DUE DATE: **By the end of your lab session**

CSE-271: Object-Oriented Programming

Exercise #3

Max Points: 20

Name: Jacob Igel

For your own convenient reference – You should first save/rename this document using the naming convention `MUid_Exercise3.docx` (example: `raodm_Exercise3.docx`) prior to proceeding with this exercise.

Objectives: The objectives of this exercise are to:

1. Understand the concept associated with object-oriented programming
2. Gain familiarity with Java's approach for developed classes
3. Developing Java classes with constructors, getter, and setter method(s)
4. Using custom classes (developed by you) in a Java program

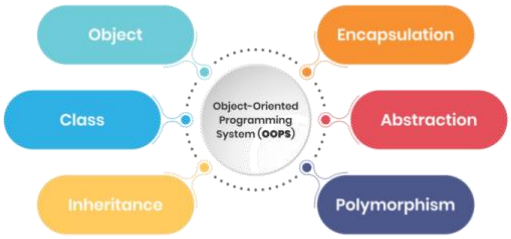
Fill in answers to all of the questions. For some of the questions you can simply copy-paste appropriate text from Eclipse output into this document. You may discuss the questions or seek help from your neighbor, TA, and/or your instructor.

Part #0: One time setup of Eclipse (IDE) – Only if needed

We already configured `Eclipse`'s source formatter and Checkstyle plug-in as part of Lab #1. If your `Eclipse` is not configured (because you are using a different computer) then use the instructions from Lab #1 to configure `Eclipse`.

Part #1: Generic concepts of Object-oriented programming

Estimate time: < 20 minutes



Background: Object-oriented Programming (OOP) is a programming paradigm that is widely used and adopted by several mainstream programming languages, such as: C++, C#, JavaScript, Java, and Python. Hence, understanding the generic concepts underlying OOP is very important for your future careers, immaterial of the programming-language that you may be working with. Moreover, clearly and concisely explaining concepts is a very important skills for your future job-interviews and in your jobs. Hence, the exams also involve such questions, and in the labs, we will practice this style of questions.

Exercise: Briefly (2-to-3 sentences each) respond to the following questions regarding generic concepts of object-oriented programming (OOP).

1. What is object-oriented programming (OOP)?

   Object-Oriented Programming is a "computer programming model that organizes software design around data, or objects, rather than function and logic. An object can be defined as a data field that has unique attributes and behavior."

   *Source

2. List the 4 tenants (or principles) of OOP? Just list the 4 words/terms and that is sufficient for this specific question in this specific lab (obviously, in job situations you should elaborate and provide good examples)

   - Encapsulation
   - Abstraction
   - Inheritance
   - Polymorphism

   *Source

3. Briefly describe (2-to-3 sentences max) the concept of "encapsulation" and how it is achieved in object-oriented programming (OOP)?

   Encapsulation is all of the important information that is contained inside an object that only has select information exposed. Each object is stated and implemented privately inside a defined class.

   *Source

4. Briefly describe (2-to-3 sentences max) the concept of "abstraction" and how it is achieved in object-oriented programming (OOP)

   Abstraction is used for methods to "hide" or abstract details of internal operations. The derived class can have its functionality extended and it helps developers make additional changes over time.

   *Source & In-Class PowerPoint

Part #2: Basics of Java classes

Estimated time: < 20 minutes

Background: The generic OOP concepts (covered in previous part) are supported by different object-oriented programming-languages in slightly different ways. Java has a specific approach

for supporting the 4-key tenants of OOP using Java classes. However, similar syntax and semantics also apply to other object-oriented programming-languages.

Exercise: Briefly (2-to-3 sentences each) respond to the following questions regarding Java classes.

1. Briefly (2-to-3 sentences) describe the difference between the access modifiers `public` and `private`.

   Private modifiers are attributes that are only accessible within the class. Typically, we use private modifiers for instance variables or helper methods. *Public* modifiers are accessible inside and outside the class. These include relevant static and non-static methods.

2. A key aspect of OOP is using the appropriate access modifiers for various attributes to ensure good encapsulation and abstraction to communicate the intention of the design. Accordingly, add access modifiers (i.e., `public` or `private`) for the different attributes of the `Person` class shown in the adjacent figure.

```
class Person {
    // Social security number
        private     String ssn;

        // Email address. Send me email
        public      String email;

        // Get to know me.
        public      String getName() {}

        // Change/correct social
        private     void correctSSN() {}
}
```

   For each attribute state why, you chose that specific access modifier (the "why" is very important)

   a. `ssn`: This variable will not be lost once the method is completed.
   b. `email`: Initializes a variables
   c. `getName`: Initializes a variables
   d. `correctSSN`: This variable will not be lost once the method is completed.

3. Briefly (2-to-3 sentences) describe the difference between static and non-static instance variables.

   Static variables acts as a global variable and is shared among all the objects of the class. Non-static variables are specific to the instance object in which they are created.

   *Source

4. What is the `this` variable in Java and how is its value determined?

> "The object on which the method is being invoked is called this within a non-static method. Every non-static method has an implicit local object than can be referred to using the *this* keyword."
>
> *PowerPoint

Part #3: Develop & test a custom Java class

Estimated time: 40 minutes

Background: Given a brief description or requirements, developing a class is a common operation that you will routinely perform, including in your internships and jobs. Hence, it is important to become adept at using description to develop a class. In addition, it is important to test the methods in your class to ensure they operate correctly. The process of testing each method is called unit testing. Later on, we will learn to use a special framework called `Junit` that streamlines unit testing in Java.

In this part of the exercise, you will be guided to systematically develop methods and test the method(s) as you develop each one of them. In the future, you are **expected to follow the same process <u>without having to be reminded</u>**!

Exercise: In this part of the exercise, you will be guided through a series of steps to develop and test a simple `Car` class.

Task 3.1: Create & setup Java project in Eclipse

1. Create a Java project called `Lab3` in Eclipse.
2. Download the supplied `CarTester.java` to your Eclipse project. The `CarTester.java` is a simple starter file. You will need to modify `CarTester.java` as you proceed through this laboratory exercise.

Task 3.2: Create the basics of a `Car` class

In your Eclipse project, create a new Java class called `Car`. It **<u>should not</u>** have a `main` method. Next add the two instance variables and two constructors shown below. This exercise is a recap of the topics discussed in the lecture. Hence, reviewing lecture materials and keeping up with reading is an indispensable success skill in this course.

Data/Instance variables to add to your `Car` class:	
`private String make;`	This instance variable is used to maintain the make (such as: "Honda", "Toyota" etc.) of a "Car" object.

`private int mileage;`	This instance variable is used to track the distance (in miles) that a car has traveled.
Constructor (Methods) to add to your `Car` class:	
`public Car(String theMake, int theMileage)`	This constructor should initialize the `make` and `mileage` of the `Car` object to the corresponding values specified as parameters.
`public Car(String theMake)`	This constructor should initialize the `make` to the value supplied as parameter and initialize `mileage` to zero.

Testing:
Once you have implemented and **documented (using Javadoc)** the constructors, switch to `CarTester.java` and find the comments where you are asked to create two `Car` objects. Write this code. Save the Java file(s) and Eclipse should not report any errors. It is important that you complete this task without any errors before proceeding to the next task.

Task 3.3: Implement a couple of getters (aka accessor) methods

In Java, methods that essentially return the values of instance variables are called getter or accessor methods. The `Car` class contains two instance variables, namely: `make` and `mileage`. Accordingly, implement the following two getter methods:

Getter/Accessor Methods:	
`public String getMake()`	This method (a 1 liner) simply returns the `make` of the `Car`.
`public int getMileage()`	This method (a 1 liner) simply returns the `mileage` of the `Car`.

Testing:
Once you have implemented and **documented (using Javadoc)** the 2 methods, switch to `CarTester.java` and find the comments where you are asked to test these two methods. Write this code and run the tester to ensure the outputs are as expected.

Task 3.4: Implement two straightforward methods

Once you have successfully completed and tested the previous tasks, implement the following methods:

Methods:	
`public void drive(int distance)`	This adds the given `distance` to the current `mileage` of the `Car`. The `distance` value can be positive or negative (for driving in reverse) and needs to be suitably handled to always *increase* the `mileage` by the given `distance`. In other words, the mileage should never decrease, even if the car is driven in reverse. (See: Ferris Bueller's Day Off for further clarification.)
`public void honkHorn()`	This method (a 1 liner) simply prints the make of the car followed by the phrase `": beep"`. For example, if the `make` of the `Car` is "`Toyota-Corolla`", this method should print `Toyota-Corolla: beep` (followed by a new line).

Testing:
Once you have implemented and **documented (using Javadoc)** the above methods, switch to `CarTester.java` and find the comments where you are asked to test these two methods. Write this code and run the tester to ensure the outputs are as expected. Note that in order to test the `drive()` method, you should drive the car AND print its resulting mileage.

Task 3.5: Implement the toString method

The `toString` method is a standard Java method that must be implemented in a Java class to conveniently create user readable information about an object. Typically, a `toString` method will put together all of an object's data in some useful format.

Methods:	
public String toString()	This method should return a single `String` that has the `make` and `mileage` value separated by a `": "` (example: `Toyota-Corolla: 20000`). See sample output below.

Testing:
Once you have implemented and **documented (using Javadoc)** the above method, switch to `CarTester.java` and find the comments where you are asked to test this method. Write this code and run the tester. If you have followed the steps up to here, then the output of those two lines of code should be as shown below:

Expected output:

```
Toyota-Corolla: 10223
Honda-Accord: 20556
```

Part #5: Submit to Canvas via CODE plug-in

Estimated time: < 5 minutes

Exercise: You will be submitting the following files via the Canvas CODE plug-in:

1. This MS-Word document saved as a PDF file – **Do not upload Word documents. Only submit PDF file**.
2. The Java source files `Car.java` and `CarTester.java` that you developed in this exercise

Ensure you actually complete the submission on Canvas by verifying your submission