**Final Exam Instructions**

**OBJECT-ORIENTED PROG**

* This is a take-home exam. You can use any resources that are available for you to finish this exam, except
  + Outsourcing the exam to any person or to any third party websites
  + Copying from other students work
  + Copying direct quotes from the books or internet
* Do not lose your opportunity to learn while working on the exam. Understand the concept and write answers on your own.
* Usually, in life, we have several choices. Unfortunately, you don’t have any choice on this exam. You have to answer all the questions and each part of the problem.
* All the topics on this exam were discussed in class . So, you cannot claim that the questions are out of the syllabus!
* Refer to Microsoft Word tutorials for proper formatting
* Points will be deducted for grammatical and spelling mistakes
* No two brains think alike unless you are soulmates. Definitely your answers will not be same as other students.
* Read the code of academic integrity before you start the exam. <https://www.nwmissouri.edu/policies/academics/Academic-Integrity.pdf>
* Push your source code to GitHub and provide your GitHub link at the end of the document and in the comment section.
* Don’t use examples that already explained in class or worksheets.
* Provide the input and output screenshots for every program.

**Final Exam OBJECT-ORIENTED PROG 01FA20 150 pts**

1. (20-Points) Define the terms abstract classes and interfaces. What are the similarities and differences between abstract classes and interfaces? Why interfaces are preferred over abstract classes? Explain and demonstrate with examples.

**Answer for the Question1**:

Data abstraction is the procedure of hiding the implementation and displaying limited useful data to the user. Abstraction can be accomplished with either abstract classes or interfaces.

An **Abstract class** is a class which is declared with the keyword Abstract. It is a restricted class which cannot be used for creating objects.

A boundary or interconnection between two entities is referred to as an **interface**. Interfaces specify what a class should do but not how. It is the blueprint of the class. In an interface, all the methods are abstract and public. Interfaces can have constants but no instance variables. Recently, from java 1.8 default methods are introduced in the interfaces. A class implementing an interface, inherits the abstract methods of the interface.

|  |  |
| --- | --- |
| **Abstract Class** | **Interface** |
| Similarities:   1. Abstract classes have abstract methods. 2. Abstract classes cannot be instantiated. 3. Don’t have default constructor. 4. All the abstract methods must be overridden by subclass. | Similarities:   1. Interfaces can have abstract methods. 2. Interface can not be instantiated. 3. Don’t have default constructor. 4. All the methods of the interface must be overridden by the implementing sub class. |
| Differences:   1. an abstract class can be extended using the keyword “extends”. 2. Abstract class does not support multiple inheritance as we cannot extend more than one class. 3. Abstract classes can have normal non-abstract methods. 4. No methods are created default with the word ‘default’. 5. An abstract class may contain non-final variables. 6. Abstract class can have final, non-final, static and non-static variables. 7. Abstract class can provide the implementation of the interface. | Differences:   1. A Java interface can be implemented using the keyword “implements”. 2. We can achieve multiple inheritance using interface as we can implement multiple interfaces to any class. 3. Interfaces cannot have non-abstract methods. 4. Interfaces can have default methods created with the word ‘default’. 5. Variables declared in a Java interface are by default final. 6. The interface has only static and final variables. 7. Interface can’t provide the implementation of an abstract class. |

Multiple inheritance isn't supported in Java, so you can't have a class that implements two abstract classes at the same time. When you want several classes to implement the interface and don't want to inherit default behavior, an interface is **preferable** to an abstract class.  **Interfaces** are better in situations in which you do not need to inherit implementation from a base class.

Example 1 :

1. (10-Points) Design an interface named Colorable with a void method named howToColor(). Every class of a colorable object must implement the Colorable interface. Design a class named Square that extends GeometricObject and implements Colorable Implement howToColor to display the message Color all four sides.

Draw a UML diagram that involves Colorable, Square, and GeometricObject. Write a test program that creates an array of five GeometricObjects. For each object in the array, display its area and invoke its howToColor method if it is colorable.

1. (10-Points) What is casting? What are different types of casting? Explain and demonstrate with examples.
2. (15-Points) Suppose that Fruit, Apple, Orange, GoldenDelicious, and McIntosh are defined in the following inheritance hierarchy:

Fruit

Orange

Apple

GoldenDelicious

McIntosh

Assume that the following code is given:

Fruit fruit = new GoldenDelicious();

Orange orange = new Orange();

Answer the following questions and explain why these Statements are legal or illegal.

a. Is fruit instanceof Fruit?

b. Is fruit instanceof Orange?

c. Is fruit instanceof Apple?

d. Is fruit instanceof GoldenDelicious?

e. Is fruit instanceof McIntosh?

f. Is orange instanceof Orange?

g. Is orange instanceof Fruit?

h. Is orange instanceof Apple?

i. Suppose the method makeAppleCider is defined in the Apple class. Can fruit invoke this method? Can orange invoke this method?

j. Suppose the method makeOrangeJuice is defined in the Orange class. Can orange invoke this method? Can fruit invoke this method?

k. Is the statement Orange p = new Apple() legal?

l. Is the statement McIntosh p = new Apple() legal?

m. Is the statement Apple p = new McIntosh() legal?

1. (10-Points) Define a class named ComparableCircle that extends Circle and implements Comparable. Draw the UML diagram and implement the compareTo method to compare the circles on the basis of area. Write a test class to find the larger of two instances of ComparableCircle objects.
2. (15-Points) What is an exception? What are checked and unchecked exceptions? Explain and demonstrate with examples.
3. (10-Points) Write a program that meets the following requirements:

* Creates an array with 100 randomly chosen integers.
* Prompts the user to enter the index of the array, then displays the corresponding element value. If the specified index is out of bounds, display the message Out of Bounds.

1. (10-Points) What is the purpose of declaring exceptions? How do you declare an exception, and where? Can you declare multiple exceptions in a method header? Explain and demonstrate with examples.
2. (10-Points) What is the keyword throw used for? What is the keyword throws used for? Can you throw multiple exceptions in one throw statement? Explain with examples.
3. (15-Points) What is a recursive method? What is an infinite recursion? Explain and demonstrate with examples. Implement the search (element) in a list using recursion.
4. (10-Points) Write a java program that illustrates how equals() and hashCode() methods work? Explain your code in comments.
5. (15-Points) Design Employee class and Employee driver class as follows:
6. **Employee Class implements Comparable<Employee**>

* Data fields named empId, empName and empSalary
* A constructor with parameters, listed in the same order as above.
* Create getter methods for all the parameters.
* A toString method that prints the empId, empName and empSalary. There should be one space between each value output.
* Because Employee implements the Comparable interface, you must also implement the compareTo method as defined by the Comparable interface. Define this method in such a way that the natural ordering of employees will be by id number, in ascending order.

1. **EmployeeDriver Class**

* Begin by filling an ArrayList with at least 5 employees. Add employees in random order – not by id number, not by name, and not by salary. The original list should not be in order by any of these attributes.
* Use an enhanced for loop to print the original list.
* Call the one-parameter sort method of the Collections class to sort the list by its natural order (empId number) and then print the list again.
* Call the two-parameter sort method of the Collections class, supplying a new Comparator<Employee> that sorts by salary. Print the list again.
* Call the two-parameter sort method of the Collections class, supplying a new Comparator<Employee> that sorts by name. Print the list again.