**Howard University**

**School of Engineering &Architecture**

**Department of Electrical Engineering & Computer Science**

**Large Scale Programming**

**Spring 2024**

**Final Exam**

100 pts.

**Name :\_\_\_\_\_Riana Battick\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

I declare that I have not collaborated with anyone on this examination

X \_\_\_\_\_\_Riana Battick\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Type Your Name Here

**Instructions:**

* **Submit completed exam to your github repository. Create packages:**

**org.howard.edu.lsp.oopfinal.document**

**org.howard.edu.lsp.oopfinal.question1**

**org.howard.edu.lsp.oopfinal.question2**

**Verify that the commit(s) completed successfully in your Git repository**

* **OPEN BOOK, OPEN NOTES. THERE IS NO COLLABORATION ON THIS EXAM**

**PLEASE DO NOT CONVERT TO PDF WHEN SUBMITTING.**

**Section 1: True/False. (20 pts., 1 pt. each)**

1. T When designing object-oriented systems, nouns in a problem statement may represent classes/variables needed in the design.
2. FWhen designing a class, each class should be designed to have multiple goals so that your overall design can have fewer classes.
3. T Design patterns are a mechanism that enable developer to reuse code in their implementations.
4. T Because of potential problems, developers must be aware of the effects of modifications in a superclass and in each of the subclasses that will inherit the modifications.
5. T Information hiding makes program maintenance easier by hiding data and procedure from unaffected parts of the program.
6. T Software designs are refactored to allow the creation of software that is easier to integrate.
7. FInheritance provides a mechanism by which changes to lower-level classes can be propagated to all super classes quickly.
8. F When iterating a Java HashSet, you are guaranteed to retrieve objects stored in the same order they were inserted.
9. T Behavioral design patterns are all about Class's objects communication
10. FWhen designing a class, each class should be designed to have multiple goals so that your overall design can have fewer classes.
11. F Creational design patterns are all about Class and Object composition.
12. F Structural design patterns are all about class instantiation
13. FDesign patterns are best thought of as coding patterns.
14. T Factory pattern can be combined with multiple design patterns.
15. F A try block can occur without an accompanying catch clause or finally clause.
16. T Low levels of inheritance coupling in a system is good and desirable.
17. T In Java, the signature of a method is completely specified by the name of the method and the parameters that must be passed to the method.
18. T In Java, when an instance of a class, or object, is specified as a parameter to a method, a reference to the said object is passed to the method.
19. T The Builder pattern separates the construction of a complex object from its representation, allowing the same construction process to create different representations.
20. F Polymorphism is the mechanism that combines processes and data into a single object.
21. FFrameworks and design patterns are the same thing as far as designers are concerned.
22. F The relationship between two objects related by composition cannot be changed at runtime.
23. T The Factory Method pattern is a creational pattern that defines an interface for creating objects but let’s subclasses decide which class to instantiate.
24. **F JUnit is a test framework that is part of the Java Programming Language.**
25. F The difference between S3 and EBS storage is that EBS is object-based and S3 is block-based.

**Section 2: Multiple Choice, answer each question. (30 pts., 1 pt. each)**

1. How does the “Liskov Substitution Principle” contribute to good design?
   1. It promotes the idea that a class should have only one responsibility.
   2. It ensures that classes that use each other are loosely coupled.
   3. It dictates that objects in a system should be replaceable with instances of their subtypes without affecting the correctness of the program.
   4. It suggests that a class should be open for extension but closed for modification.

Ans. C

1. What is the role of mocking frameworks like Mockito in unit testing?
   1. They provide assertions for test cases.
   2. They are used to create mock objects for unit tests.
   3. They execute test cases in parallel.
   4. They automate the testing process.

Ans. B

1. What is the primary purpose of unit testing?
   1. To verify the correctness of the entire system.
   2. To test the integration between different components.
   3. To validate that individual units of code work as expected.
   4. To assess the performance of the application.

Ans. C

1. What is regression testing?
   1. Testing the system in various environments.
   2. Repeating previous tests to ensure existing functionality is not affected by changes.
   3. Testing the performance of the system under load.
   4. Verifying the correctness of a single unit of code.

Ans. B

1. What is the main benefit of using CRC cards in the object-oriented design process?
   1. It helps with the implementation of code
   2. It facilitates communication and collaboration among team members.
   3. It ensures the security of the software.
   4. It automates the testing process.

Ans. B

1. Cohesion is a qualitative indication of the degree to which a module
   1. can be written more compactly.
   2. focuses on just one thing.
   3. is able to complete its function in a timely manner.
   4. is connected to other modules and the outside world.

Ans. B

1. Coupling is a qualitative indication of the degree to which a module
2. can be written more compactly.
3. focuses on just one thing.
4. is able to complete its function in a timely manner.
5. is connected to other modules and the outside world.

Ans. D

1. The root interface of the Java Collection framework hierarchy is
   * 1. Collection
     2. Root
     3. Collections
     4. List/Set

Ans. A

1. Suppose the class Undergraduate extends the class Student that extends the class Person.

Given the following variable declarations:

Person p = new Person();

Student s = new Student();

Undergraduate ug = new Undergraduate();

Which of the following assignments are legal?

1. p = ug;
2. p = new Undergraduate();
3. ug = new Student();
4. ug = p;
5. s = new Person();
   1. I and IV
   2. I and II
   3. III and IV
   4. II, III and V

Ans. B

1. Which of the following option leads to the portability and security of Java?
   1. Bytecode is executed by JVM
   2. Use of exception handling
   3. Dynamic binding between objects
   4. Proper encapsulation of classes and objects.

Ans. A

1. What interface in the Java Collections framework extends Map and represents a collection of key/value pairs where keys are ordered?
   1. HashMap
   2. LinkedHashMap
   3. TreeMap
   4. HashTable

Ans. B

1. What interface in the Java Collections framework represents a last-in, first-out (Last In First Out) collection of objects?
   1. Queue
   2. List
   3. HashMap
   4. Stack

Ans. D

1. What does the “Single Responsibility Principle” state?
   1. A class should only have one instance.
   2. A class should be open for extension but closed for modification.
   3. A class should have only one responsibility or reason for change.
   4. Objects in a system should be replaceable with instances of their subtypes without affecting the correctness of the program.

Ans. C

1. Which of the following are concerned with communication between objects?
   1. J2EE Design Patterns
   2. Behavioral Design Patterns
   3. Structural Design Patterns
   4. Creational Design Patterns

Ans. B

1. Why is “Scalability” an important consideration in software design?
   1. To ensure that classes that use each other are loosely coupled.
   2. To design software in a way that the behavior is as expected and doesn’t surprise users or developers.
   3. To measure how closely the members of a class are related to one another.
   4. To accommodate future growth by allowing the software to handle increased workload and demand.

Ans. D

1. In Arthur Riel’s design heuristics, what is the significance of the heuristic “Prefer Polymorphism to if-then-else or switch statements?
   1. Encourage the extensive use of conditional statements for clarity.
   2. Suggest using polymorphism to handle different cases rather than extensive conditional statements.
   3. Prioritize the use of switch statements for better performance.
   4. Discourage the use of polymorphism due to its complexity.

Ans. B

1. Which of the following is true about design patterns? (Choose the best answer).
2. Design patterns represent the best practices used by experienced object-oriented software developers.
3. Design patterns are solutions to general problems that software developers faced during software development.
4. Design patterns are obtained by trial and error by numerous software developers over quite a substantial period.
5. All of the above.

Ans. D

1. What is the primary goal of the “Composition Over Inheritance” principle?
   1. To promote the idea that a class should have only one responsibility.
   2. To suggest that a class should be open for extension but closed for modification.
   3. To encourage favoring object composition over class inheritance for code reuse and flexibility.
   4. To ensure that objects in a system should be replaceable with instances of their subtypes without affecting the correctness of the program.

Ans. C

1. You want all the clients using class A to use the same instance of class A when the class is instantiated, what should you do to achieve this goal?
   1. Mark class A final
   2. Mark class A abstract
   3. Apply the Singleton pattern to class A
   4. Apply the Proxy pattern to class A

Ans. C

1. You have a class that accepts and returns values in British Imperial units (feet, miles, etc.), but you need to use metric units. The design pattern that would best solve your problem is:
2. Adapter
3. Decorator
4. Delegation
5. Proxy

Ans. A

1. In the Proxy design pattern, what is the purpose of the Proxy class?
   1. To provide an interface for creating families of related or dependent objects without specifying their concrete classes.
   2. To ensure a class has only one instance and provides a global point of access to it.
   3. To control access to the real object by serving as a surrogate or placeholder.
   4. To define a family of algorithms, encapsulate each one, and make them interchangeable.

Ans. C

1. Which of the following describes the Facade pattern correctly?
   1. This pattern allows a user to add new functionality to an existing object without altering its structure.
   2. This pattern is used when we need to treat a group of objects in a similar way as a single object.
   3. This pattern hides the complexities of the system and providers an interface to the client using which the client can access the system.
   4. This pattern is primarily used to reduce the number of objects created and to decrease memory footprint and increase performance.

Ans. C

1. Which of the following is correct about Creational design patterns?
   1. [These design patterns are specifically concerned with communication between objects.](javascript:void(0);)
   2. These design patterns provide a way to create objects while hiding the creation logic, rather than instantiating objects directly using new operator.
   3. [These design patterns concern class and object composition. Concept of inheritance is used to compose interfaces and define ways to compose objects to obtain new functionalities.](javascript:void(0);)
   4. [None of the above.](javascript:void(0);)

Ans. B

1. What is the role of the Template Method in the Template Method design pattern?
   1. To ensure a class has only one instance and provides a global point of access to it.
   2. To define a family of algorithms, encapsulate each one, and make them interchangeable.
   3. To provide an interface for creating families of related or dependent objects without specifying their concreate classes.
   4. To define the skeleton of an algorithm in the superclass but let subclasses override specific steps of the algorithm without changing its structure.

Ans. D

1. Which of the following describes the Factory pattern correctly?
   1. This pattern creates an object without exposing the creation logic to the client and refers to newly created objects using a common interface.
   2. In this pattern, an interface is responsible for creating a factory of related objects without explicitly specifying their classes.
   3. This pattern involves a single class that is responsible to create an object while making sure that only a single object is created.
   4. This pattern is used when we want to pass data with multiple attributes in one shot from client to server.

Ans. A

1. This design pattern should be used to access the contents of a collection without exposing its internal representation, to support multiple traversals of a collection, and to provide a uniform interface for traversing different collections.
   1. Template method
   2. Strategy
   3. Iterator
   4. Factory method

Ans. C

1. Which design pattern should you use when you want to provide a simple interface to a complex subsystem?
2. Adapter
3. Facade
4. Abstract Factory
5. Singleton

Ans. B

1. What is the intent of the Adapter design pattern?
   1. To provide an interface for creating families of related or dependent objects without specifying their concrete classes.
   2. To convert the interface of a class into another interface clients expect.
   3. To ensure a class has only one instance.
   4. To define a family of algorithms, encapsulate each one, and make them interchangeable.

Ans. B

1. In the Command design pattern, what is the purpose of the Command interface?
   1. To provide an interface for creating families of related or dependent objects without specifying their concrete classes.
   2. To encapsulate a request as an object, thereby allowing for parameterization of clients with different requests.
   3. To define a family of algorithms, encapsulate each one, and make them interchangeable.
   4. To ensure a class has only one instance.

Ans. B

1. What is the main advantage of the Strategy design pattern?
   1. It allows a class to have only one instance.
   2. It defines a family of algorithms, encapsulates each one, and makes them interchangeable.
   3. It allows the definition of a family of algorithms, encapsulates each one, and makes them interchangeable.
   4. It enables the selection of an algorithm’s implementation at runtime.

Ans. D

**Section 3: Object-Oriented Design Heuristics (20 pts. total, 5 pts. each)**

One of your project partners has been working on a design for the HU University registration system. How would you answer the following questions? Support your decisions by citing relevant design concepts and heuristics presented in the course. Express each answer as **succinctly** as possible.

1. Right now, our RegistrationSystem class holds all courses and students, contains all logic for checking prerequisites, and has the methods for adding and dropping a course. It also has the database connection logic right now. Is this, okay?

**Current Design Issue**: The RegistrationSystem class in its current state is handling too many responsibilities: managing courses and students, processing prerequisite checks, course management functions, and database connectivity. This violates the Single Responsibility Principle(SRP), which suggests that a class should have one reason to change. Such a design also negatively impacts maintainability and scalability.

**Better Approach**: Refactor the RegistrationSystem to focus on core registration functionalities. Separate concerns by creating dedicated classes for handling database operations (e.g., Database Manager) and move prerequisite checking into either a utility class or integrate it within course-related classes.

1. Right now, the Student class has a display method that prints out the HTML for the student to be displayed on the web site. We also have a similar display method in the Course class. Is this okay, and should we add a display method to other classes?

**Current Design Issue:** Incorporating presentation logic (HTML rendering) directly within the Student and Course classes mixes data management with display logic, contravening the Separation of Concerns principle. This design makes the system less flexible and harder to maintain, especially if changes are required in the display logic or format.

**Better Approach**: Utilize a design pattern like MVC (Model-View-Controller) where Models (Student, Course) are separated from Views (HTML presentations). Introduce a separate class or classes responsible for rendering models in HTML or other formats. This makes the system more adaptable and easier to manage different aspects of the application.

1. Should a Student object store a list of the student's courses, or should a Course object store a list of the course's students? Or both, or neither?

**Current Design Issue**: Storing a list of courses within a Student object and vice versa can lead to redundancy and potential issues with data synchronization and integrity. It also complicates the serialization and deserialization processes.

**Better Approach:** Apply the Law of Demeter for designing loose coupling and high cohesion between classes. Consider a third class to manage the relationship, such as Registration that keeps track of which students are registered in which courses. This design adheres better to object-oriented principles by maintaining clear roles and responsibilities.

1. Since there are different levels of students, we made an inheritance hierarchy. The superclass is called Student, and the subclasses are called Freshman, Sophomore, Junior, and Senior. We were going to add another subclass called GradStudent; is this the right design?

**Current Design Issue:** Using an inheritance hierarchy to represent different levels of students

(Freshman, Sophomore, Junior, Senior, GradStudent) might not be ideal as it can lead to a

rigid system where changes in the student's level would require instantiation of a new type of

object.

**Better Approach:** Instead of inheritance, consider using a composition approach. Have a single Student class that includes a property or component class representing the student's level or status. This design is more flexible and scalable, allowing easy modification of a student’s status without needing multiple subclasses. For the distinct characteristics and behaviors of graduate students, strategy or state design patterns might be utilized to encapsulate varying behaviors without the complexity of subclassing.

**Section 4: Programming Problems (30 pts. total)**

**Question 1.**

**Please upload to org.howard.edu.lsp.oopfinal.question1**

**(15 pts.)**

**Convert the following class into a Singleton pattern. The class represents a simple logging utility. Ensure that every time the Logger is instantiated, the same instance is returned**

public class Logger {

private String log;

public Logger() {

this.log = "";

}

public void logMessage(String message) {

this.log += message + "\n";

}

public String displayLog() {

return("Log:\n" + log);

}

}

**Create TestLogger and implement JUnit test case called *testDisplayLog*. In addition, create JUnit test case *testLoggerInstances* where the intent is to determine if multiple instantiations of Logger(2 should suffice) actually return the same instance. Look through the variety of assertions that exist and choose one that is appropriate. Both implementation and test code can go in the same package.**

**(5 pts.)**

**Your solution should have:**

* + - 1. **Logger.java (implementation)**
      2. **TestLogger.java (JUnit test cases). You only need to write a JUnit test case for displayLog().**

**Question 2.**

**Please upload to org.howard.edu.lsp.oopfinal.question2**

**(15 pts.)**

Implement the **Strategy Pattern** to model a payment system.

Define an interface called **PaymentStrategy** with a method pay that takes the payment amount as a parameter, i.e.,

interface PaymentStrategy {

void pay (double pay);

}

Create concrete classes **CreditCardPayment**, **PayPalPayment** and **BitcoinPayment** that implements the above. Each concrete class should provide its own implementation of the pay method.

The **CreditCardPayment** class requires a credit card number for instantiation (passed in constructor, String), the **PayPalPayment** class requires an email address (passed in constructor, String), and the **BitconPayment** class requires a Bitcoin address (passed in constructor, String). You should assign these to appropriately named variables in each constructor.

Create a **ShoppingCart** class that has **PaymentStrategy** instance/variable and a method called ***checkout*** that takes the payment amount and calls the pay method of the provided payment strategy (review **Strategy** pattern from lecture notes).

Finally, create **PaymentStrategyDriver.java** that contains a main program that produces the following output. Essentially, using the Strategy pattern, your main program should show how easily you can change your payment algorithm. You must generate the same output as below.

// Output

Paid 100.0 using credit card 1234-5678-9012-3456

Paid 50.0 using PayPal account user@example.com

Paid 75.0 using Bitcoin address 1AaBbCcDdEeFfGgHh