Mocks 101: El Arte del Testing

Aitor Santana

Aitor Reviriego

Preparación del entorno

https://github.com/Sstark97/mock-101

Preparación del entorno

- Un miembro de la pareja hacer un Fork del repositorio
- El otro miembro se clona ese Fork
- El miembro que haga el Fork hace colaborador al otro miembro
- Empezamos!

- Si los dos miembros tiene un IDE de Jetbrains se puede usar “Code With Me”

10 min

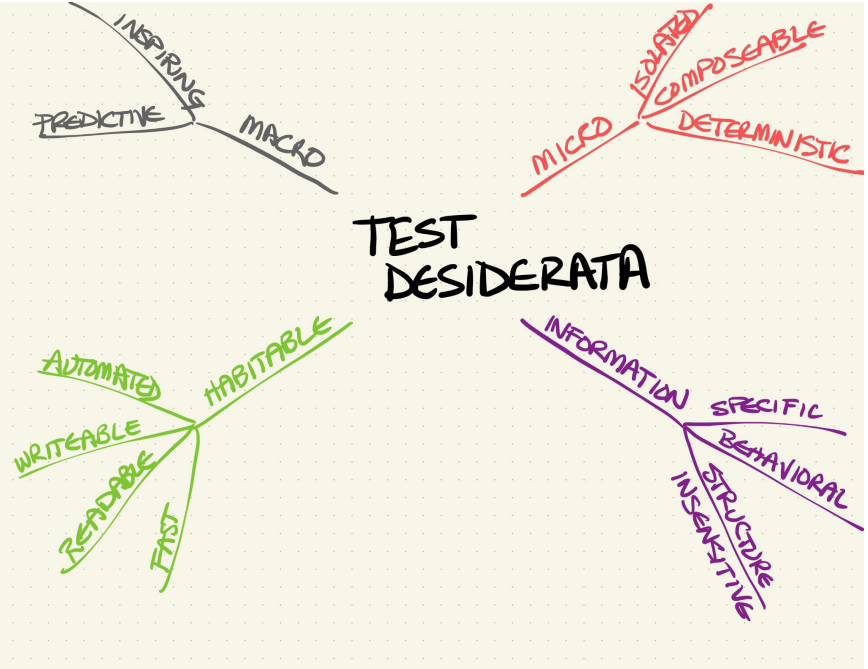
Guia del Taller

- Primeros conceptos
  - ¿Qué es un buen test? Test desiderata
  - Test Solitario y Test Social
  - ¿Qué son los dobles y para que se usan?
- Step 1: Random Number Kata
  - Dummies y Stubs
- Step 2: Print Date
  - Spies, Mock Estricto y Fake Object
- Step 3: Usando una librería de terceros

⚠️Importante⚠️

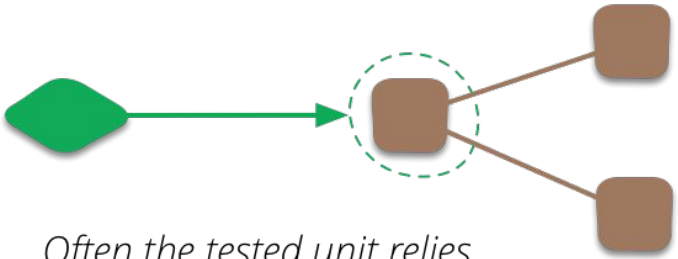
- Cumplir las restricciones de cada ejercicio

¿Qué es un buen test?



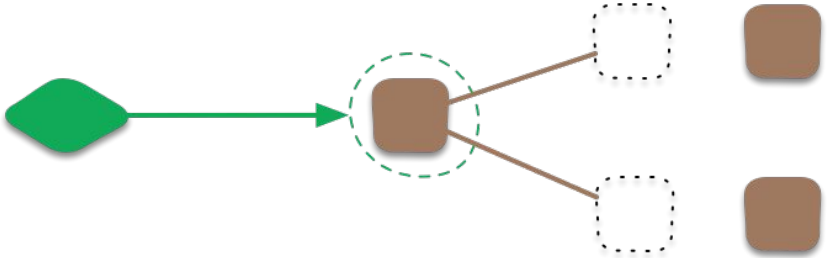
Test Sociables y Solitarios

Sociable Tests



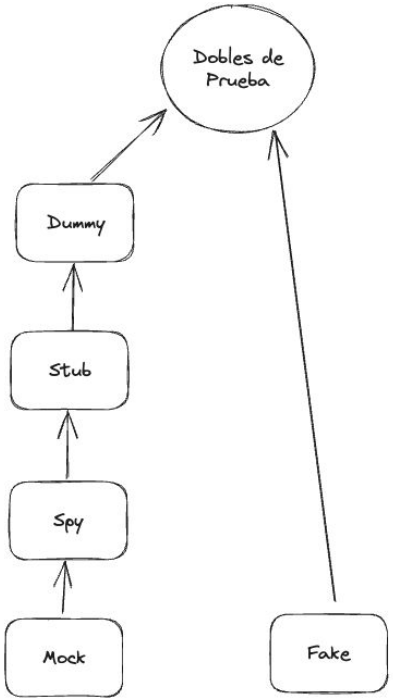
Often the tested unit relies on other units to fulfill its behavior

Solitary Tests



Some unit testers prefer to isolate the tested unit

¿Qué son los dobles de test y para que se usan?



Introducción al Ejemplo (Login Dialog)

```
interface Authenticator {
    boolean authenticate(String username, String password)
}

public class LoginDialog {
  private final Authenticator authenticator;
  private boolean isOpen = false;

  public LoginDialog(Authenticator authenticator) {
    this.authenticator = authenticator;
  }

  public boolean submit(String username, String password) {
    if(isOpen) {
      close();
      return authenticator.authenticate(username, password);
    }

    return false;
  }

  // More code...
}
```

Dummies y Stubs

```
public class AuthenticatorDummy implements Authenticator {
  @Override
  public boolean authenticate(String username, String password) {
    return false;
  }
}

@Test
void when_closed_login_is_canceled() {
  Authenticator authenticator = new AuthenticatorDummy();
  LoginDialog dialog = new LoginDialog(authenticator);

  dialog.show();
  dialog.close();

  assertFalse(dialog.isOpen());
}
```

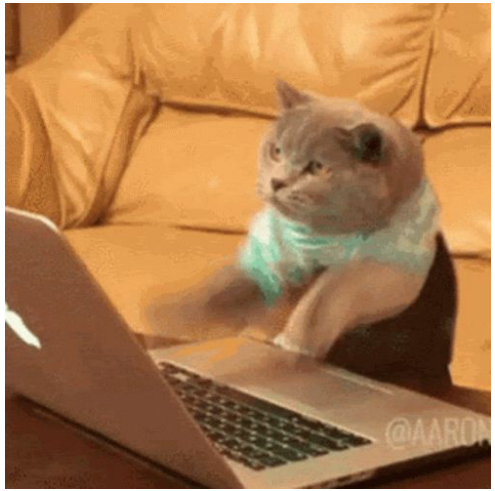
Dummies y Stubs

```java
public class AuthenticatorStub implements Authenticator {
  private final boolean allowLogin;

  public AuthenticatorStub(boolean allowLogin) {
    this.allowLogin = allowLogin;
  }

  @Override
  public boolean authenticate(String username, String password) {
    return allowLogin;
  }
}

@Test
public void when_authorizer_deny_login_work_well() {
  Authenticator authenticator = new AuthenticatorStub(false);
  LoginDialog dialog = new LoginDialog(authenticator);

  dialog.show();
  boolean success = dialog.submit("username", "password");

  assertFalse(success);
}
```

Random number kata

Requerimientos

- El usuario empieza a jugar, el juego genera un número aleatorio que no debe cambiar hasta que termine la partida.
- Si el usuario acierta el número, el jugador gana.
- Si el usuario no adivina el número, el sistema tendrá que notificar al usuario si el número es mayor o menor.
- Si el usuario no acierta el número en tres intentos perderá.



A trabajar ! 👷

Random number kata

Requerimientos

- El usuario empieza a jugar, el juego genera un número aleatorio que no debe cambiar hasta que termine la partida.
- Si el usuario acierta el número, el jugador gana.
- Si el usuario no adivina el número, el sistema tendrá que notificar al usuario si el número es mayor o menor.
- Si el usuario no acierta el número en tres intentos perderá.

Spies

```
public class AuthenticatorSpy implements Authenticator {

  private final boolean allowLogin;
  private int calls = 0;
  private String registeredUserName;
  private String registeredPassword;

  public AuthenticatorSpy(boolean allowLogin) {
    this.allowLogin = allowLogin;
  }

  @Override
  public boolean authenticate(String username, String password) {
    calls ++;
    registeredUserName = username;
    registeredPassword = password;
    return allowLogin;
  }

  public int calls() {
    return calls;
  }

  public String registeredUserName() {
    return registeredUserName;
  }

  public String registeredPassword() {
    return registeredPassword;
  }
}
```

```
@Test
void loging_dialog_correctly_invokes_authenticator()  {
  AuthenticatorSpy authenticatorSpy = new AuthenticatorSpy(true);
  LoginDialog dialog = new LoginDialog(authenticatorSpy);

  dialog.show();
  boolean success = dialog.submit("user", "pw");

  assertTrue(success);
  assertEquals(1, authenticatorSpy.calls());
  assertEquals("user", authenticatorSpy.registeredUserName());
  assertEquals("pw", authenticatorSpy.registeredPassword());
}
```

Mock Estricto

```java
public class AuthenticatorStrictMock implements Authenticator {

  private boolean authenticateCalled = false;
  private final String expectedUsername;
  private final String expectedPassword;
  private final boolean authenticationResult;

  public AuthenticatorStrictMock(
    String expectedUsername,
    String expectedPassword,
    boolean authenticationResult
  ) {
    this.expectedUsername = expectedUsername;
    this.expectedPassword = expectedPassword;
    this.authenticationResult = authenticationResult;
  }

  @Override
  public boolean authenticate(String username, String password) {
    if (!expectedUsername.equals(username) || !expectedPassword.equals(password)) {
      throw new AssertionError("Authenticator was called with unexpected arguments");
    }
    if (authenticateCalled) {
      throw new AssertionError("Authenticator authenticate method called more than once");
    }
    authenticateCalled = true;
    return authenticationResult;
  }

  public void verify() {
    if (!authenticateCalled) {
      throw new AssertionError("Expected authenticate method was not called");
    }
  }
}
```

```java
@Test
void login_dialog_correctly_invokes_authenticator() {
  AuthenticatorStrictMock authenticatorMock = new AuthenticatorStrictMock("user", "password", true);
  LoginDialog dialog = new LoginDialog(authenticatorMock);

  dialog.show();
  dialog.submit("user", "password");

  authenticatorMock.verify();
}
```

Fake Object

```
public class AuthenticatorFake implements Authenticator {

  @Override
  public boolean authenticate(String username, String password) {
    return username.length() == 5 & password.length() == 8;
  }
}

@Test
void bad_password_attempt_login_fail() {
  AuthenticatorFake authenticatorFake = new AuthenticatorFake();
  LoginDialog dialog = new LoginDialog(authenticatorFake);

  dialog.show();
  boolean success = dialog.submit("user", "pw");

  assertFalse(success);
}
```

Print Date kata

Requerimientos

- Probar un método que imprime la fecha actual.
- Ser capaz de probar la función printCurrentDate sin cambiar la firma del método.

A trabajar ! 👷

35 min

Print Date kata

Requerimientos

- Probar un método que imprime la fecha actual.
- Ser capaz de probar la función printCurrentDate sin cambiar la firma del método.

Ahora nos toca a nosotros ! 🤸

Muchas gracias!!

Aitor Santana

Aitor Reviriego

LeanMind
EFFICIENT PEOPLE