**Object Oriented Programming in Java**

1. **Constructor challenge:**

* Create a new class for a ***Bank Account***.
* Create fields for the **Account Number**, **Balance**, **Customer Name**, **Email** and **Phone Number**.
* Create **getters** and **setters** for each field.
* Create **two additional methods**: to allow the customer to deposit the founds (this should increment the balance field) and to allow the customer to withdraw founds (this should deduct from the balance fields but not allow the withdrawal to complete if their founds are insufficient).
* You will want to create various code in the main class (the one created by IntelliJ) to confirm your code is working. Add some ***System.out.println()***’s in the two methods above as well.
* Create a new class ***VipCustomer*** it should have 3 fields ***name***, ***credit limit***, and ***email address***.
* Create 3 constructors.
* 1st constructor empty should call the constructor with 3 parameters with default values.
* 2nd constructor should pass on the 2 values it receives and add a default value for the 3rd.
* 3rd constructor should save all fields.
* Create ***getters*** only for this using code generation of IntelliJ as ***setters*** won’t be needed.
* Test and confirm it works.

1. **Inheritance challenge:**

* Create a base class of a ***Vehicle***, then create a ***Car*** class that inherits from this base class. Finally create another class, a specific type of car that inherits from the ***Car*** class.
* You should be able to **hand steering**, **changing gears** and **moving** (speed in other words).
* You will want to decide where to put the appropriate state and behaviors (fields and methods).
* For your specific type of vehicle, you will want to add something specific for that type of car.

1. **Encapsulation challenge:**

* Create a class and demonstrate proper encapsulation techniques. The class will be called ***Printer*** and will simulate a real computer printer.
* It should have fields for ***Toner Level***, ***Number of Pages Printed*** and also whether it’s a ***Duplex Printer*** (capable of printing on both sides of the paper).
* Add methods **to fill up the toner** (up to a maximum of **100%**).
* Add another method **to simulate printing a page** (which should **increase** the number of pages printed).
* Decide on the scope whether to use constructors and anything else you think is needed.

1. **Polymorphism challenge:**

* We are going back to the ***Car*** analogy.
* Create a base class called ***Car***. It should have a few fields that would be appropriate for a generic ***Car*** class (as ***engine***, ***cylinders***, ***wheels*** etc.).
* Constructor should initialize ***cylinders*** (number of) and name and set ***wheels*** to 4 and ***engine*** to true. Cylinders and names will be passed parameters.
* Create appropriate ***getters***.
* Create some methods like ***start engine***, ***accelerate*** and ***brake***. Show a message for each in the base class.
* Now create 3 subclasses for your favorite vehicles.
* ***Overwrite*** the appropriate methods to demonstrate polymorphism in use.
* Put all classes in one java file.

1. **Object Oriented Programming Master Challenge:**

* The purpose of the application is to help a fictitious company called ***Bills Burgers*** to manage their process of selling hamburgers.
* Our application will help Bill to select types of burgers, some of the additional items (additions) to be added to the burgers and pricing.
* We want to create a ***base hamburger***, but also two other types of hamburgers that are popular ones in Bills store.
* The basic hamburger should have the following items:
  + ***bread roll type***
  + ***meat***
  + ***and up to 4 additional additions*** (things like lettuce, tomato, carrot, etc.) that the customer can select to be added to the burger.

Each one of these items gets charged an additional price so you need some way to track how many items got added and to calculate the price (for the base burger and all the additions).

* This burger has a ***base price*** and the ***additions are all separately priced***.
* Create a ***Hamburger*** class to deal with all the above.
* The constructor should only include the roll type, meat and price.
* Also create two extra varieties of Hamburgers (classes) to cater for
  + ***Healthy burger*** (on a brown rye bread roll), plus two addition items can be added. The healthy burger can have a total of 6 items (Additions) in total.

HINT: you probably want to process the 2 additional items in this new class, not the base class, since the 2 additions are only appropriate for this new class.

* + ***Deluxe hamburger*** - comes with chips and drinks as additions, but no extra additions are allowed.

HINT: You have to find a way to automatically add these new additions at the time the deluxe burger object is created, and then prevent other additions being made.

* All 3 classes should have a method that can be called anytime to show the base price of the hamburger plus all additional items, each showing the addition name, and addition price, and a grand total for the burger
* For the two additional classes this may require you to be looking at the base class for pricing and then adding totals onto that.