - Design Patterns: Elements of Reusable Object-Oriented Software.

2. Specification

You are required to design, implement, and test a framework that will be used by 3rd party developers. Sample frameworks could target the games domain, order processing, etc.

- Create a briefing document that specifies the requirements for a framework and include one or more use case diagram(s) with light weight sample semi-structured use case descriptions. This document should also contain a listing of quality attributes with brief descriptions. For example, portability, extensibility, performance, and security. Discuss how these architectural use cases might be supported.
- Design, implement, and test a framework that illustrates the consequences of the Interceptor architectural pattern and 5 design patterns from CS5722.
    - You are not allowed to **COUNT** the Observer, Decorator, Singleton, State, and Iterator design patterns as part of the 5.
    - The Factory Method and Composite – both covered in CS5721, will count if used.
    - You must have at least one creational, structural, and behavioural design pattern.

- **Two additional patterns** must be researched and incorporated into the implementation. They can be enterprise and/or architectural patterns; and must **Not be design patterns**, and Not be from the set of patterns covered in CS5721 or CS5722 lectures.

- The system must have at least four packages.
- The project should demonstrate "added value" if aiming for an A grade. Examples include:
  - Use of code metric tools to evaluate "quality" of implementation
  - Use of refactoring tools to identify bad code smells and their resolution through supported refactoring techniques.
  - DevOps
  - Microservices
  - Rest / GraphQL
  - Concurrency.
  - Security
  - Use of frameworks, for example the Entity Component API
  - Others such as languages, tools, libraries, etc.

  Include coding fragments and/or screen shots with brief descriptions in this section of the report to illustrate implementation of Added Value.

  Databases and GUIs are core components in any application but are not the focus of this module and therefore will not be counted.
- Use of Github for version control is mandatory and not counted as Added Value
- Evidence of testing and critique is critical.

Sample Scenario

Your team has been commissioned by a client to create a framework for a multiplayer maze playing game i.e. Pacman++. The engine will incorporate an autonomous play mode in that a human can play against the machine. Minimum requirements are:

1. Game characters should register with the games engine so that any significant state changes of interest can be propagated to all the relevant players who have signed up for notification. Players can also belong to teams, with each member having the capacity to notify fellow team members of its health status. Use the Mediator.

2. Autonomous game characters are configured with a Strategy for learning to play better over time such as Evolutionary Algorithms or Artificial Neural Networks.
3. The framework will support mazes but does not know what mazes will be deployed at runtime. The framework should use a *creational* pattern such as factory method(s).
4. The framework should maintain snapshots of state. Use the Memento design pattern.
5. Use the Command design pattern to process user input.
6. The Interceptor architectural pattern is used to support extensibility.

ALL PATTERNS MUST BE INTEGRATED. It is not acceptable to provide a stand-alone implementation for each pattern!

3. Documentation

The submission must contain in the following order:

1. Front cover with scenario title, student names and IDs, and module details.
2. Table of contents.
3. Requirements
    - A brief outline of the scenario chosen for the framework.
    - Use Case diagram, and one or two sample Use Case Descriptions.
    - Discussion on quality attributes.
    - Discussion on tactics selected to support architectural use cases. See the chapter on tactics in Software Architecture in Practice by Bass et al. (2010).
4. A brief discussion of the Interceptor architectural pattern, 5 design patterns, and the two independently researched pattern. Emphasis should be placed on the two independently researched pattern.
5. A discussion on the architecture of the system illustrated with one a diagram that presents an architectural view.
6. A structural and behavioural diagram:
    - A class diagram that also shows the allocation of classes to packages.
    - An interaction (communication or sequence) diagram for a key use case.
7. Fragments of the code to illustrate "interesting" elements of the implementation. For example, coding fragments to show the implementation of the architecture and design patterns.
8. Visualisation from source control platform such as Github/Gitlab.

9. Documentation ref "added value".
10. Evidence of testing.
11. Discussion of problems encountered, solutions attempted that failed to resolve said problem with supporting evidence, etc.
12. Evaluation/Critique of support for relevant Non-Functional requirements (NFRs) through the patterns selected. This should be done using scenarios.
13. References
14. A description in FULL of the contribution of each team member to the implementation and report in the form of tables similar to that in CS5721.

4. Submission Guidelines.

- You can undertake this assignment in teams of **four**. Please seek permission for a team of three or five.
  - Email lecturers with subject "CS5722 Team" a.s.a.p with name and id of team members. Include the preferred name for the team.
  - If you are not in a team by Wednesday Week 4, please email the lecturers with a request to be allocated to a team. Subject must be "CS5722 Team".
- This assignment **constitutes 50%** of the total marks awarded for this module.
- **An F grade for this project will automatically result in an F grade for the module. Likewise for an NG!**
- **Submission deadline 16:00 Friday 22/April/2022 (UL Week 13 / Teaching Week 12)**
- The programming language used is at your discretion, but it must be Object-Oriented (OO).
- You may be required to give a code walkthrough and demo towards the end of the semester. Failure to do so will result in an F grade for the assignment.
- A GUI is not required, but a text-based screen displaying sequence of invocations is necessary
- If the presented work is not to the standard expected at this level, the submitted work will be heavily penalised:
  - Layout conforms to specification.
  - Page numbers in table of contents and throughout report.
  - Spell and grammar checks done.
- Accidental loss of work will not be accepted as an excuse.

- **Plagiarism will be reported to the disciplinary board without exception.**
- Usual caveat applies to teams:
  1. Problems with group dynamics to be reported immediately to lecturer.
  2. Individual grades will be given to each team member that may differ depending upon estimated understanding of and effort invested in creating submission.
  3. **IT IS EXPECTED THAT EACH TEAM MEMBER WILL CONTRIBUTE EQUALLY TO THE PROJECT - BOTH CODING AND DOCUMENTATION**.

- WORK CLEVER, WORK HARD
- PROMISE LESS DELIVER MORE
- KEEP IT SIMPLE

5. Reusing the CS5721 Project.

You may use the code developed in CS5721 project. In that case, the Added Value section will be a discussion on Refactoring with the following subheadings:

- Diagram before Refactoring
- Bad Code Smells
- Refactorings Applied
- Diagram after Refactoring

Discuss with the lecturer before starting. This approach is sometimes problematic.

6. Meetings

- 3 online meetings during the semester will be scheduled for each team. More info later in Week 4.

Grading Rubric

Beginning [C3-C2]	**Developing [C1-B2]**	**Accomplished [B1-A2]**	**Exemplary [A1]**
Poor supporting documentation	Supporting documentation is satisfactory	Supporting documentation is good but not complete	Supporting documentation is comprehensive & has utility
< 5 patterns implemented	5 Patterns implemented, some are a little trivial w.r.t implementation	5 Patterns implemented, a few being the more challenging covered in lectures	5 Patterns implemented, a few being the more challenging covered in lectures
Interceptor not implemented or poorly attempted	A reasonable attempt made to implement the interceptor	The interceptor is implemented	2 interceptors implemented, one using Chain of Command
7th pattern trivial or missing	7th pattern trivial.	7th pattern has depth.	7th pattern is challenging and has depth.
Intent is not explicit in code	Intent is frequently not explicit in code	Intent is not always explicit in code	Intent is always explicit in code
Does not understand the code	Understands most of the implementation	Good understanding of the code	Excellent understanding of every line of code
Added Value missing	Added Value partially present or poorly implemented	Added Value present but lacks depth	Added Value present and has depth and relevance i.e the WOW factor
Testing missing or not automated	Testing missing or not automated	Testing automated	Testing automated
Evaluation is superficial	Evaluation lacks depth	Evaluation is good	Evaluation is comprehensive
Poor team dynamics, weak collaboration	Satisfactory team dynamics and/or weak collaboration	Good team dynamics, good collaboration, good division of labour	Excellent team dynamics, excellent collaboration, excellent division of labour