CPSC 501 – Advanced Programming Techniques
Midterm Examination - November 2022

Name:_____ ID:_____

Time limit: 50 minutes. You may use your 8.5 by 11-inch double-sided cheat sheet (printed or written); however, no electronic devices are allowed. There is a total of 2 long-answer questions, each worth 50%. Answer these questions in this booklet in the space provided.

Fowler text Code Smells

Duplicated Code, Long Method, Long Parameter List, Shotgun Surgery, Data Clumps, Primitive Obsession, Switch Statements, Lazy Class, Speculative Generality, Temporary Field, Message Chains, Middle Man, Inappropriate Intimacy, Alternative Classes with Different Interfaces, Incomplete Library Class, Data Class, Refused Bequest, Comments

Fowler text Refactoring

Composing Methods (Extract Method, Inline Method, Inline Temp, *Replace Temp with Query*, Introduce Explaining Variable, Split Temporary Variable, Remove Assignments to Parameters
Replace Method with Method Object, Substitute Algorithm)
Moving Features Between Objects (Move Field, Inline Class, Remove Middle Man, Introduce Local Extension)
Organizing Data (Self Encapsulate Field, Change Value to Reference, Replace Array with Object, Duplicate Observed Data, Change Unidirectional Association to Bidirectional, Replace Magic Number with Symbolic Constant, Encapsulate Collection, Replace Type Code with Class, Replace Type Code with Subclasses, Replace Type Code with State/Strategy,
Replace Subclass with Fields)
Simplifying Conditional Expressions (Decompose Conditional, Consolidate Conditional Expression, Consolidate Duplicate Conditional Fragments, Remove Control Flag, Replace Nested Conditional with Guard Clauses, Replace Conditional with Polymorphism, Introduce Assertion)
Making Method Calls Simpler (*Rename Method*, Remove Parameter, Parameterize Method, Replace Parameter with Explicit Methods, Preserve Whole Object, Replace Parameter with Method, Introduce Parameter Object, Remove Setting Method, Hide Method, Replace Constructor with Factory Method, Encapsulate Downcast, Replace Error Code with Exception,
Replace Exception with Test)
Dealing with Generalization (*Pull Up Field, Pull Up Constructor Body, Push Down Method, Push Down Field,* Extract Subclass, Extract Superclass, Extract Interface, Collapse Hierarchy, Form Template Method, Replace Inheritance with Delegation, Replace Delegation with Inheritance)
Big Refactorings (Tease Apart Inheritance, Convert Procedural Design to Objects, Separate Domain from Presentation, Extract Hierarchy)

Part A: Refactoring

Refactor the code given, applying a series of refactorings. Do **3** refactorings in total, each of them different. **None of the refactorings can be renaming, push-up/pull-down types, or replace temp with query, unless they are used within a bigger refactoring. You can perform a refactoring more than once (if needed within another to accomplish it for example), but you will only get credit for it once.** At least one of the refactorings must be structural in nature, resulting in a major change to the class diagram. Half of your grade will be given for the code changes, and the other half for your explanation. Incomplete sentences or explanations which are too broad, general, or brief will not receive full grades for your explanation. Similarly, code that is indecipherable will not receive full grades.

Show each refactoring as a separate step on the following pages, and for each step also briefly explain:

- **a.** What needed to be improved? That is, what "bad smell" was detected? **Use the official name for this from the Fowler text (names above).**
- **b.** What refactoring was applied? **Use the official name for this from the Fowler text (names above)**
- c. Why is the code better structured after the refactoring?

Each refactoring should build on the previous refactoring. To save space and time, you **are required** to indicate code that is copied from a previous step (including starter code, you should use ellipses …) and show **only** what has changed. (Of course if code is left out that is important or otherwise ambiguous you will lose marks.)

After doing 3 refactorings you will be asked to create unit tests for Part B on the result

Refactoring 1:

a) Bad smell detected: _____

b) Name of refactoring: _____

c) Why is the code better structured after the refactoring?

Show the code that results from your refactoring here:

Refactoring 2:

a) Bad smell detected: _____

b) Name of refactoring: _____

c) Why is the code better structured after the refactoring?

Show the code that results from your refactoring here:

Refactoring 3:

a) Bad smell detected: _____

b) Name of refactoring: _____

c) Why is the code better structured after the refactoring?

Show the code that results from your refactoring here:

Part B: Unit Testing

Three Unit Tests around the concept of XXX (based on your code after Refactoring 3):

For each of your **three** tests specify

1. the function being tested, and
2. the input/output combination,
3. the reason for choice of these values,
4. along with JUnit code that accomplishes the refactoring.

Full marks require tests that consider good coverage. (Do not write the Test Class around the function. Everything should be self-contained in your unit test function itself). Y*ou can assume all the necessary Junit 5 libraries are imported already for you.* Write each unit test as a single function.

Simple summary of Junit 5 assert API:

assertEquals(expected, actual, message)
assertEquals(expected, actual, delta, message)
assertNotEquals(expected, actual, message)
assertNotEquals(expected, actual, delta, message)
assertArrayEquals expected, actual, message)
assertArrayEquals expected, actual, delta, message)
assertNull(actual, message)
assertNotNull(actual, message)
assertFalse(actual, message)
assertTrue(actual, message)
assertThrows(class, () -> {code})
assertSame(expected, actual, message)
assertNotSame(expected, actual, message)
fail(message)