CSCI 185 Midterm Exam - Spring 2022 - March 31, 2022

- Due No due date
- Points 110
- Questions 19
- Available Mar 31, 2022 at 9am - Mar 31, 2022 at 12:30pm 3 hours and 30 minutes
- Time Limit 120 Minutes

Instructions

Please note that the midterm is closed book, closed notes, and any type of chatting and/or discussion among students is NOT allowed. If any academic integrity violation is identified, the student will get an immediate zero for the midterm exam. If it is a repeated violation, the whole incidents will be reported to department chair, the Dean, and the campus Dean of Students.

For the Java programming questions, please type your answer to Canvas directly, or copy and paste the program from a text editor (such as notepad, wordpad, notepad++, etc.). Please be aware that it is NOT allowed to use any type of Java Integrated Development Environment (IDE) tools (such as Eclipse, NetBeans, BlueJ, IntelliJ, and so on).

This quiz was locked Mar 31, 2022 at 12:30pm.

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	120 minutes	89 out of 110

Score for this quiz: 89 out of 110

Submitted Mar 31, 2022 at 11:04am

This attempt took 120 minutes.

Question 1

0 / 2 pts

We can use **Object_name.Member_name** to access all **static** class members in Java.

You Answered

● True

Correct Answer

○ False

Question 2

2 / 2 pts

An object is an instance of a class.

Correct!

● True

○ False

Question 3

2 / 2 pts

Passing objects to a method is "**passing by value**".

○ True

Correct!

● False

Question 4

2 / 2 pts

Making a copy of an object requires a special method called a copy constructor.

Correct!

● True

○ False

Question 5

2 / 2 pts

Inheritance models the "**has a**" relationship.

○ True

Correct!

● False

Question 6

2 / 2 pts

The term “**early** binding” has the same meaning as “**dynamic** binding”.

○ True

Correct!

● False

Question 7

2 / 2 pts

Suppose we create two objects **a1**, **a2** out of the class ***A***, then we can use the statement **a1 = a2** to make a **deep copy** of a2 and save it in a1.

○ True

Correct!

● False

Question 8

2 / 2 pts

Default constructor should still have at least one parameter as its input.

○ True

Correct!

● False

Question 9

2 / 2 pts

Which of the following is **true** for constructor?

○ We can only have one constructor per each class.

○ Constructor cannot be overloaded.

Correct!

● Constructor has no return type.

○ The constructor's name should be different from the class name.

Question 10

2 / 2 pts

Which of the following two classes should be implemented using composition?

○ Software Engineer <-> IT Professional

Correct!

● JFK Airport <-> IATA code

○ University <-> Educational_Institute

○ IT_Company <-> Business_Organization

Question 11

2 / 2 pts

If an instance variable or a method is marked as **protected**, then it can be accessed ________________ _

○ in this class itself

○ in any derived class of this class

○ in any class within the same package

Correct!

● ALL of the above

Question 12

2 / 2 pts

Which of the following methods will be applied **late** binding by Java?

○ a method marked as ”final”

○ a method marked as ”private”

○ a static method

Correct!

● NONE of the above

Question 13

7 / 10 pts

What do **abstraction** and **encapsulation** mean in Java? Please also describe the reason why we should maintain these two properties for Java.

Your Answer:

Abstraction is when you share generalized code without sharing its implementation. It is a useful code reuse strategy that allows us to specify the behavior of specific classes that fall into the same base class. In Java, abstraction is done through abstract classes, where the "abstract" keyword must be stated in the class declaration (public abstract class <class_name>) and can only be instantiated in the derived class. The derived class inherits all the members of the abstract class and defines its abstract methods.

Encapsulation in Java is when you hide the members of a class using modifiers (public, private, protected, final). By controlling the level of access to its instance variables (or methods), you are able to control who can access and/or modify data. Generally, data fields can be declared private and only be accessible through accessor-mutator methods to maintain privacy.

Both of these properties help protect privacy and maintain the stability of data. Without encapsulation, any class could access and/or modify object data. Similarly, abstraction protects privacy by hiding the implementation of code from the base class and other derived classes.

Abstraction does not necessarily mean abstract class. -3 pts

Question 14

8 / 8 pts

What are the major differences between **early** and **late** bindings? Which of them is used to implement **polymorphism**?

Your Answer:

Early binding is when an object knows all important data at compile time. In early binding, you cannot add new methods and variables to a derived class of the superclass.

Late binding is when an object does not know everything at compile time. Late binding means binding code later so that you can add new methods and variables to the derived class of the object.

Polymorphism uses late/dynamic binding to specify behavior in its derived classes.

ok

Question 15
9 / 9 pts
What are the major differences between **method overloading** and **method overriding**? Please use some **Java programs** as an example to show how each of them works.
Your Answer:

Method overloading is when two methods have the same name different parameters list, so they can be differentiated based on the arguments passed to the method. Method overloading allows us to perform different behavior based on the method that is being called. An example is the fully-loaded constructor and copy-constructor of the Student class - one creates a new object given its data fields' values (String name, int age, int gpa), while the other creates a new object that is deep copy of another object.

```
public Student (String name, int age, int gpa){

  this.Name = name;

  this.Age = age;

  this.GPA = gpa;
}

public Student (Student A){

  this.Name = A.Name;

  this.Age = A.Age;

  this.GPA = A.GPA;
}
```

Method overriding is when a method of the derived class writes over a method of the base class. The derived class method must have the same parameters in order to override the base-class method. An example is the method numDaysOff, where the base-class method numDaysOff and derived-class method numDaysOff have the same input (void) but different output.

```
//base-class method
```

```
public int numDaysOff (){

return 0;
}

//derived-class method that returns days off of professor

public int numDaysOff (){
return 30;
}
```

ok

Question 16

9 / 9 pts

Name the three code reuse techniques in object-oriented programming not including copy-paste. Briefly describe how each of them works.

Your Answer:

Composition models a "has a" relationship. It is when a class uses an object of another class in its specified behavior. For example, each Course object has at least one Student object, and the Course class uses the Student class to instantiate the Student object.

Inheritance models a "is a" relationship. It uses a base class to define derived classes with more specific behavior. For example, the base-class Animal is used by the derived-class Monkey to specify the behavior of a specific type of animal called a monkey. The derived class inherits all of the base class's members (but doesn't necessarily have access).

Polymorphism is when a base-class method is overridden by the derived class to specify a different behavior. For example, a "eat" method may be defined in the animal class in which the animal eats a cantaloupe. In the derived class, this method may be modified for the monkey to eat a banana.

Good

Question 17

18 / 20 pts

Define a base class **Driver** and a derived class **Taxi_Driver** in Java. The class **Driver** should contain the data fields of **name** (String), **lic_number** (String), and **lic_class** (String). The class **Taxi_Driver** should contain additional data fields of **taxi_model** (String), **night_shift** (boolean), and **yrs_exp** (int). For both classes, you should maintain the **privacy** of each data field, and declare one fully loaded constructor, one copy constructor, accessor and mutator (only for one data field) and toString() method. Also declare a main method that creates one taxi driver object and print it out.

Your Answer:

```
public class Driver {
   private String name;
   private String lic_number;
   private String lic_class;
   //constructor
   public Driver (String n, String ln, String lc){
      this.name = n;
      this.lic_number = ln;
      this.lic_class = lc;
   }
   //copy-constructor
   public Driver (Driver A){
      this.name = A.name;
      this.lic_number = A.lic_number;
      this.lic_class = A.lic_class;
   }
   //accessor and mutator for one field only
   public void setName (String n){
      this.name = n;
   }
   public String getName (){
      return this.name;
   }
   //toString method
   public String toString (){
      String output = "Driver info:\n******************************\n";
      output += "Name: " + this.name;
      output += "\nLIC number: " + this.lic_number;
      output += "\nLIC class: " + this.lic_class;
      return output;
   }
}
public class Taxi_Driver extends Driver{
   private String taxi_model;
   private boolean night_shift;
   private int yrs_exp;
   //constructor
   public Taxi_Driver (String n, String ln, String lc, String tm, boolean ns, int ye){
      super (n, ln, lc);
```

```
        this.taxi_model = tm;
        this.night_shift = ns;
        this.yrs_exp = ye;
    }
    //copy-constructor - assume I've defined all the get methods
    public Taxi_Driver (Taxi_Driver A){
        super.toString (A.getName(), A.getLICnumber(), A.getLICclass());
        this.taxi_model = A.taxi_model;
        this.night_shift = A.night_shift;
        this.yrs_exp = A.yrs_exp;
    }
    //accessor and mutator for one field only
    public void setTaxiModel (String t){
        this.taxi_model = t;
    }
    public String getTaxiModel (){
        return this.taxi_model;
    }
    //toString method
    public String toString (){
        String output = super.toString();
        output += "Taxi Model: " + this.taxi_model;
        output += "\nNight Shift?: ";
        if (this.lic_number == true) output += "Yes";
        else output += "No";
        output += "\nYears till expiry: " + this.yrs_exp;
        return output;
    }
}
public class DriverTest {
    public static void main (String [] args){
        Taxi_Driver Bill = new Taxi_Driver ("Bill", "34343", "first-class", "ford model t", false, 5);
        System.out.println(Bill.toString ());
    }
}
```

super.toString (A.getName(), A.getLICnumber(), A.getLICclass()) This statement is not correct. -2 pts

Question 18

13 / 15 pts

Define a Java class named **Taxi_Company** and integrate the **Taxi_Driver** class that was already defined in this exam as an instance variable in this class using **composition**. In addition, define one fully loaded constructor, and the accessor, mutator (for **list_drivers** field), and toString() methods. Also define a main method that will create one taxi company object with at least three taxi drivers, and print the taxi company out.

```
public class Taxi_Company {

           //instance variables
           private String company_name;
           private int number_taxi_cabs;
           private Taxi_Driver[] list_drivers;

}
```

Your Answer:

```
public class Taxi_Company {
   //instance variables
   private String company_name;
   private int number_taxi_cabs;
   private Taxi_Driver[] list_drivers;
   //fully-loaded constructor
   public Taxi_Company (String cn, int ntc, Taxi_Driver [] list_d){
      this.company_name = cn;
      this.number_taxi_cabs = ntc;
      this.list_drivers = new Taxi_Driver [list_d.length];
      for (int i = 0; i < list_d.length; i++){
         this.list_drivers[i] = new Taxi_Driver (list_d[i]);
      }
   }
   //accessor, mutator for list_drivers
   public void setListDrivers (Taxi_Driver [] list_d){
      this.list_drivers = new Taxi_Driver [list_d.length];
      for (int i = 0; i < list_d.length; i++){
         this.list_drivers[i] = new Taxi_Driver (list_d[i]);
      }
   }
   public Taxi_Driver [] getListDrivers () {
      Taxi_Driver [] temp = new Taxi_Driver [list_drivers.length];
      for (int i = 0; i < list_drivers.length; i++){
         temp[i] = new Taxi_Driver (list_drivers[i]);
      }
   }
```

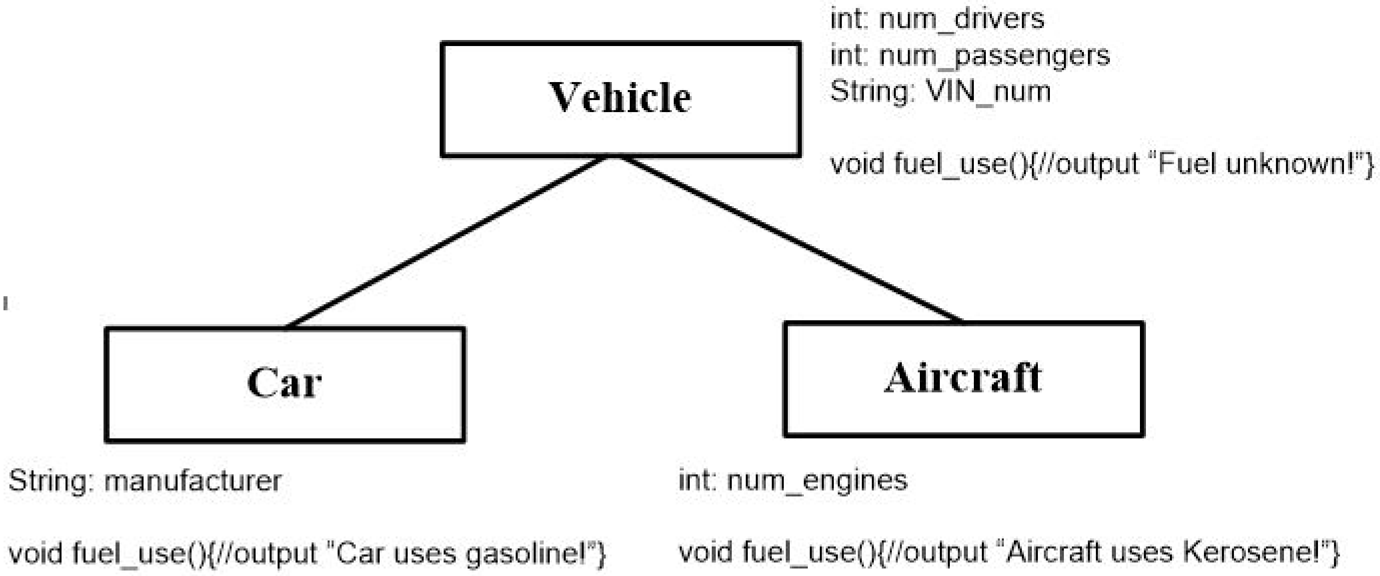
```
    //toString
    public String toString (){
        String output ++= "Taxi Company Info: \n\n";
        output += "Company Name: " + this.company_name;
        output += "\nNumber of Taxi Cabs" + this.taxi_cabs;
        output += "\nDrivers Info:";
        (for int i = 0; i < list_drivers.length; i++){
            output += "No." + (i+1) + "\n" + list_drivers[i].toString();
        }
        return output;
    }

}



public class CompanyTest {
    public static void main (String [] args){
        Taxi_Driver Bill = new Taxi_Driver ("Bill", "34343", "first-class", "ford model t", false, 5);
        Taxi_Driver Joe = new Taxi_Driver ("Joe", "34343", "first-class", "ford model t", false, 5);
        Taxi_Driver Byron = new Taxi_Driver ("Byron", "34343", "first-class", "ford model t", false, 5);
        Taxi_Driver [] list_1  = {Bill, Joe, Byron};
        Taxi_Company tc = new Taxi_Company ("TC", 5, list_1);
    }
}
```

public Taxi_Driver [] getListDrivers () { Taxi_Driver [] temp = new Taxi_Driver [list_drivers.length]; for (int i = 0; i < list_drivers.length; i++){ temp[i] = new Taxi_Driver (list_drivers[i]); } } missing return -2 pts

Question 19
3 / 15 pts

Based on the “Vehicle” class hierarchy below, **define a “test” class in Java** that uses the **polymorphism** technique to determine what fuel each specific type of vehicle will use. In you “test” class, you should define a method called **what_to_refill** with the parameter **(Vehicle v)**. Then in the main method, create an object of Car and an object of Aircraft and decide what fuel each of them uses using the method **what_to_refill**. (15 points)

Note: you can assume that all three classes in the Vehicle hierarchy are well defined and ready to use.


int: num_drivers
int: num_passengers
String: VIN_num
void fuel_use(){//output "Fuel unknown!"}
Vehicle
Car
Aircraft
String: manufacturer
void fuel_use(){//output "Car uses gasoline!"}
int: num_engines
void fuel_use(){//output "Aircraft uses Kerosene!"}

Your Answer:

```
public class VehicleTest{
   public String getFuel (Vehicle A){
      Vehicle.fuel_use();
   }
```

The code is not complete. -12 pts

Quiz Score: 89 out of 110