**Howard University**

**School of Engineering &Architecture**

**Department of Electrical Engineering & Computer Science**

**Large Scale Programming**

**Fall 2023**

**Final Exam**

100 pts.

**Name :Ricky Edwards**

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

X Ricky Edwards

Ricky Edwards

**When completed, the Word document exam and respective programming solutions should be uploaded to**

**your Github repository. The packages for this exam are:**

**org.howard.edu.lsp.exam.document**

**org.howard.edu.lsp.exam.problem51**

**org.howard.edu.lsp.exam.problem52**

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

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

1. T FWhen designing a class, each class should be designed to have multiple goals so that your overall design can have fewer classes.

**F**

1. T F Design patterns are a mechanism that enable developer to reuse code in their implementations.

**T**

1. T F Information hiding makes program maintenance easier by hiding data and procedure from unaffected parts of the program.

**T**

1. T F Software designs are refactored to allow the creation of software that is easier to integrate.

**T**

1. T FInheritance provides a mechanism by which changes to lower-level classes can be propagated to all super classes quickly.

**F**

1. T FDesign patterns are best thought of as coding patterns.

**F**

1. T FFrameworks and design patterns are the same thing as far as designers are concerned.

**F**

1. T FFactory pattern can be combined with the Singleton pattern.

**T**

1. T F Creational design patterns are all about Class and Object composition.

**F**

1. T F Structural design patterns are all about class instantiation

**F**

1. T F Behavioral design patterns are all about Class's objects communication

**T**

1. T F Polymorphism is the mechanism that combines processes and data into a single object.

**F**

1. T F 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.

**T**

1. T F High levels of inheritance coupling in a system is good and desirable.

**F**

1. T F 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.

**T**

1. T F 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.

**T**

1. T F The return value from a Java method must always match the declared return type.

**T**

1. T F The relationship between two objects related by composition cannot be changed at runtime.

**F**

1. T F A try block can occur without an accompanying catch clause or finally clause.

**F**

1. T F When iterating a Java HashSet, you are guaranteed to retrieve objects stored in the same order they were inserted.

**F**

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

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.

**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. III and IV
   2. I and IV
   3. I and II
   4. II, III and V

**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.

**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.

**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.

**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.

**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.

**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.

**D**

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

**A - collection**

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

**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

**D - stack**

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.

**C**

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.

**C**

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.

**C**

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.

**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.

**D**

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.

**D**

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

**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

**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.

**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.

**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

**C**

1. Which of the following is correct about Creational design patterns.
   1. These design patterns are specifically concerned with communication between objects.
   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.
   4. None of the above.

**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.

**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.

**A**

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.

**D**

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

**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

**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.

**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.

**B**

**Section 3: Object-Oriented Design Heuristics (20 pts. total, 4 pts. each)**

* + 1. 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. 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?

**Both Student and Course objects can store a list of each other. This is known as a bidirectional association. It allows for efficient querying of data from both perspectives. However, it also requires careful synchronization when adding or removing elements.**

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?

**No, the RegistrationSystem class has too many responsibilities, violating the Single Responsibility Principle. It’s better to separate concerns. For instance, database connection logic could be moved to a separate DatabaseManager class. Prerequisite checking could be part of the Course class.**

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?

**Using inheritance for different levels of students might not be the best design. The behavior of a student doesn’t necessarily change with their level, so using a “level” attribute in the Student class could be more appropriate. If GradStudent has different behaviors, then it could be a subclass.**

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?

**Having display methods in the Student and Course classes violates the Separation of Concerns principle. It’s better to have a separate class (e.g., StudentView, CourseView) responsible for presentation logic.**

1. Right now, the Student object talks to the Registrar object when it wants to send messages to a Course object. The Registrar object intercepts the messages and passes them along to the relevant Course. The Course also sends messages to the Registrar object when it wants to access a particular Student. What changes, if any, would you suggest to this design?

**The Registrar object acting as a mediator between Student and Course objects can be a good design if you want to encapsulate how objects interact and promote loose coupling. However, if a Student needs to frequently interact with a Course, direct communication might be more straightforward.**

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

The following programming problems should be uploaded to your github repository. You should have packages:

1. howard.edu.lsp.final.problem51
2. howard.edu.lsp.final.problem52
3. howard.edu.lsp.final.document (put multiple choice questions here)
   * 1. **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.

**(10 pts.)**

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)

The package structure for this problem is: **package org.howard.edu.lsp.exam.problem51**

* + 1. Implement the **Strategy Pattern** to model a payment system.

**(15 pts.)**

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.

or

// 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

The package structure for this problem is: **package org.howard.edu.lsp.exam.problem52**