Escribir un stub con Mockito

Agrega una prueba que confirme el comportamiento esperado. Capturaremos esto en una aserción:

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.junit.jupiter.MockitoExtension;
import static org.assertj.core.api.Assertions.assertThat;
@ExtendWith(MockitoExtension.class)
public class UserGreetingTest {
        @Test
        void formatsGreetingWithName() {
        String actual = “”;
        assertThat(actual)
        .isEqualTo("Hola y bienvenido, Kapumota");
        }
}
```

Este es el uso estándar de los frameworks JUnit y AssertJ como hemos visto antes.

Problema: ¿Qué sucede si ejecutas la prueba ahora?.

La prueba falla porque la cadena actual es distinta al mensaje.

```
Agrega un esqueleto de ID de usuario de clase:
```

```
public class UserId {
        public UserId(String id) {
        }
}
```

Problema: ¿Qué sucede si ejecutas la prueba ahora?. Explica la salida

Se lanza una excepción (UnsupportedOperationException) al llamar al método formatGreeting porque dicho método aún no está implementado, solo lanza una excepción cuando se llama.

Otra decisión de diseño a capturar es que la clase UserGreeting dependerá de una interfaz UserProfiles. Necesitamos crear un campo, crear el esqueleto de la interfaz e inyectar el campo en un nuevo constructor para el SUT

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.junit.jupiter.MockitoExtension;
import static org.assertj.core.api.Assertions.assertThat;
@ExtendWith(MockitoExtension.class)
public class UserGreetingTest {
        private static final UserId USER_ID
            = new UserId("1234");
             private UserProfiles profiles;
        @Test
        void formatsGreetingWithName() {
```

```
        var greeting
                        = new UserGreeting(profiles);
        String actual =
                          greeting.formatGreeting(USER_ID);
        assertThat(actual)
                          .isEqualTo(“Hola y bienvenido Kapumota”);
        }
}
```

Continuamos agregando el código mínimo para que la prueba se compile.

Problema: ¿Qué sucede si ejecutas la prueba ahora?. Explica la salida

Si se ejecuta nuevamente el código se lanza la misma excepción (UnsupportedOperationException) al llamar al método formatGreeting, ya que el cuerpo del método solo lanza una excepción cuando se llama.

Agrega comportamiento al método formatGreeting():

```
public class UserGreeting {
        private final UserProfiles profiles;
        public UserGreeting(UserProfiles profiles) {
        this.profiles = profiles;
        }
 public String formatGreeting(UserId id) {
        return String.format("Hola y  Bienvenidos, %s",
                profiles.fetchNicknameFor(id));
        }
}
```

Agrega fetchNicknameFor() a la interfaz UserProfiles.

Ejecute la prueba. ¿Qué sucede?.

Se produce una excepción (NullPointerException) al intentar llamar el método fetchNicknameFor en la variable profiles, ya que esta variable no se ha inicializado y por lo tanto tiene un valor null.

Agrega la anotación @Mock al campo profiles:

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;
import static org.assertj.core.api.Assertions.assertThat;
@ExtendWith(MockitoExtension.class)
public class UserGreetingTest {
     private static final UserId USER_ID = new
        UserId("1234");

        @Mock
        private UserProfiles profiles;
```

```
         @Test
        void formatsGreetingWithName() {
            var greeting = new UserGreeting(profiles);

               String actual =
                         greeting.formatGreeting(USER_ID);

               assertThat(actual)
                .isEqualTo("Hola y bienvenido  Kapumota");
        }
}
```

Problema: ¿Qué sucede si ejecutas la prueba ahora?. Explica la salida.

Ya no se lanza la excepción NullPointerException sino que se ejecuta la prueba y falla, mostrándose la diferencia entre el valor esperado y el valor obtenido en la prueba:

```
expected: "Hola y bienvenido  Kapumota"

 but was: "Hola y  Bienvenidos, null"
```

Ambos valores se diferencian en la palabra final, mostrándose null en vez de Kapumota en la salida real obtenida en la prueba. No se lanzó la excepción NullPointerException porque “profiles” fué inicializado mediante la anotación @Mock con un doble de prueba; sin embargo no se ha definido el comportamiento de este objeto al llamar al método fetchNicknameFor por lo que devuelve null.

Configura @Mock para devolver los datos del stub correctos para la prueba.

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;

import org.mockito.Mockito;
import org.mockito.junit.jupiter.MockitoExtension;
import static org.assertj.core.api.Assertions.assertThat;
import static org.mockito.Mockito.*;
@ExtendWith(MockitoExtension.class)
public class UserGreetingTest {

     private static final UserId USER_ID = new
     UserId("1234");

      @Mock
       private UserProfiles profiles;

       @Test
          void formatsGreetingWithName() {
              when(profiles.fetchNicknameFor(USER_ID))
                        .thenReturn("Kapumota");
             var greeting = new UserGreeting(profiles);
             String actual =
```

```
                        greeting.formatGreeting(USER_ID);
        assertThat(actual)
                .isEqualTo("Hola y bienvenido, Kapumota");
        }
}
```

¿Qué sucede si vuelves a ejecutar la prueba?.

Esta vez la prueba es exitosa debido a que se ha definido el comportamiento del objeto doble de prueba (stub) profiles cuando se llama el método fetchNicknameFor con argumento USER_ID, retornando el valor necesario para que pase la prueba.