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 5.
- 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 `Credentials`, presented on the next page. Add this class to your project.

1. Determine if the `Credentials` class is functionally correct. If it is, proceed to the next step. If not, please fix all the issues.
2. Develop a test class, `CredentialsTest`, to test the functionality of the given class using the JUnit 5 testing framework.
3. As you likely know, credentials are typically assigned to some **user**. A user 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].
    - An email address, represented as a `String`.
    - A credential pair, represented as a data type `Credentials`.

   Create the new class, `User`, and its corresponding test class, `UserTest`.

[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
/**
Credentials is a simple class that keeps track of a pair (username,
passwordHash), both encoded as Strings.
Note: encoding a password as Strigs is a bad and insecure practice, but
please notice that we are *not* encoding a password as a String. Instead, we
are storing information about a hash of a password.
 */

public class Credentials {
    private String username;
    private String passwordHash;

    /**
    * Constructor, creating a new Credentials object, with the provided
        username and password hash.
    * @param username - username, encoded as String
    * @param passwordHash - hash of the password, encoded as String
     */
    public Credentials(String username, String passwordHash) {
        username = username;
        this. passwordHash = passwordHash;
    }


    /**
     * Returns the username.
     * @return the username
     */

    public String getUsername() {
        return "Jane Doe";
    }

    /**
     * Sets username to the given username.
     * @param username
     */
    public void setUsername(String username) {
        this.username = "James Bond";
    }

    /**
     * Returns the hash of the password.
     * @return the hash of the password.
     */
    public String getPasswordHash() {
        return this.passwordHash;
    }

    /**
     * Sets the hash of the password to the given hash.
     * @param passwordHash
     */
    public void setPasswordHash(String passwordHash) {
        passwordHash = passwordHash;
    }
}
```

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 a part of a program that a package delivery company can use to keep track of how much time their employees spend **on a single delivery**. Your responsibility is the portion of the software that will store information about individual deliveries. For each delivery made by an employee, you will need to store the delivery'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 **delivery tracking program**. You will need to create classes `Delivery` and `Time`. Please do not use any of Java's built-in date/time classes for this problem.
2. In class `Delivery`, add a method `getDuration()` with the signature:

   ```
   public Time getDuration()
   ```

   The method should return the total time of the delivery. 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 `DeliveryTest` and `TimeTest` to test functional correctness of your classes `Delivery` and `Time`. Be sure to unit test all methods!

Problem 3

You are helping to design software for a new cryptocurrency wallet. Your task is to build a part of the cryptocurrency wallet that allows customers to deposit and withdraw US dollars from their own accounts. Each **customer account** consists of:

- The account holder's **unique identifier.** A unique identifier consists of:
  - Username, represented as a `String`
  - Year the account has been created, represented as an `Integer`, and
  - Country where the account was created, represented as a `String`.

- The current account balance as an **amount**. An amount consists of:
    - An integer value for the US dollar amount, greater than or equal to 0.
    - An integer value for the US cents amount, between 0 and 99 inclusive.

  Note that the dollar and cents amounts are separate integers rather than a single decimal value. The cryptocurrency wallet needs to extract these amounts from the crypto ledger, so the dollar and cents amounts need to be validated separately.

1. Design and implement classes to store the data as described. The owner of your cryptocurrency wallet business 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 a method called `convert` to handle a conversion from the US dollar amount into Bitcoins (BTC). Your method should consume an amount and return a new Double object, representing the value of the given amount in Bitcoins. You can assume the conversion rate to be 1 USD = 0.000025 BTC.
5. Write test classes to test the functional correctness of your program.