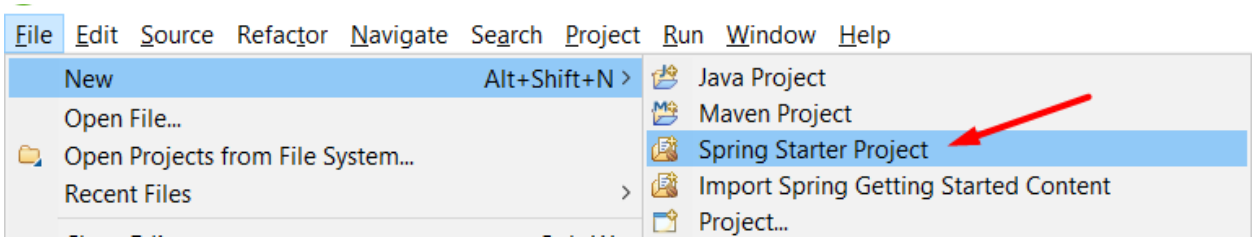
PROYECTO “mockito-demo”

Paso 1: Crear el Proyecto en STS

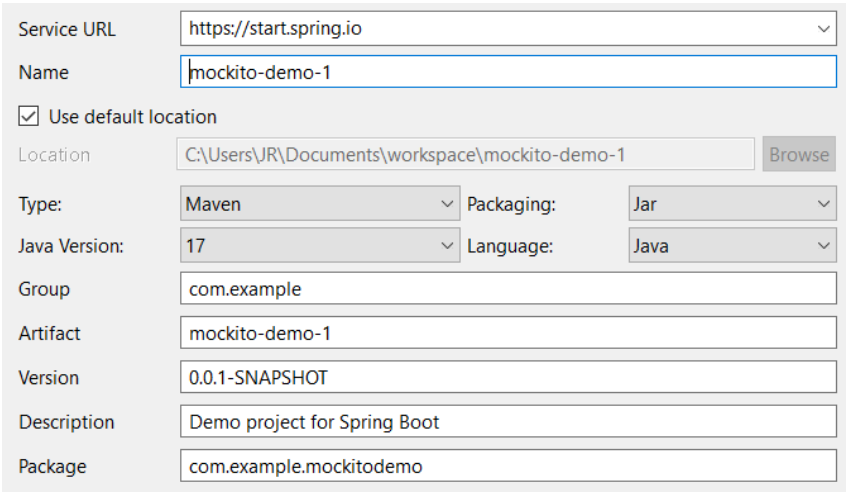
Abrir Spring Tool Suite (STS).

Ir a **File → New → Spring Starter Project**



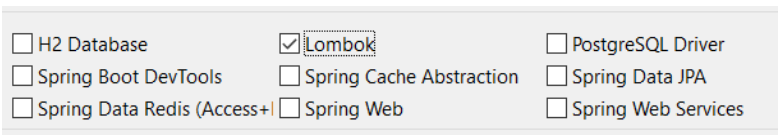
Completar los datos del proyecto:

- **Name:** mockito-demo
- **Type:** Maven
- **Java Version:** 17
- **Group:** com.example
- **Artifact:** mockito-demo
- **Package:** com.example.mockitodemo
- **Packaging:** Jar



Seleccionar las dependencias necesarias:

- **Spring Boot Starter Test** (Para usar JUnit y Mockito).
- **Lombok** (Para reducir código innecesario).



Hacer clic en **Finish** para generar el proyecto.

Paso 2: Crear una Interfaz UsuarioRepository

Creamos la interfaz UsuarioRepository

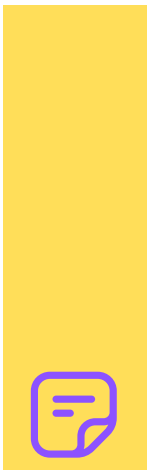
```java
package com.example.mockitodemo.repository;

import com.example.mockitodemo.model.Usuario;
import java.util.Optional;

public interface UsuarioRepository {
    Optional<Usuario> findById(Long id);
    Usuario save(Usuario usuario);
}
```

Explicación:

- **UsuarioRepository**: Es una interfaz que define las operaciones que un repositorio de usuarios debe implementar.
- **Métodos**:
  1. **Optional<Usuario> findById(Long id)**:
     - Busca un usuario por su ID.
     - Devuelve un Optional<Usuario>, lo que significa que el resultado puede ser un objeto Usuario o estar vacío si no se encuentra ningún usuario con ese ID.
  2. **Usuario save(Usuario usuario)**:
     - Guarda o actualiza un usuario en el repositorio.
     - Recibe un objeto Usuario como parámetro y devuelve el usuario guardado (que puede ser el mismo objeto o uno modificado).

Paso 3: Crear la Clase Usuario

Creamos la clase Usuario



```java
package com.example.mockitodemo.model;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
public class Usuario {
    private Long id;
    private String nombre;
    private String email;
}
```

Explicación:

- Usuario representa a un usuario en el sistema.
- Usamos **Lombok** (@Getter, @Setter, etc.) para simplificar el código.

Paso 4: Crear la Clase UsuarioService

Creamos la clase service UsuarioService

```java
package com.example.mockitodemo.service;

import com.example.mockitodemo.model.Usuario;
import com.example.mockitodemo.repository.UsuarioRepository;
import org.springframework.stereotype.Service;

@Service
public class UsuarioService {
    private final UsuarioRepository usuarioRepository;

    public UsuarioService(UsuarioRepository usuarioRepository) {
        this.usuarioRepository = usuarioRepository;
    }

    public Usuario obtenerUsuarioPorId(Long id) {
        return usuarioRepository.findById(id).orElse(null);
    }

    public Usuario registrarUsuario(Usuario usuario) {
        return usuarioRepository.save(usuario);
    }
}
```

Explicación:

- Tiene dos métodos:
  a. **obtenerUsuarioPorId(Long id):** Busca un usuario en el repositorio.
  b. **registrarUsuario(Usuario usuario):** Guarda un usuario en el repositorio.

Paso 5: Crear las Pruebas Unitarias con Mockito

Creamos las pruebas unitarias en la ubicación: src/test/java/com/example/mockitodemo/service

Creamos la prueba UsuarioServiceTest

```java
package com.example.mockitodemo.service;

import com.example.mockitodemo.model.Usuario;
import com.example.mockitodemo.repository.UsuarioRepository;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;
import java.util.Optional;
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

@ExtendWith(MockitoExtension.class)
class UsuarioServiceTest {

    @Mock
    private UsuarioRepository usuarioRepository;

    @InjectMocks
    private UsuarioService usuarioService;

    private Usuario usuario;

    @BeforeEach
    void setUp() {
        usuario = new Usuario(1L, "Juan Pérez", "juan@example.com");
    }
```

Explicación:

- **@Mock**: Crea un objeto simulado (mock) de UsuarioRepository. Este objeto no es una instancia real del repositorio, sino una simulación que permite controlar su comportamiento en las pruebas.
- **@InjectMocks**: Crea una instancia real de UsuarioService e inyecta los mocks (en este caso, usuarioRepository) en ella. Esto permite probar el servicio con dependencias simuladas.
- **@BeforeEach**: Este método se ejecuta antes de cada prueba. Aquí se inicializa un objeto Usuario que se utilizará en las pruebas.

```java
@Test
void testObtenerUsuarioPorId() {
    // GIVEN: Simulamos el comportamiento del repositorio
    when(usuarioRepository.findById(1L)).thenReturn(Optional.of(usuario));

    // WHEN: Llamamos al servicio
    Usuario resultado = usuarioService.obtenerUsuarioPorId(1L);

    // THEN: Verificamos el resultado
    assertNotNull(resultado);
    assertEquals("Juan Pérez", resultado.getNombre());
    assertEquals("juan@example.com", resultado.getEmail());

    // Verificamos que el método se llamó exactamente una vez
    verify(usuarioRepository, times(1)).findById(1L);
}
```

Explicación:

- **when(...).thenReturn(...)**: Simula el comportamiento del método findById del repositorio. Cuando se llame con el argumento 1L, devolverá un Optional que contiene el objeto usuario.
- **assertNotNull(resultado)**: Verifica que el resultado no sea null.
- **assertEquals(...)**: Compara los valores esperados con los obtenidos (nombre y email).
- **verify(...)**: Verifica que el método findById del repositorio se haya llamado exactamente una vez con el argumento 1L.

```java
@Test
void testRegistrarUsuario() {
    // GIVEN: Simulamos que el repositorio guarda correctamente el usuario
    when(usuarioRepository.save(usuario)).thenReturn(usuario);

    // WHEN: Llamamos al servicio
    Usuario resultado = usuarioService.registrarUsuario(usuario);

    // THEN: Verificamos el resultado
    assertNotNull(resultado);
    assertEquals("Juan Pérez", resultado.getNombre());
    assertEquals("juan@example.com", resultado.getEmail());

    // Verificamos que el método se llamó exactamente una vez
    verify(usuarioRepository, times(1)).save(usuario);
}
```

Explicación:

- @Mock: Crea un **objeto simulado** de UsuarioRepository.
- @InjectMocks: Inyecta el **Mock** en UsuarioService.
- when(...).thenReturn(...): Simula respuestas del repositorio.
- verify(..., times(1)): Verifica que el método se llamó exactamente una vez.

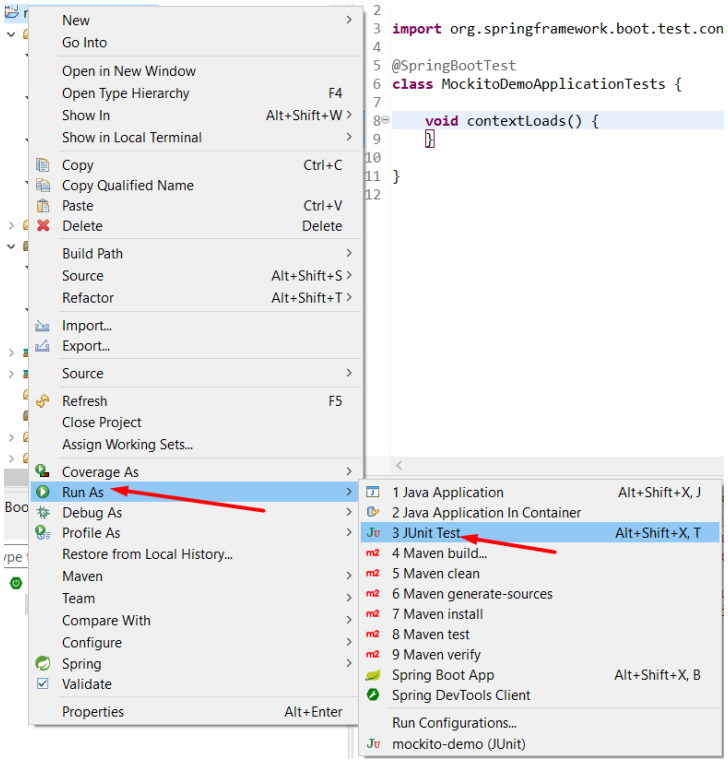
Paso 7: Quitamos el @test de la clase principal

En nuestro ejemplo no realizamos una implementación concreta en el repository, y para evitar que nos de error al ejecutar las pruebas unitarias del proyecto completo **eliminamos el @Test**

```java
package com.example.mockitodemo;

import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class MockitoDemoApplicationTests {

    void contextLoads() {
    }

}
```

Paso 8: Ejecutar las Pruebas

Ir a UsuarioServiceTest.java y hacer clic derecho → Run As → JUnit Test.



Deberías ver que todas las pruebas pasan en verde .