**Howard University**

**College of Engineering and Architecture**

**Department of Electrical Engineering & Computer Science**

**Large Scale / Object-Oriented Programming**

**First Exam**

March 16, 2023

**Name : Sanjeev Roka**

**ID :@03028036**

1. **Please return this exam in Word format, do not convert to pdf**
2. **THERE IS NO COLLABORATION ON THIS EXAM. By submitting this exam, you are making a declaration that you have not discussed or collaborated with any other person.**

**Section 1: True/False, type or color your answer. (25 pts., 1 pt. each)**

1. **T F** Google naming conventions recommend that a Java class should begin with a lowercase letter.
2. **T F** Polymorphism is used extensively in implementing inheritance.
3. **T F** Object-oriented design strives to have low cohesion and high coupling in class design.
4. **T F** The ability to hide the implementation details of an object is called encapsulation.
5. **T F** The “IsA’ relationship describes the relationship when one class is contained within another.
6. **T F** Java code is a compiled language that creates binary code for a specific architecture and must be recompiled when moved to another machine.
7. **T F** The Java programming language supports multiple inheritance.
8. **T F** All classes in the Java programming language implicitly inherit from the Top base class.
9. **T F** The static modifier on a member variable associates it with the entire class itself rather than separately for each object.
10. **T F** To maximize encapsulation, instance variables should always be declared to be public.
11. **T F** Methods are said to be *overridden* if they are in the same scope and have the same names but different signatures.
12. **T F** A method that *overrides* another must have the same name but a different signature.
13. **T F** A subclass can add behavior that is not present in the superclass.
14. **T F** If class A extends class B, class A is a subclass of B and B is a superclass of A
15. **T F** Testing ensures that our programs will never have any bugs.
16. **T F** The @Test annotation is used in JUnit to identify methods in a class that are executed as test methods.
17. **T F** A constructor can be invoked multiple times once an object is instantiated.
18. **T F** A constructor can be overloaded within a class definition.
19. **T F** An array in Java will automatically change its size as needed.
20. **T F** An ArrayList in Java will automatically change its size as needed.
21. **T F** An inheritance relationship between two classes can’t be changed at runtime.
22. **T F** An interface in Java contains methods with full implementations that are inherited by classes that implement them.
23. **T F** A Java method returns void if it does not return a value.
24. **T F** A class in Java may implement only one interface.
25. **T F** A class in Java may extend only one class.

**Section 2: Multiple Choice, type answer below each question. (25 pts., 1 pt. each)**

1. Which of the following is one of the benefits of object-oriented programming?   
   all of the above
   1. enables code reuse
   2. more understandable code
   3. more maintainable code
   4. all of the above
2. In object-oriented terminology, the fact that an object can be considered a conceptually independent entity is known as:  
   encapsulation
   1. polymorphism
   2. inheritance
   3. overloading
   4. message passing
   5. encapsulation
3. Which of the following is also known as run-time binding or late binding?  
   dynamic binding
   1. dynamic typing
   2. dynamic loading
   3. dynamic binding
   4. data hiding
4. Which two features of object-oriented programming are the same?  
   encapsulation and abstraction
   1. abstraction and polymorphism features are the same
   2. inheritance and encapsulation features are the same
   3. encapsulation and polymorphism features are the same
   4. encapsulation and abstraction
5. An object-oriented design tends to focus on blank to identify objects:   
   nouns
   1. verbs
   2. nouns
   3. encapsulation
   4. inheritance
6. Standard code libraries in Java are organized into  
   package
   1. interfaces
   2. classes
   3. objects
   4. package
7. What is the extension of compiled java classes?  
   .class
   1. .txt
   2. .js
   3. .class
   4. .java
8. Which of the following statements is incorrect in Java? (1pt.)  
   private members of a class can be inherited by a sub class and become protected members in sub class.
   1. public members of a class can be accessed by any other class.
   2. private members of a class can be inherited by a sub class, and become protected members in sub class.
   3. protected members of a class can be accessed by any class in the same package.
   4. private members of a class can only be accessed by members of that same class it was declared
9. Which of the following is true about an abstract method inherited into a class called C?  
   It must be overridden in C before C can be instantiated
   1. It must be defined in C before C can be instantiated
   2. It must be overridden in C before C can be instantiated
   3. It always forces C to become abstract, even if defined in C.
   4. It overrides any method in C with the same name
10. What can directly access and change the value of the variable roomNr?

package com.mycompany;

public class Hotel {

protected int roomNr = 100;

}

any class that extends Hotel

* 1. only the Hotel class
  2. any class
  3. any class in com.mycompany package
  4. any class that extends Hotel

1. Given the code. What is true?

public class Room {

public int roomNr;

private Date beginDtm;

private Date endDttm;

public void book(int roomNr, Date beginDttm, Date endDttm) {

this.roomNr = roomNr;  
 this.beginDtm = beginDttm;  
 this.endDttm = endDttm;

}

}

the variable roomNr breaks encapsulation

1. the code demonstrates polymorphism
2. the class is fully encapsulated
3. the variable roomNr breaks encapsulation
4. variables beginDttm and endDttm break polymorphism
5. the method book breaks encapsulation
6. Which of the following allow us to define an IS-A relationship in Java? Circle **all** that apply.  
   interfaces, classes, dynamic binding
7. interfaces
8. classes
9. local variables
10. dynamic binding
11. none of the above
12. Why would a class be declared as abstract?  
    Because it doesn’t make logical sense to instantiate it
    1. Because it doesn’t make logical sense to instantiate it
    2. So that it can be used as an interface
    3. So that it cannot be inherited from
    4. Because it has no abstract methods
13. Which of these can be overloaded?  
    both a & b
    1. methods
    2. constructors
    3. both a & b
    4. classes
    5. interfaces
14. Which of the following statements is correct for a method that is overriding the following method:

public void add(int a) { … }

the overriding method must return void

1. the overriding method must return void
2. the overriding method can return whatever it likes
3. the overriding method must return int
4. the overriding method must return different parameters
5. the overriding method must exist in the same class definition as the above
6. What is the return type of a Constructor?  
   none of the mentioned
   1. float
   2. int
   3. void
   4. none of the mentioned
7. Amazon S3 is which type of storage service?  
   object
8. object
9. block
10. simple
11. secure
12. Which of the following is a message queue or transaction system for distributed internet-based applications?  
    Amazon Simple Queue Service
    1. Amazon Simple Notification Service
    2. Amazon Elastic Compute Cloud
    3. Amazon Simple Queue Service
    4. Amazon Simple Storage System
13. **Which of the following features of Amazon RDS allows for better availability of databases? Choose the answer from the options given below.  
    Multi-AZ**
    1. VPC Peering
    2. Multi-AZ
    3. Read Replicas
    4. Data encryption
14. Which of the following are the recommended resources to be deployed in the Amazon VPC private subnet, i.e., makes the resource(s) more secure?  
    Internet Gateways
    1. NAT Gateways
    2. Bastion Hosts
    3. Database Servers
    4. Internet Gateways
15. What is the AWS feature that enables fast, easy, and secure transfers of files over long distances between your client and your Amazon S3 bucket?  
    Amazon S3 Transfer Acceleration
    1. File Transfer
    2. HTTP Transfer
    3. Amazon S3 Transfer Acceleration
    4. S3 Acceleration
16. Which statement is true about cohesion
    1. Cohesion is the OO principle most closely associated with hiding implementation details.
    2. Cohesion is the OO principle most closely associated with making sure classes know about other classes only through their APIs.
    3. Cohesion is the OO principle most closely associated with making sure that a class is designed with a single, well focused purpose.
    4. Cohesion is the OO principle most closely associated with allowing a single object to be seen as having many types.
17. Which of the following is a method having same name as that of its class?  
    constructor
    1. finalize
    2. interface
    3. class
    4. constructor
18. Given:

public interface Jumper { public void jump(); }

public class Animal { }

public class Dog extends Animal { protected Tail tail; }

public class Beagle extends Dog implements Jumper {

public void jump() {};

}

public class Cat implements Jumper { public void jump(); }

Which of the following are true?  
Cat is-a Jumper, Dog-is-a Animal, Beagle has a Tail

a) Cat is-a Animal

b) Cat is-a Jumper

c) Dog is-a Animal

d) Dog is-a Jumper

e) Cat has-a Animal

f) Beagle has a Tail

g) Beagle has a Jumper

1. Which of the following describes the Model View Controller (MVC) pattern?  
   The pattern is used to decouple an application's presentation tier and business tier.
   1. The pattern is used to encapsulate the process of creating objects.
   2. The pattern is used to identify common communication patterns among objects.
   3. The pattern is used to decouple an applications presentation tier and business tier.
   4. The pattern is used [to ease the design by identifying a simple way to realize relationships among entities.](https://www.bing.com/ck/a?!&&p=f0515b2d16a8ec16JmltdHM9MTY3MDM3MTIwMCZpZ3VpZD0zMTQ1YTBiZi05NGFkLTY1ZWMtMTk5YS1iMmNjOTVlMjY0YTgmaW5zaWQ9NTY4NA&ptn=3&hsh=3&fclid=3145a0bf-94ad-65ec-199a-b2cc95e264a8&psq=what+are+structural+design+patterns&u=a1aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnL3dpa2kvU3RydWN0dXJhbF9wYXR0ZXJu&ntb=1" \o "en.wikipedia.org" \t "_blank)

**Section 3: Object-Oriented Heuristics (20 pts.)**

* + 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?

I think both classes must keep the track of the information that is associated with them. The Student class must keep track of all the courses a student is enrolled in and the Course class must maintain a list of courses students are enrolled in. This is because Student and Course have a many-to-many relationship. As per OOP heuristic, each class is responsible for maintaining their own state and perform necessary operations on them.

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 above-given concept is not okay. The approach integrates multiple concepts and implementations within the same class which can be termed as the God class problem. The class RegistrationSystem will have information for all the courses, students will be responsible for the communication between them. That will be too much within the same class and the whole implementation will be very complicated. Hence, the class RegistrationSystem should not be made the God class. In such case, we can put all the logic for checking perquisites inside the Student class and now RegistrationSystem class can call the required method from the Student class (For e.g.: Student::meets\_prequisites()). Similarly, the methods for adding and dropping a course can be put inside the Course Method and handled accordingly. (H3.2)
2. 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?

For me, this is not the right design as well and is because there will be too many small classes that will act as subclasses, and logic for each will be spread over all of them. This problem as per OOP heuristics is known as the proliferation of classes problem and must be avoided. With this design, there can be methods with the same name in different classes. It will make debugging and maintaining the code difficult. Also, the subclasses should maintain the is-a relationship with the superclass. As one of the heuristics suggests that, do not turn objects of a class into the derived classes of the class. I think GradStudent can be objects of the Student class or even their subclasses but not the separate derived class itself. Also, we need to avoid using inheritance if an instance of a subclass needs to be an object of another class. Here a GradStudent can be an instance of either a Freshman, Sophomore, Junior, and Senior class. Also, the defined subclasses already refer to the actual states of the main class Student. A Freshman in a year can change to a sophomore and so on. So, it is not a good practice to create classes in such a way that the instances might need to change their type on the runtime. This design requires objects to dynamically change their type. So it is not a right design. (H5.15,

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?  
   Following different heuristics of OOP, it is a good concept for each class to be responsible for their own attributes and operations. But if multiple classes need to perform a similar kind of operation and generate a similar output format, we can create a separate class for that output. In our context, it is okay for two classes Student and Course to have a display method. This wouldn’t be much of a confusion. But if there are other classes that require the same display method to print out an HTML, we can consider Display as a noun and create a whole new class called Display so that all the logic needed to generate the HTML is handled by the Display class. The Display class can act as a View class and allow code reusability, making the design less complex and easy to maintain. Moreover, we can have a controller class that connects the model class with our view class. For e.g., Displayer. This will introduce the MVC design pattern in the project.
2. 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 proposed design sound quite reasonable as there are two classes Student and Course, that communicate with each other through the Registrar class. This is a fine approach only if the Registrar class is not maintaining any complicated logic individually related to the Student or the Course class. It must only have basic functionality that can act as a mediator between the two classes. Any complicated logic must reside within the Student and the Course class itself and the Registrar class must be able to call the required methods to perform the desired operation as requested by the Student or the Course classes.

**Section 4: Programming Problem (30 pts.)**

**5 pts. for each correct implementation: contains, overlaps, size**

**5 pts. for each corresponding Junit test. In addition, do not forget to write JUnit test cases for exception handling.**

* + 1. Use the following to create a working program in eclipse and upload to your github repository. In addition to a working program, you must provide appropriate JUnit test cases for each method in IntegerRange.

/\*

A Range objects represents an integer range, such as 1-10 or 50701-50799. The lower and

upper bounds of a Range are given at the time the object is created.

\*/

**Range.java**

public interface Range {

public boolean contains( int value );

// returns true if v is ≥ lower bound and ≤ upper bound,

// and false otherwise

public boolean overlaps( Range other ) throws EmptyRangeException;

// returns true if the receiver contains at least

// one value in common with other, and false otherwise

// EmptyRangeException is thrown when a null Range object is passed to the method

public int size();

// returns the number of integers in the range

}

**IntegerRange.java**

public class IntegerRange implements Range {

private int lower;

private int upper;

// Hint: implement getters for lower and upper

IntegerRange(int lower, int upper) { // Constructor

this.lower = lower;

this.upper = upper;

}

public boolean contains(int value) {

// implement

}

public boolean overlaps(Range other) throws EmptyRangeException {

// implement

}

public int size() {

// implement

}

}

Lastly, submit your exam to your **github** repository. You will upload your exam artifact according to the following guidelines.

1. org.howard.edu.lsp.mid-term.word (contains your exam in word format)
2. org.howard.edu.lsp.mid-term.problem51 (contains implementation and JUnit test cases)