

Guía de Estudio: Mockito para Desarrolladores Junior

Una introducción al framework de pruebas unitarias



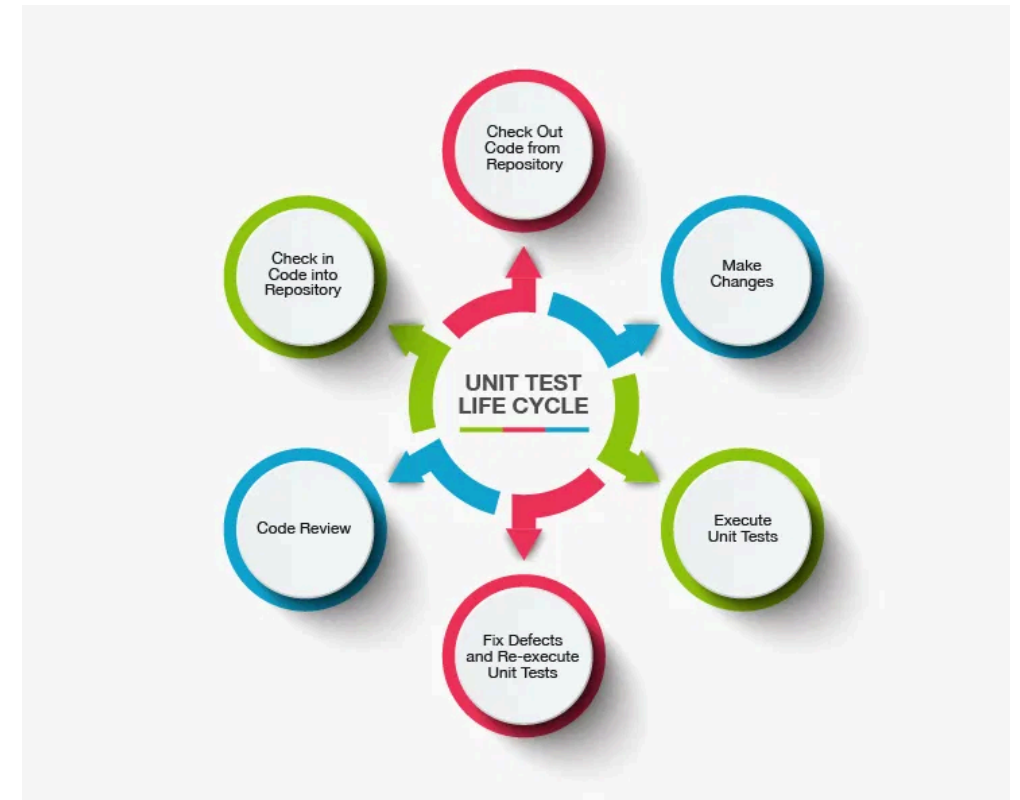
Introducción a las Pruebas Unitarias

¿Qué son? Las pruebas unitarias son una técnica de desarrollo que consiste en probar pequeñas partes de código (unidades) de forma aislada para verificar su correcto funcionamiento.

Importancia: Son fundamentales en el ciclo de desarrollo porque permiten detectar errores temprano, facilitan el refactoring y mejoran la calidad del código.

Beneficios para desarrolladores junior:

- Mejor comprensión del código
- Desarrollo de código más modular
- Confianza al realizar cambios
- Habilidad valorada en el mercado laboral



¿Qué es Mockito?

Mockito es un **framework de código abierto** para crear pruebas unitarias en Java que permite la creación de objetos simulados (mocks).

Propósito principal: Facilitar la creación de objetos simulados (mocks) para aislar el código que se está probando de sus dependencias externas.

Características:

- API limpia y simple
- Integración con JUnit
- Verificación de comportamiento
- Soporte para anotaciones

Origen: El nombre y el logo son un juego de palabras con los "mojitos", haciendo referencia a que las pruebas deberían ser "refrescantes".



¿Por qué usar Mockito?

Mockito ofrece numerosos beneficios para el desarrollo de software:

- ✓ **Aislamiento de dependencias:** Permite probar unidades de código sin depender de componentes externos.
- ✓ **API limpia y simple:** Sintaxis intuitiva que facilita la escritura de pruebas legibles.
- ✓ **Pruebas más rápidas:** Al no depender de bases de datos o servicios externos, las pruebas se ejecutan más rápido.
- ✓ **Mejora la calidad del código:** Fomenta el diseño modular y la separación de responsabilidades.
- ✓ **Integración con JUnit:** Funciona perfectamente con el framework de pruebas más popular en Java.



Conceptos Clave: Mocks, Stubs y Spies

Mocks

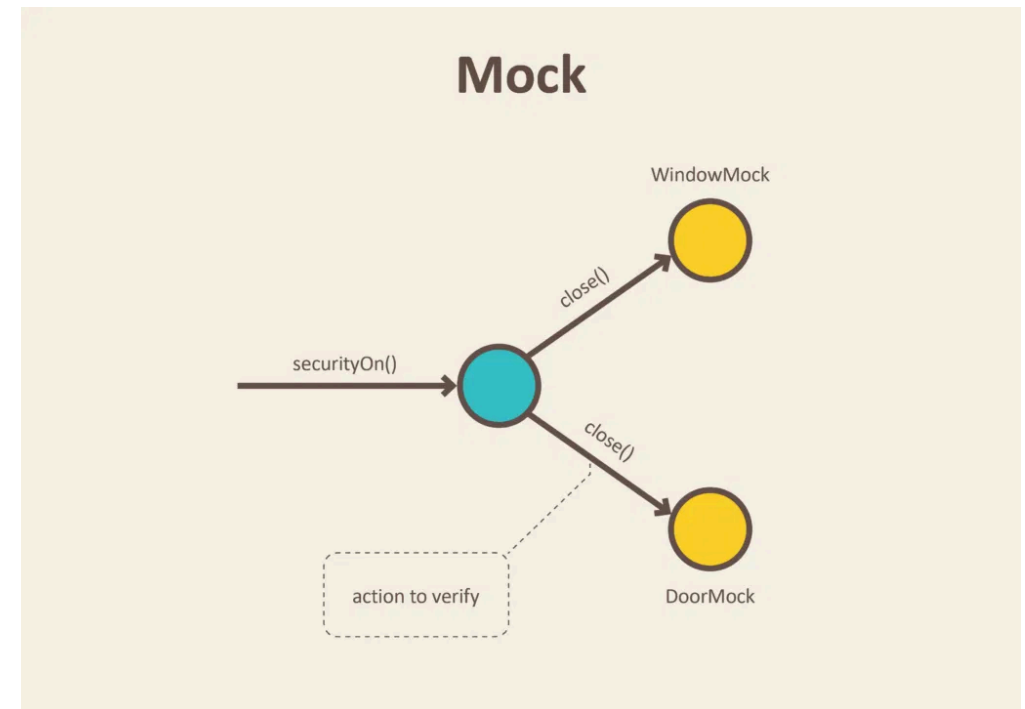
Objetos simulados que imitan el comportamiento de objetos reales. Se utilizan para controlar el comportamiento de las dependencias de la clase que se está probando.

Stubs

Mocks que devuelven valores predefinidos cuando se invocan ciertos métodos. Se utilizan cuando necesitamos que un método devuelva un valor específico para la prueba.

Spies

Objetos reales que se pueden espiar para verificar interacciones con ellos, pero que también ejecutan su comportamiento real. Útiles cuando queremos verificar que ciertos métodos fueron llamados sin modificar su comportamiento.



Anotaciones Comunes en Mockito

@Mock

Crea un objeto mock. Es útil cuando queremos usar el objeto simulado en varios lugares porque evitamos llamar al método `mock()` varias veces.

@InjectMocks

Injecta los objetos mock creados con `@Mock` en el objeto que se está probando. Útil para inyectar dependencias automáticamente.

@RunWith(MockitoJUnitRunner.class)

Habilita el uso de anotaciones de Mockito en las pruebas JUnit. Alternativa a `MockitoAnnotations.openMocks()`.

```
@RunWith(MockitoJUnitRunner.class)
public class UsuarioServicioTest {

    // Crea un mock del repositorio
    @Mock
    UsuarioRepositorio usuarioRepositorio;

    // Inyecta el mock en el servicio
    @InjectMocks
    UsuarioServicio servicio;

    @Test
    public void testObtenerUsuario() {
        // Configuración del mock
        when(usuarioRepositorio.obtenerUsuario(0))
            .thenReturn(new UsuarioDto(0, "Juan"));

        // Ejecución del método a probar
        UsuarioDto resultado = servicio.obtenerUsuario(0);

        // Verificación
        assertEquals("Juan", resultado.getNombre());
    }
}
```

Métodos Importantes de Mockito

Mockito.mock()

Crea un objeto mock de una clase o interfaz.

```
List<String> mockList = Mockito.mock(List.class);
```

when().thenReturn()

Define el comportamiento de un método mock.

```
when(mockList.get(0)).thenReturn("Hola Mundo");
```

when().thenThrow()

Define que un método mock lance una excepción.

```
when(mockList.get(1)).thenThrow(new RuntimeException());
```

verify()

Verifica llamadas a métodos en un mock.

```
verify(mockList).add("elemento");
```

- ① `Target mockTarget = mock(Target.class);`
Creates an instance of Target which method invocations are redirected to an interceptor
- ② `when(mockTarget.doIt("Read")).thenReturn("Mocked!");`
The last invocationDetails which is "Target", "doIt" and "Read" is returned
Interceptor adds "Target", "doIt" and "Read" to invocationDetails list
"Mocked!" is set as the return value of the invocation "Target", "doIt" and "Read"
- ③ `mockTarget.doIt("Read").equals("Mocked!"); // true`
Returns the recorded "Mocked!" instead of "Done: Read"

Ejemplo Práctico: Mockito en Acción

```
@RunWith(MockitoJUnitRunner.class)
public class CarritoServiceTest {

    // Creamos un mock del repositorio
    @Mock
    ProductoRepository productoRepository;

    // Inyectamos el mock en el servicio
    @InjectMocks
    CarritoService carritoService;

    @Test
    public void testCalcularTotal() {
        // Configuramos el comportamiento del mock
        when(productoRepository.obtenerPrecio("P001")).thenReturn(100.0);
        when(productoRepository.obtenerPrecio("P002")).thenReturn(200.0);

        // Ejecutamos el método a probar
        List<String> productos = Arrays.asList("P001", "P002");
        double total = carritoService.calcularTotal(productos);

        // Verificamos el resultado
        assertEquals(300.0, total, 0.01);

        // Verificamos que se llamó al repositorio
        verify(productoRepository).obtenerPrecio("P001");
        verify(productoRepository).obtenerPrecio("P002");
    }
}
```


Beneficios para Desarrolladores Junior

Mockito ofrece ventajas específicas para desarrolladores que están comenzando:



Facilita el aprendizaje: Permite entender mejor los conceptos de pruebas unitarias con una API intuitiva.



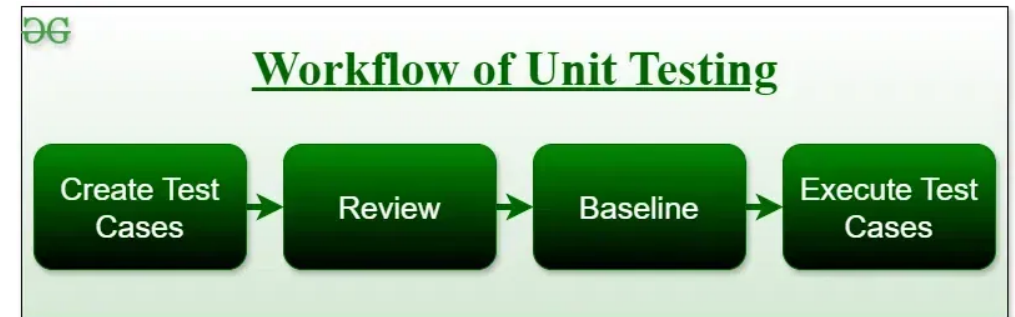
Mejora la calidad del código: Fomenta buenas prácticas de diseño y arquitectura desde el inicio.



Aumenta la empleabilidad: Las habilidades en pruebas unitarias son altamente valoradas en el mercado laboral.







Prepara para el trabajo en equipo: Facilita la integración en equipos que siguen metodologías ágiles y TDD.



Recursos Adicionales y Próximos Pasos

Recursos para Seguir Aprendiendo

-  **Documentación oficial:** site.mockito.org
-  **Tutoriales en video:** Canales como "Programando en Java" y "Amigoscode"
-  **Repositorios de ejemplos:** GitHub tiene numerosos proyectos con ejemplos prácticos
-  **Comunidades:** Stack Overflow, Reddit r/java, Discord de Java

Próximos Pasos

- ✓ **Practicar con proyectos pequeños:** Crear pruebas para aplicaciones simples
- ✓ **Explorar funcionalidades avanzadas:** ArgumentCaptor, doAnswer(), spy()
- ✓ **Integrar con otros frameworks:** Spring Boot, TestNG
- ✓ **Contribuir a proyectos open source:** Aplicar conocimientos en proyectos reales