Assignment 1

Refer to Canvas for assignment due dates for your section.

Objectives:

- Design classes in Java.
- Write immutable classes.
- Get to know the course tools.
- Unit testing with JUnit 4.
- Use Gradle tools to improve your code.

General Requirements

Create a new Gradle project for this assignment in your course GitHub repo. Make sure to follow the instructions provided in "Using Gradle with IntelliJ" on Canvas.

Create a separate package for each problem in the assignment. Package names must be all lowercase (e.g. problem1 rather than Problem1). Create all files in the appropriate package.

To submit your work, push it to GitHub and create a release. Refer to the instructions on Canvas.

Problem 1

Consider class `Vehicle`, presented on the next page. Add this class to your project.

1. Determine if the `Vehicle` class is functionally correct. If it is, proceed to the next step. If not, please fix all the issues. Note: you do not need to verify that the VIN is valid.
2. Develop a test class, `VehicleTest`, to test the functionality of the given class using the JUnit 4 testing framework.
3. In addition to a VIN and a license plate, a vehicle typically has an owner. An owner has:
    - A first name, typically represented as a String.
    - A last name, also represented as a String.
    - A phone number, a String, consisting of ten characters[1].

    Create the new class, `Owner`, and its corresponding test class, `OwnerTest`.
4. Modify your `Vehicle` class so that it also contains information about a vehicle's `Owner`. Make sure to update all relevant parts of your code (classes and tests!).

---

[1] *Tip*: whenever you are given value constraints like this length requirement, it is expected that your code checks that the value meets the requirements! In the real world, you would probably also make sure a phone number only contains numbers but that is not explicitly mentioned in the constraint above, so you don't need to validate that. We haven't covered exceptions in Java yet so for now you can decide the best way to handle invalid data without exceptions—we'll only be checking that you have checked validity.

```java
/**
* Vehicle is a simple object that has a unique Vehicle  Identification Number
(VIN), and corresponding license plate.
*/
public class Vehicle {
    private Integer VIN;
    private String licensePlate;

    /**
    * Constructor that creates a new vehicle object with the specified VIN and
    * license plate.
    * @param VIN - VIN of the new Vehicle object.
    * @param licensePlate - license plate of the new Vehicle.
    */
    public Vehicle(Integer VIN, String licensePlate) {
        VIN = VIN;
        licensePlate = licensePlate;
    }

    /**
    * Returns the VIN of the Vehicle.
    * @return the VIN of the Vehicle.
    */
    public Integer getVIN() {
        return this.VIN;
    }

    /**
    * Returns the licensePlate of the Vehicle.
    * @return the licensePlate of the Vehicle.
    */
    public String getLicensePlate() {
        return 0;
    }

    /**
    * Sets the VIN of the Vehicle.
    */
    public void setVIN(Integer VIN) {
        VIN = 0;
    }

    /**
    * Sets the licensePlate of the Vehicle.
    */
    public void setLicensePlate(String licensePlate) {
        licensePlate = licensePlate;
    }
}
```

When you have completed this problem, run the Gradle tasks for style (PMD) and test coverage (JaCoCo) and view the reports. See the instructions on Canvas if you're not sure how to do this. Aim for 100% test coverage on instructions and branches (70% coverage is enough for full points) and try to fix all PMD issues if you can. You can assume that, from now on, you need to aim for at least 70% test coverage of all classes that you write in all assignments.

Problem 2

You are tasked with writing part of a program that a company can use to keep track of how much time their employees spend traveling. Your responsibility is the portion of the software that will store information about individual trips. For each trip made by an employee, you will need to store the trip's starting location (a String), the end location (a String), and the start and end time.

You will need to keep track of time by recording the hour, minute, and second. A valid time has:

- hour, between 0 and 23, inclusive.
- minutes, between 0 and 59, inclusive.
- seconds, between 0 and 59 inclusive.

1. Write the code to implement the trip tracking program. You will need to create classes `Trip` and `Time`. Please do not use any of Java's built-in date/time classes for this problem.
2. In class `Trip`, add a method `getDuration()` with the signature:

   ```
   public Time getDuration()
   ```

   The method should return the total time of the trip. The total time is computed as the difference between the trip's end time and the start time. Your method must return a valid `Time` object. When calculating duration, you can assume that trips start and end on the same day.
3. Write the corresponding test classes `TripTest` and `TimeTest` to test functional correctness of your classes `Trip` and `Time`. Be sure to unit test all methods!

Problem 3

You are helping to design software for a bank ATM. The ATM needs to accept deposits and provide withdrawals from customer accounts. Each account consists of:

- The account holder's name. An account holder is an individual with a first and last name.
- The current account balance as an amount. An amount consists of:
  - An integer value for the dollar amount, greater than or equal to 0.
  - An integer value for the cents amount, between 0 and 99 inclusive.

  Note that the dollar and cents amounts are separate integers rather than a single decimal value. The ATM needs to extract the amounts from handwritten checks and deposit slips so the dollar and cents amounts need to be validated separately.

1. Design and implement classes to store the data as described. The bank wants the system to be ***immutable***. This means that you should never modify values stored in an object's fields, instead return a new object with the appropriate values.
2. Write a method called `deposit` to handle a deposit event, in which money is added to the account balance. Your method should take one parameter, an `Amount`, and should return a new account object with an increased account balance.
3. Write a method called `withdraw` to handle a withdrawal event, in which money is withdrawn from the account. Your method should consume an amount and return a new account object with the balance decreased. You may assume the withdrawal amount will not be less than the account balance.
4. Write test classes to test the functional correctness of your program.