Name: \_\_\_Eyad Alsahori\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |
| --- |
| **ASSIGNMENT GOALS:**   1. **Model/Implement an entity of your own choosing:**    1. **Its Structure**    2. **Its Functionality** 2. **Gain more experience with custom Classes.** 3. **Use UML to document/communicate your design.** |

(20 Points)

1. **Read Pragmatic Programmer Textbook – start of Chapter 2 pp 27 ~ 37 (a Pragmatic Approach)**
2. **Read Java Textbook Chapter 6 - “317 – 389”.** Much of this material will be a review of what was covered in class, but it’s a worthy reinforcement. (20 Points)

Complete the following exercises at the bottom of page 393 (**short answers**):

2)

class is a blueprint and house is an instance, they are both correct

4)

it is a good idea because no body will then have access to it directly, therefor we can avoid modifications

9)

The circumstance is that If there is no other constructor the compiler will make one

11)

when using method overloading, Java can knows the difference between methods with different method signatures. The methods within a class can have the same name if they have different parameter lists.

**(2a)**

**Model a physical entity as a Java Class (70 points) – focus on Encapsulation**:

NOTE package up programming assignment Classes under: **edu.cuny.csi.csc330.lab4**

**This is an Object Modeling/Programming exercise – albeit around 1 (maybe 2) Object(s).** Using the Radio Class implementation provided as a point of reference, think of a common (or maybe, not so common) device or appliance that’s *worthy* of modeling as an Object. Consider the following:

* 1. The basic operations/functionality of your device/appliance become public methods of the implementing Class (e.g., on, off, go, start, stop, … )
  2. The internal structure (or how you envision the internal structure) of your device/appliance become the data members of the implementing Class. This would include the internal piece-parts of the device and well as “state components”. For example, modeling a bread toaster might require a “heat coil” as well as a “power on/off” indicator.
  3. In the spirit of modularity, reusability, etc., private/protected methods can and should be used to implement sub-tasks performed by public methods.

In addition to Class members supporting (a) (b) (c) above, you also need to implement:

* Either a **public void main(String [] ){}** method or a second “launch” Class whose only job will be to:
  + Create an instance of your modeled Class
  + Invoke a sequence of methods on that instance that demonstrates the functionality and basic usefulness of what you built.
* A “String toString()” method – allowing you to display ‘the state’ of the Object Instance.

What do we mean by **the state** of the Instance? The collective value of all data members ‘at a moment in time’ is considered the instance’s **state**. *That’s a valid definition of state, but for this lab, customize the Class’ toString() method to display values of data members that are considered significant to the device/application’s operation.*

**Note**: Refer to the Radio Class’ *toString()* and *main()* methods as a guide. Also refer to the Radio’s execution output below – showing that a single instance has been created, and taken through a series of operations (method invocations). At critical points, a “toString()” representation of the instance’s state is displayed to the console using System.out.println()or like method.

|  |
| --- |
| **Slight variant: or, you can choose to model a simulation that proves a hypothesis.** Something similar to the Classes implemented under our edu.cuny.csi.csc330.stats package …  To be expanded on in class, but the general idea is to model the proof of a hypothesis that may be a bit counterintuitive or difficult to verbally explain as a proof. |

**Replace the output below with yours!**

|  |
| --- |
| Initial State:  Car [carBrand=null, destination=null, Passengers=0, speed=0.0mph  startdrive=false, GPSTime=null  carstoppedmoving=false,  Is driving: false.  After Loading passengers  Car [carBrand=Honda Pilot, destination=Newyork, Staten Island, Passengers=3, speed=0.0mph  startdrive=false, GPSTime=null  carstoppedmoving=false,  Is driving: false.  After the car starts moving  Car [carBrand=Honda Pilot, destination=Newyork, Staten Island, Passengers=3, speed=25.0mph  startdrive=true, GPSTime=Mon Oct 31 19:49:16 EDT 2022  carstoppedmoving=false,  Is driving: true.  After the car stop moving  Car [carBrand=Honda Pilot, destination=Newyork, Staten Island, Passengers=3, speed=0.0mph  startdrive=false, GPSTime=Mon Oct 31 19:49:16 EDT 2022  carstoppedmoving=true,  Is driving: false. |

**(2b) UML Class Diagram (10 points)**

Provide a **simple** UML Class diagram that documents the structure and functionality of the Class (or Classes) you implemented.

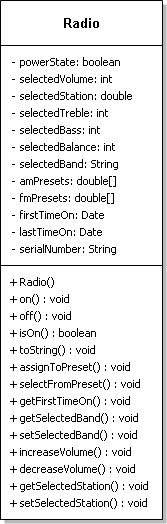
* 1. Class Name
  2. Non-public members (-) or (#)
  3. Public Methods (+)

**Note: You may hand-draw and scan the image, or use a diagramming tool like NClass, Visio, draw.io, etc. or better yet, an online tool like** <https://www.lucidchart.com/pages/> (LucidChart)  **See Radio Example below:**

MY UML Class diagram is shown below:

**Text, table

Description automatically generated**



**Some example device/entities worthy of modeling as a Java Class:**

1. **Just about any household appliance**
2. **A simple communication device**
3. **Moving Ground or Air Vehicle**
4. **A living organism – *real or imagined*.**
5. **A secondary computer device (disk, printer/scanner/fax, etc.).**
6. **Musical Instrument or Related Device**
7. **A non-existent device such as: Time-travel Machine**