Getting Started With Mockito

Allows to mock objects...
Allows to create test doubles and add in behaviours and verifications.
Allows to have mock behaviour.
Keeps the code light, small and unity.
It is second to junit in terms of popularity.

Provides mock [alternate representation of objects] which can replace real objects and tests.

Types of mock:

Spy: We can intercept calls to the object or inspect things or pass directly to actual underlying objects.

We have to have these dependencies:

```
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>2.23.4</version>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-junit-jupiter</artifactId>
    <version>2.23.4</version>
    <scope>test</scope>
</dependency>
```

Mockito is a popular Java framework used for mocking objects in unit tests. It allows developers to create mock objects that simulate the behavior of real objects in a controlled manner, enabling more focused and isolated unit testing. Here are some key features and functionalities of Mockito:

- **Mocking Objects**: Mockito provides a simple and flexible API for creating mock objects. These mock objects have the same interface as the real objects they are mocking but allow developers to define specific behaviors and expectations for method calls.
- **Behavior Verification**: Mockito allows developers to verify interactions with mock objects, such as method calls, parameter values, and invocation counts. This helps ensure that the tested code behaves as expected and interacts correctly with its dependencies.
- **Stubbing Methods**: Developers can specify the return values of methods on mock objects using Mockito's stubbing feature. This

allows them to simulate different scenarios and control the behavior of the mocked dependencies during testing.

- **Argument Matchers**: Mockito provides a wide range of argument matchers to define flexible and expressive conditions for method invocations on mock objects. These matchers allow developers to specify matching criteria for method parameters and make tests more readable and maintainable.
- **Annotations**: Mockito supports annotations that simplify the creation and management of mock objects in test classes. Annotations like @Mock, @InjectMocks, and @Spy help streamline the setup of mock objects and reduce boilerplate code in test cases.
- **Integration with JUnit and Other Testing Frameworks**: Mockito seamlessly integrates with popular testing frameworks like JUnit, TestNG, and others, making it easy to incorporate mock objects into unit tests and test suites.

Overall, Mockito is widely used by Java developers to improve the effectiveness and efficiency of unit testing by enabling the creation of isolated and controlled testing environments through mock objects.