Week 12 Research Assignment

Note: All answers are a synthesis of what I've learned from the class materials, unless a source is linked specifically.

Prompt:

What is the difference between TDD and BDD?

Note: The section on BDD is derived from the Wikipedia article on the subject, available at https://en.wikipedia.org/wiki/Behavior-driven_development. The section on TDD is solely my own understanding of the subject, as it was covered in the class materials of previous weeks.

Response:

TDD: Test Driven Development is a software development process that requires writing tests of a codebase before the codebase is actually fleshed out with functions, classes, etc. Once tests are written, they act as a scaffold for the actual logic and functionality of the codebase. Because the code has not been written yet, the tests will naturally result in failure; the next step is for the developer to write code that passes the test. This approach tends to simplify the development process by breaking it down into smaller, more manageable chunks. It also makes it easier to ensure the code still works as intended when features are added or refactoring occurs -- in these cases, the tests are already written, so the developer can simply run the tests again to ensure the code still passes.

BDD: Behavior Driven Development is less a separate approach from Test Driven Development and more a complementary approach. The emphasis of BDD is focus on high-level functionality of a codebase as told through a formal narrative (with specific conventions and structure) called a "User Story", and asks that the developer and the organization who hired them use certain keywords to describe the behavior of the codebase. These keywords are "Given", "When", and "Then". "Given" describes the initial state of the codebase, "When" describes the action that will be taken, and "Then" describes the expected outcome. For example, a User Story might read: "Given a user has a preference set for Dark Mode display for our website's UI, if they submit an updated preference for Light Mode display, this preference will then be updated in our database." This allows the developer to then utilize the TDD approach in a way that reflects what the organization wants the codebase to do, and allows the organization to communicate their needs to the developer in a way that is more easily understood.

Prompt:

What is one method that can be used to avoid putting plaintext database usernames and passwords into your code?

Response:

Note: all information in this section is my synthesis of information from the JCA documentation, available at https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html., as well as my own understanding of encapsulation in OOP and its relevance to security.

One method that can be used to avoid putting plaintext database usernames and passwords into your code is to use encryption. There are many robust encryption algorithms available today (such as SHA-256, AES, or RSA, to name a few), and all of them are varyingly appropriate for different use cases. There is a system for encryption and decryption of data that comes built in to Java called the **Java Cryptographic Architecture** or **JCA**. When writing an application that stores user credentials in a database, they'd likely have a layer of their application that uses functionality from the JCA (including `Providers`, a `Cipher`, and so on) to encrypt any credentials before they are stored in the database, and decrypt them when they are retrieved from the database. This would be wrapped in a class that would be used by the rest of the application to access the database, and the rest of the application would not need to know the details of how the encryption/decryption process works, nor would this class be accessible to users or other developers.