Lab 02: Pattern Documentation & GoF Format

Week 2 Assignment

Student Name: _____ Date: _____

Overview

Total Points: 100 (50 points per lab)
Required Reading: *Learning JavaScript Design Patterns, 2nd Ed.* by Addy Osmani
Chapter 3: "Structuring and Writing Patterns" (Sections: "The Structure of a Design Pattern", "Well-Written Patterns", "Writing a Pattern")

Key Vocabulary

Before beginning, ensure you understand these terms from the book:

- **Pattern Structure**: A formal pattern description must include Context, Problem, and Solution (Ch. 3, "The Structure of a Design Pattern").
- **Context (Prerequisites)**: The situation where the pattern applies, including prerequisites and environmental constraints that must be present for the pattern to be applicable.
- **Problem (Forces)**: The specific design problem or forces the pattern addresses, clearly articulating what we are trying to solve and the constraints we face.
- **Solution (Implementation)**: The concrete implementation approach, including the structure, participants, and collaborations between objects.
- **GoF Format**: The classic Gang of Four format includes Context, Problem, Solution, Consequences, Related Patterns, Illustrations, Known Usage, and Discussions (Ch. 3, "Writing a Pattern").
- **Consequences**: Trade-offs and results of applying the pattern, including benefits and liabilities.
- **Related Patterns**: How this pattern relates to others in the catalog, including patterns it can be combined with or patterns it replaces.
- **Well-Written Pattern**: A pattern that provides substantial reference material, evidence of necessity, and helps developers identify patterns in code (Ch. 3, "Well-Written Patterns").

1. Lab 2.1: Pattern Documentation Architect (50 Points)

Objective

Write a formal pattern specification following the GoF (Gang of Four) format described in Chapter 3. You will document a utility function pattern that you've identified in code, demonstrating your understanding of pattern structure and academic documentation standards.

Background

According to Chapter 3, Section "Writing a Pattern", well-written patterns follow a standard structure that enables the development community to understand, evaluate, and apply them. This lab requires you to document a pattern using the formal structure from the textbook.

Provided Code Pattern

You are given the following JavaScript utility function that implements a retry mechanism for asynchronous operations:

Listing 1: Retry Utility Function

```
// retryUtility.js
async function retryOperation(operation, maxRetries = 3, delay = 1000)
    {
     let lastError;

     for (let attempt = 1; attempt <= maxRetries; attempt++) {
          try {
               const result = await operation();
               return result;
          } catch (error) {
               lastError = error;
               if (attempt < maxRetries) {
                    await new Promise(resolve =>
                         setTimeout(resolve, delay * attempt)
                    );
               }
          }
     }

     throw lastError;
}

// Usage example:
// const data = await retryOperation(
//      () => fetch('https://api.example.com/data'),
//      3,
//      1000
// );
```

Tasks

Task 1: Pattern Name and Classification (5 points)

1. Propose a name for this pattern (e.g., "Retry Pattern", "Exponential Backoff Pattern").
2. Classify it into one of the three categories from Chapter 6: Creational, Structural, or Behavioral.

3. Justify your classification with reference to the category definitions in the textbook.

Task 2: Core Structure Documentation (25 points) Document the pattern following Chapter 3, Section "The Structure of a Design Pattern":

1. **Context (Prerequisites) (8 points)**:
   - Describe the situation where this pattern applies
   - List prerequisites (e.g., async operations, error handling needs)
   - Identify environmental constraints
2. **Problem (Forces) (8 points)**:
   - Clearly articulate the problem this pattern solves
   - Describe the forces or constraints that make this problem challenging
   - Explain why a simple solution isn't sufficient
3. **Solution (Implementation) (9 points)**:
   - Describe the structure of the solution
   - Identify the participants (functions, parameters, return values)
   - Explain the collaborations between components
   - Provide implementation guidelines

Task 3: GoF Format Extensions (15 points) Complete the pattern documentation with additional elements from Chapter 3, "Writing a Pattern":

1. **Consequences (5 points)**: Describe the trade-offs, benefits, and liabilities of using this pattern.
2. **Related Patterns (5 points)**: Identify at least two related patterns from the GoF catalog or modern JavaScript patterns. Explain how they relate (complement, replace, or combine).
3. **Known Usage (5 points)**: Provide at least one real-world example where this pattern is used (e.g., in popular libraries, frameworks, or applications). Include specific references if possible.

Task 4: Pattern Illustration (5 points) Create a simple diagram (ASCII art, flowchart, or UML-style) showing:

- The flow of execution
- The relationship between the retry function and the operation
- The retry loop structure

Deliverable

Submit a formal pattern specification document (PDF) that includes:

- All four tasks completed
- Proper academic formatting
- References to specific book sections (e.g., "Ch. 3, 'The Structure of a Design Pattern'")

- Professional technical writing
- Code examples demonstrating the pattern

Lab 2.1 Assessment Rubric (50 points)

Task-Specific Criteria:

- **Task 1 - Pattern Name and Classification (5 pts)**:
  - Appropriate pattern name: 2 pts
  - Correct classification (Creational/Structural/Behavioral): 2 pts
  - Justification with category definitions: 1 pt
- **Task 2 - Core Structure Documentation (25 pts)**:
  - Context (prerequisites, environment): 8 pts
  - Problem (forces, constraints): 8 pts
  - Solution (structure, participants, collaborations): 9 pts
- **Task 3 - GoF Format Extensions (15 pts)**:
  - Consequences (trade-offs, benefits): 5 pts
  - Related Patterns (2+ patterns, relationships): 5 pts
  - Known Usage (real-world examples): 5 pts
- **Task 4 - Pattern Illustration (5 pts)**:
  - Clear diagram (flow, relationships): 3 pts
  - Accurate representation: 2 pts

Grading Notes: Documentation quality assessed on clarity, completeness, and adherence to GoF format from Chapter 3.

2. Lab 2.2: Pattern Refactoring and Documentation (50 Points)

Objective

Refactor a utility function into a well-structured pattern and document it using the GoF format. This lab combines practical refactoring skills with academic pattern documentation.

Background

Chapter 3, Section "Well-Written Patterns" emphasizes that patterns should provide substantial reference material and evidence of necessity. This lab requires you to identify a pattern in existing code, refactor it to make the pattern explicit, and document it formally.

Provided Code

You are given the following code that implements a simple caching mechanism:

Listing 2: Legacy Caching Implementation

```
// cacheUtility.js - Legacy implementation
var cache = {};
var maxSize = 100;

function getCached(key) {
    if (cache[key]) {
        return cache[key];
    }
    return null;
}

function setCached(key, value) {
    if (Object.keys(cache).length >= maxSize) {
        var firstKey = Object.keys(cache)[0];
        delete cache[firstKey];
    }
    cache[key] = value;
}

function clearCache() {
    cache = {};
}

function getCacheSize() {
    return Object.keys(cache).length;
}

// Usage scattered throughout codebase:
// var data = getCached('user-123');
// if (!data) {
//     data = fetchUserData('123');
//     setCached('user-123', data);
// }
```

Tasks

Task 1: Pattern Identification and Refactoring (20 points)

1. Identify the design pattern present in this code (hint: it's a Creational pattern from Chapter 7).
2. Refactor the code to make the pattern explicit and well-structured:
   - Use ES6+ syntax (classes or modules)
   - Encapsulate the cache state
   - Provide a clean public API
   - Add proper error handling
3. Implement the pattern following best practices from the textbook.
4. Include JSDoc comments explaining the pattern usage.

Task 2: Pattern Documentation (20 points) Document your refactored pattern using the GoF format from Chapter 3:

1. **Pattern Name**: Give it a formal name (e.g., "Cache Manager Pattern", "LRU Cache Pattern").
2. **Context**: Describe when and where this pattern should be used.
3. **Problem**: Explain the problem it solves (performance, memory management, etc.).
4. **Solution**: Document your refactored implementation structure.
5. **Consequences**: List benefits and trade-offs.
6. **Related Patterns**: Connect it to patterns from Chapter 7 (e.g., Singleton, Factory).
7. **Code Example**: Provide a complete, runnable example demonstrating usage.

Task 3: Comparison and Analysis (10 points) Write a brief analysis (300-400 words) comparing:

- The original implementation vs. your refactored pattern
- How the pattern structure improves maintainability
- How following the GoF format helps other developers understand and use the pattern
- Reference to Chapter 3 concepts about well-written patterns

Technical Requirements

- Code must use ES6+ syntax (classes, modules, const/let)
- Include proper error handling
- Code must be fully functional and testable
- Include unit tests demonstrating the pattern works correctly (bonus: 5 points)
- Follow JavaScript best practices and naming conventions

Deliverable

Submit:

- Refactored code file(s) with proper structure
- Pattern documentation (PDF) following GoF format
- Comparison analysis document
- Unit tests (if completed for bonus)
- README explaining how to run and test the code

Lab 2.2 Assessment Rubric (50 points)

Task-Specific Criteria:

- **Task 1 - Pattern Identification and Refactoring (20 pts)**:
  - Correct pattern identification (Creational pattern): 5 pts
  - Proper encapsulation (ES6+ classes/modules): 5 pts
  - ES6+ syntax (const/let, classes, modules): 5 pts
  - Code quality (JSDoc comments, error handling): 5 pts
- **Task 2 - Pattern Documentation (20 pts)**:
  - Pattern name and classification: 3 pts
  - Context documentation: 3 pts
  - Problem documentation: 3 pts
  - Solution documentation (structure, implementation): 4 pts
  - Consequences (benefits, trade-offs): 3 pts
  - Related Patterns (Chapter 7 connections): 2 pts
  - Code example (runnable): 2 pts
- **Task 3 - Comparison and Analysis (10 pts)**:
  - Comparison quality (original vs refactored): 5 pts
  - Book references (Chapter 3 concepts): 3 pts
  - Writing quality (300-400 words, clarity): 2 pts
- **Bonus - Unit Tests (5 pts)**:
  - Complete test coverage (all methods tested): 5 pts

Grading Notes: Code must compile and run. Non-functional refactored code receives 0 points for Task 1.

Submission Requirements and Regulations

OquLabs Protocol (30 points)

All labs must be completed using OquLabs with the following requirements:

- **Full Screen Mode (10 points)**: Maintain full screen mode throughout the session. Background applications or partial screen mode will result in a **-20 point penalty**.
- **Active Typing (10 points)**: All code must be typed actively. Copy-paste detection will result in a **-30 point penalty**. Boilerplate templates are allowed, but logic must be typed.
- **Session Duration (10 points)**: Minimum session duration of 30 minutes. Suspiciously fast submissions (< 30 mins) will result in a **-10 point penalty** and may be considered as absence.

Note: OquLabs session logs serve as digital attendance records. Short sessions may be treated as absence.

Git Discipline (10 points)

All code submissions must follow professional Git practices:

- **Folder Structure (5 points)**: Organize code in proper folder structure: `Lab_02/` containing subdirectories (e.g., `Lab_02/task1/`, `Lab_02/task2/`). Flat file dumps or root-level submissions will result in a **-10 point penalty**.
- **Conventional Commits (5 points)**: Use conventional commit messages (e.g., `feat: implement retry pattern`, `fix: correct cache eviction logic`). Bad commit messages will result in a **-5 point penalty**.
- **Incremental History**: Submit with at least 3 meaningful commits showing incremental progress. Single file dumps will result in a **-5 point penalty**.

Code Quality Standards (20 points)

- **Naming Conventions (10 points)**: Use consistent camelCase naming for variables and functions (e.g., `retryOperation`, `cacheManager`). Inconsistent naming will result in a **-5 point penalty**.
- **Comments (10 points)**: Include meaningful comments explaining *why* code exists, not just *what* it does. Zero comments will result in a **-10 point penalty**.

AI Usage Policy

If you use AI tools (ChatGPT, Copilot, etc.) during development:

- **AI Report Required**: You must submit an `AI_REPORT.md` file documenting:
  - Which AI tool was used
  - Specific prompts you used
  - How you modified/verified the AI-generated code
  - What you learned from the process
- **Penalties**: Missing AI report when AI was used: **-100 points** (Academic Dishonesty). Lazy or incomplete reporting: **-20 points**.
- **Transparency**: Using AI is allowed, but attribution is mandatory. This follows academic citation standards.

Submission Instructions

1. **Lab 2.1**: Submit your pattern specification document as a PDF
2. **Lab 2.2**: Submit your refactored code, documentation, and analysis as a Git repository (GitHub/GitLab link) or ZIP file containing:
   - All source files in proper folder structure (`Lab_02/task1/`, `Lab_02/task2/`)
   - Pattern documentation (PDF)
   - Comparison analysis document
   - Unit tests (if completed for bonus)
   - `AI_REPORT.md` (if AI was used)
   - README explaining how to run and test the code
3. Include your name, student ID, and date on all submissions
4. Submit through the course portal by the deadline
5. Ensure OquLabs session logs are accessible for verification

Comprehensive Assessment Summary

Total Lab Score Breakdown (100 points per lab)

The final grade for each lab combines task-specific points with general requirements:

Category	Points	Assessment Method
Lab-Specific Tasks		
Lab 2.1: Documentation Tasks	50	See Lab 2.1 Rubric
Lab 2.2: Refactoring Tasks	50	See Lab 2.2 Rubric
Functionality & Code Quality		
Code runs without errors	20	[YES] Runs / [NO] Doesn't run (-20)
Edge cases handled	20	-5 per failed test case
CamelCase naming conventions	10	[YES] Consistent / [NO] Inconsistent (-5)
Comments (Why vs What)	10	[YES] Present / [NO] Zero comments (-10)
OquLabs Protocol		
Full Screen Mode maintained	10	[YES] Full screen / [NO] Partial (-20)
Active Typing (No Copy-Paste)	10	[YES] Typed / [NO] Paste detected (-30)
Session Duration > 30 mins	10	[YES] > 30 min / [NO] < 30 min (-10)
Git Discipline		
Folder Structure (Lab_02/...)	5	[YES] Proper / [NO] Flat dump (-10)
Conventional Commits (feat: fix:)	5	[YES] Proper / [NO] Bad messages (-5)
Incremental History (3+ commits)	0	[YES] 3+ commits / [NO] Single dump (-5)
Documentation		
AI Report (if AI used)	0	[YES] Included / [NO] Missing (-100)
AI Strategy Explained	0	[YES] Detailed / [NO] Lazy (-20)
TOTAL	**100**	

Grading Process:

1. Grade task-specific criteria (50 points) using detailed rubrics
2. Test functionality: Run code, check edge cases (40 points)
3. Review OquLabs logs: Verify protocol compliance (30 points)
4. Check Git repository: Structure, commits, history (10 points)
5. Assess code quality: Naming, comments (20 points)
6. Verify documentation: AI report if applicable
7. Apply penalties for violations
8. Calculate final score (max 100 points per lab)

Academic Integrity

All work must be your own. You may reference the textbook and official documentation, but code and documentation must be written by you. Proper citations are required for any external references. Plagiarism, unauthorized collaboration, or failure to disclose AI usage will result in a zero grade and academic penalties. OquLabs monitoring ensures fair assessment for all students.

References

- Osmani, Addy. *Learning JavaScript Design Patterns, 2nd Edition*. O'Reilly Media, 2020.
- Chapter 3: "Structuring and Writing Patterns"
- Gamma, Erich, et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995. (GoF Format Reference)