Howard University
College of Engineering and Architecture
Department of Electrical Engineering & Computer Science

Large Scale / Object-Oriented Programming
Spring 2024

Final Exam (100 pts.)
May 2, 2024

Name :Aayushka Budhathoki

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

X **Aayushka Budhathoki**

Type Your Name Here

Instructions:

- **Submit completed exam to your github repository. Create packages:**
  org.howard.edu.lsp.oopfinal.exam
  org.howard.edu.lsp.oopfinal.question1
  org.howard.edu.lsp.oopfinal.question2
  org.howard.edu.lsp.oopfinal.question3

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

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

Section 1: True/False. (15 pts., 1 pt. each)

1. T F When designing a class, each class should be designed to have multiple goals so that your overall design can have fewer classes
2. T F Inheritance provides a mechanism by which changes to lower-level classes can be propagated to all super classes quickly
3. T F Design patterns are reusable libraries that can immediately be used in your code
4. T F Frameworks and design patterns are the same thing as far as designers are concerned
5. 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.
6. T F Factory pattern can be combined with other patterns
7. T F Creational design patterns are all about Class and Object composition.
8. T F Structural design patterns are all about class instantiation
9. T F Design patterns are a mechanism that enable developer to reuse code in their implementations.
10. T F Behavioral design patterns are all about Class's objects communication
11. T F Information hiding makes program maintenance software development more difficult because other developer are not privy to an object's implementation details.
12. 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.
13. T F The relationship between two objects related by composition cannot be changed at runtime.
14. T F When iterating a Java HashSet, you are guaranteed to retrieve objects stored in the same order they were inserted
15. T F Software designs are refactored to allow the creation of software that is easier to maintain and reuse.

Section 2: Multiple Choice, answer each question. (20 pts., 1 pt. each)

1. Which of the following option leads to the portability and security of Java?

a) Bytecode is executed by JVM
b) Use of exception handling
c) Dynamic binding between objects
d) Proper encapsulation of classes and objects.

2. What is the role of mocking frameworks like Mockito in unit testing?

a) They provide assertions for test cases.
b) They are used to create mock objects for unit tests.
c) They execute test cases in parallel.
d) They automate the testing process.

3. What is the primary purpose of unit testing?

a) To verify the correctness of the entire system.
b) To test the integration between different components.
c) To validate that individual units of code work as expected.
d) To assess the performance of the application.

4. What is regression testing?

a) Testing the system in various environments.
b) Repeating previous tests to ensure existing functionality is not affected by changes.
c) Testing the performance of the system under load.
d) Verifying the correctness of a single unit of code.

5. The root interface of the Java Collection framework hierarchy is

a) Collection
b) Root
c) Collections
d) List/Set

6. What interface in the Java Collections framework extends Map and represents a collection of key/value pairs where keys are ordered?

a) HashMap
b) LinkedHashMap
c) TreeMap
d) HashTable

7. What interface in the Java Collections framework represents a last-in, first-out (Last In First Out) collection of objects?

a) Queue
b) List
c) HashMap
d) Stack

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

9. 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?
a) Mark class A final
b) Mark class A abstract
c) Apply the Singleton pattern to class A
d) Apply the Proxy pattern to class A

10. 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:
a) Adapter
b) Decorator
c) Delegation
d) Proxy

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

12. Which of the following are concerned with communication between objects?
a) J2EE Design Patterns
b) Behavioral Design Patterns
c) Structural Design Patterns
d) Creational Design Patterns

13. Which of the following is correct about Creational design patterns?
a) These design patterns are specifically concerned with communication between objects.
b) These design patterns provide a way to create objects while hiding the creation logic, rather than instantiating objects directly using new operator.
c) 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.
d) None of the above.

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

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

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

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

18. 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.

a) Template method
b) Strategy
c) Iterator
d) Factory method

19. Which design pattern should you use when you want to provide a simple interface to a complex subsystem?

a) Adapter
b) Facade
c) Abstract Factory
d) Singleton

20. What is the intent of the Adapter design pattern?

a) To provide an interface for creating families of related or dependent objects without specifying their concrete classes.
b) To convert the interface of a class into another interface clients expect.
c) To ensure a class has only one instance.
d) To define a family of algorithms, encapsulate each one, and make them interchangeable.

Section 3: Programming Questions (65 pts. total)

Question 1.
Please upload to org.howard.edu.lsp.oopfinal.question1

The **SongsDatabase** class keeps tracks of song titles by classifying them according to Genre (e.g., Jazz, Rap, Pop, Rock, etc.). The class uses a HashMap to map a genre with a set of songs that belong to such a genre. The set of songs will be represented using a HashSet.

In addition to the implementation, you are required to **write JUnit test cases** for each method.

Note: although addSong returns a void, you can write a test case that checks if the song has been added to the HashMap successfully.
(25 pts.)

```
public class SongsDatabase {
   private Map<String, Set<String>> map =
      new HashMap<String, Set<String>>();

   /* Add a song title to a genre */
   public void addSong(String genre, String songTitle) {
        // Code it!!
    }

    /* Return genre, i.e., jazz, given a song title */
   public String getGenreOfSong(String songTitle) {
       // Code it!!
   }


   /* Return the Set that contains all songs for a genre */
   public Set<String> getSongs(String genre) {
       // Code it!!
   }
}
```

```
Below is sample that uses the above implementation:
              …
 SongsDatabase db = new SongsDatabase();
 db.add("Rap", "Savage");
 db.add("Country", "Sweet Alabama");
 db.add("Jazz", "Always There");

 Set<String> songs = db.getSongs("Rap");
 System.out.println( songs ); prints [Savage, Gin and Juice]
 System.out.println( db.getGenreOfSong("Savage") );// prints "Rap"
 System.out.println( db.getGenreOfSong("Always There") );// prints "Jazz"
```

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
```

JUnit test cases not required! Just produce the above output

Question 3.
Please upload to org.howard.edu.lsp.oopfinal.question3
(25 pts.)

Consider the following code snippet:

```
// Shape.java
public interface Shape {
   void draw();
}

// Circle.java
public class Circle implements Shape {
    @Override
    public void draw() {
        System.out.println("I'm a Circle!!!);
    }
}

// Rectangle.java
public class Rectangle implements Shape {
    @Override
    public void draw() {
        System.out.println("I'm a Rectangle!!!);
    }
}

// Client.java
public class Client {
    public static void main(String[] args) {
       Circle c = new Circle();
        c.draw();
        Rectangle r = new Rectangle();
        r.draw();
    }
}
```

Refactor the above code so that class Main delegates creation of Circles and Rectangles utilizing the Factory pattern. Your output should be the same but object creation is delegated to a Factory.

Validate your implementation by **writing JUnit test** cases that instantiate Circle and Rectangle.