[**Pruebas Unitarias**](#_jzf13yjl6ie0) **1**

[Procedimiento implementación pruebas unitarias en ABAP.](#_hisfhoeutjn) 1

[Sintaxis para aplicación de pruebas unitarias](#_ys75etfmtgyd) 1

[Herramientas del Workbench para pruebas unitarias](#_8un35yrlue9l) 2

[Aplicación de pruebas unitarias utilizando TDD](#_uq9t93hpv496) 2

[**Pruebas Unitarias en Objetos de Desarrollo**](#_h7dwvhdhc0nj) **2**

[Pruebas para clases globales](#_9iwheyi1y1sd) 2

[Pruebas para clases locales](#_dost073a7opq) 4

[Pruebas para módulos de función](#_m9ntt2kj8lxm) 4

[**Manejo de dependencias**](#_6pfq7rqx7nf4) **5**

[**Nombramiento de clases para separar dependencias**](#_81q9q22vz4av) **5**

[Dependencia para acceso a base de datos](#_xqqsxpox28g8) 5

[Dependencia para conexión a sistema remoto](#_qskz5el3r7x0) 7

[Dependencia para acceso a parámetros](#_uekik1lfojj6) 7

[Inyección de dependencias con constructor](#_j1m96woef2qy) 8

[Inyección de dependencias con método setter](#_tvuijwx2y4s2) 11

[Inyección de dependencias con atributos privados (Friendship)](#_asilfr2agg46) 11

# Pruebas Unitarias

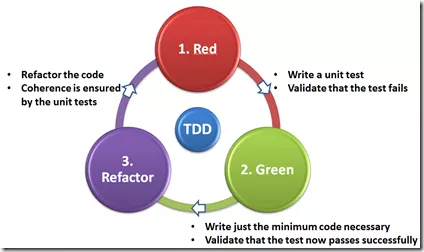
## **Procedimiento implementación pruebas unitarias en ABAP.**

### **Sintaxis para aplicación de pruebas unitarias**

### **Herramientas del Workbench para pruebas unitarias**

## **Aplicación de pruebas unitarias utilizando TDD**

La realización de desarrollo dirigido por pruebas unitarias se basa en un flujo simple de tres pasos:



*Ciclo Test Driven Development*

Antes de aplicar el ciclo de TDD es necesario tener la lista de funcionalidades con que debe cumplir el desarrollo a realizar. Esta lista es conocida como la lista **IT SHOULD**.

Una lista IT SHOULD se vería de la siguiente manera:

“...*Se requiere de una funcionalidad que permita (IT SHOULD):*

* *Consultar datos de cliente por número de identificación fiscal*
* *Se debe poder controlar por parámetro en ZTCX0001 qué tipos de identificación fiscal se permite consultar*
* *Enviar datos de cliente por interfaz hacia otro sistema*
* *Se debe poder controlar por parámetro en ZTCX0001 qué tipos de identificación fiscal se permite enviar por interfaz hacia otro sistema...* “

Teniendo clara la lista de funcionalidades con las que debe cumplir el desarrollo se aplican los pasos del ciclo TDD.

Paso 1:

Se inicia por escribir la prueba definiendo:

* Cuál es el estado inicial de datos previo al llamado del método a probar
* Cuál es el método que resuelve la funcionalidad requerida “IT SHOULD”, sus parámetros de entrada y sus parámetros de salida.
* Y cuál es la verificación que determinará que ejecuta correctamente.

En este paso aún no se tiene el código productivo escrito, por lo tanto el fallo inicial de la prueba es que no compila.

Paso 2:

Se crea la clase y método que se quiere probar en la prueba y se escribe el código necesario hasta que la prueba pase en verde.

Paso 3:

Es la oportunidad de refactorizar, organizar el código para que sea, más legible, de mejor mantenimiento, más funcional, asegurándose que las pruebas escritas no fallen.

# **Pruebas Unitarias en Objetos de Desarrollo**

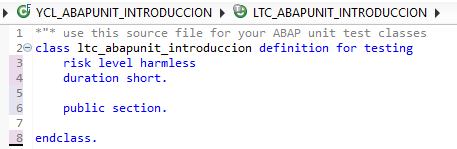
## **Pruebas para clases globales**

Teniendo la clase global YCL\_ABAPUNIT\_INTRO se procede a crear la clase de pruebas local en el include para clases de prueba.

**Sintaxis para nombre de clase**

LTC\_<Nombre de clase a probar>

**Ejemplo:** LTC\_ABAPUNIT\_INTRO



**Definition for Testing:** Determina que esta clase es una clase de pruebas y no es código productivo. De esta manera el framework ABAP no realiza ejecuciones de este código en un ambiente productivo.

**Risk Level Harmless:** determina que la clase de prueba no realiza modificaciones de base de datos ni operaciones riesgosas.

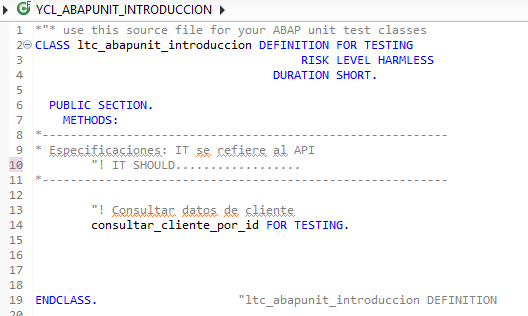
**Duration Short:** Determina que la duración de cada método de prueba es de unos milisegundos

**Métodos de prueba**

Se procede a definir los métodos de prueba según las funcionalidades que tendrá el requerimiento y que se desarrollaran en la clase.

***Definición de método***

Los métodos de prueba identifican las funcionalidades que debe cumplir el desarrollo.



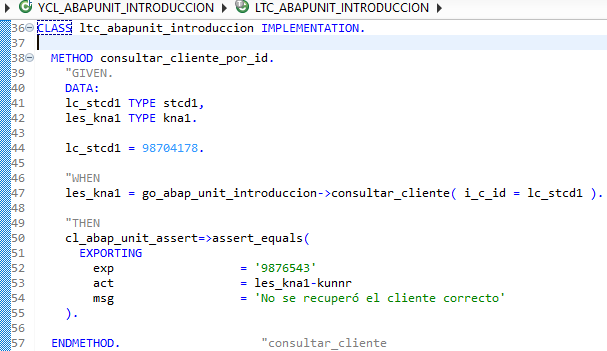
***Implementación de método***

La implementación del método de prueba debe estar segmentado en tres secciones; GIVEN, WHEN, THEN.

**Sección GIVEN:** Define los pre-requisitos del escenario de prueba a ejecutar.

**Sección WHEN:** Se realiza el llamado al método con el código productivo que se está probando.

**Sección THEN:** Se realizan las verificaciones necesarias para determinar que el método ejecuta correctamente. Se codifican los llamados a los métodos assert de la clase CL\_ABAP\_UNIT\_ASSERT.



## **Pruebas para clases locales**

Teniendo el reporte ZMME0010

El nombrado de los include sería:

ZMME0010V Variables

ZMME0010F Subrutinas

ZMME0010I PAI

ZMME0010O PBO

ZMME0010C Definición de clase / Impl

ZMME0010T Clases de Prueba Unit.

ZMME0010M Mock – Test Double

## **Pruebas para módulos de función**

# **Manejo de dependencias**

# **Nombramiento de clases para separar dependencias**

Para manejar las dependencias se define generar las clases según su funcionalidad como se muestra en las siguientes secciones del documento.

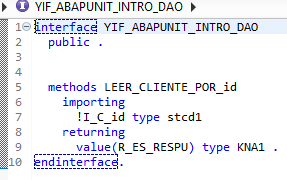
Con el fin de poder sustituir las dependencias con otras clases, en tiempo de ejecución de las pruebas, se deben utilizar interfaces que definen la funcionalidad que proveen e inyectar dependencias utilizando las referencias a las interfaces en lugar de las referencias a las clases.

## Inyección de dependencias con constructor

Se realiza definiendo un parámetro de entrada en el constructor de la clase principal con tipo de referencia a una interface, que se puede utilizar para definir diferentes clases implementadoras. De esta manera es posible pasar por parámetro en tiempo de ejecución una instancia de diferente clase implementadora.

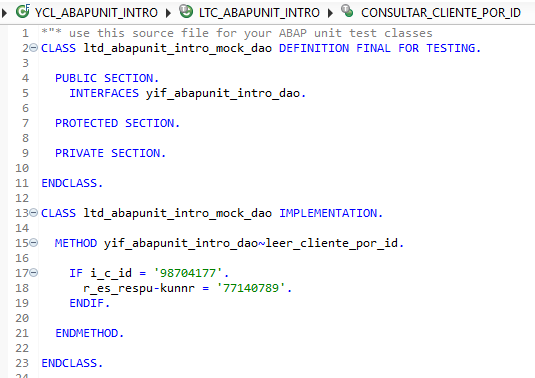
**Inyección de dependencia por constructor para inyectar un mock**

Se define la interfaz con la definición del método de acceso a base de datos



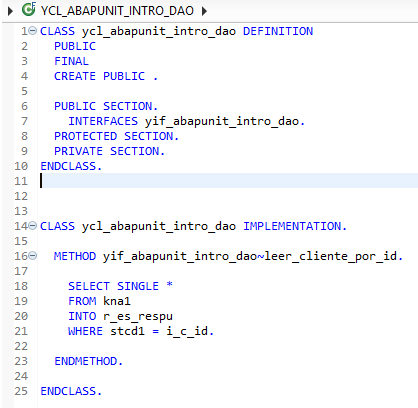
Definición de Interfaz

Se define clase local **ltd\_abapunit\_intro\_mock\_dao** que implementa la interfaz **yif\_abapunit\_intro\_dao**.



*Definición de clase test double (mock) para simular acceso a base de datos*

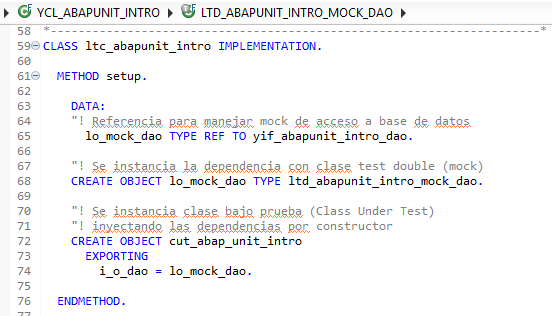
Al mismo tiempo puede existir una clase **ycl\_abapunit\_intro\_dao** con la implementación de acceso a base de datos que implementa la misma interfaz **yif\_abapunit\_intro\_dao**.



*Implementación de interfaz con código productivo*

Inyección de dependencia en clase de prueba

Se realiza inyección de dependencia con clase mock para ejecución de pruebas instanciando la clase mock **ltd\_abapunit\_intro\_mock\_dao** y enviándola como referencia de la interfaz a la clase principal por medio del constructor



*Inyección de dependencia por constructor*

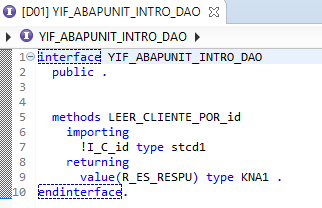
## Dependencia para acceso a base de datos

**Definición de interfaz**

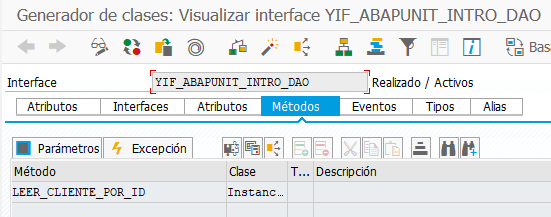
Se define una interfaz con el sufijo DAO (Data Access Object)

<L:Z:Y>IF\_<NombreInterfaz>\_DAO

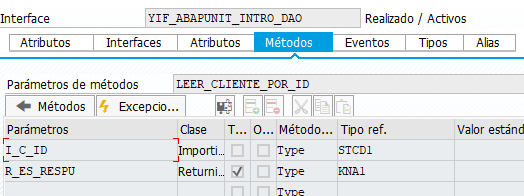
Se agrupan los métodos de acceso a base de datos del desarrollo y de llamados a BAPI o módulos de función estándar.



*Objeto en Eclipse*



*Objeto en editor ABAP*



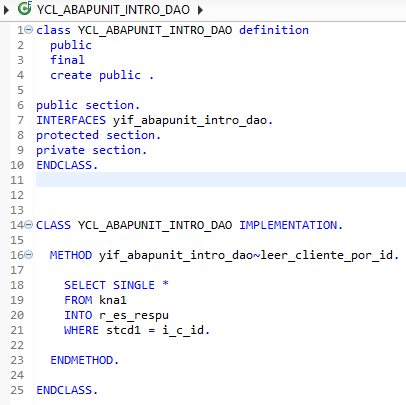
*Firma de método*

**Definición de Clase**

Se define una clase con el sufijo DAO (Data Access Object)

<L:Z:Y>CL\_<NombreClase>\_DAO

Se implementan los métodos de acceso a base de datos.



*Definición de clase en Eclipse*

## **Dependencia para conexión a sistema remoto**

Se define una clase con el sufijo CAO (Connection Access Object)

Para manejar las conexiones a otros sistemas como lo son interfaces de PI.

<L:Z:Y>CL\_<NombreClase>\_CAO

Se agrupan los métodos que consumen proxys de PI o sistemas externos.

## **Dependencia para acceso a parámetros**

Se define una interfaz con sufijo CNF con los métodos para consulta de configuración o parámetros.

<L:Z:Y>IF\_<NombreInterfaz>\_CNF

En este caso, si solo se utilizan parámetros de la ZTCX0001 entonces se puede hacer uso de la interface YIF\_CX\_PARAMETROS.

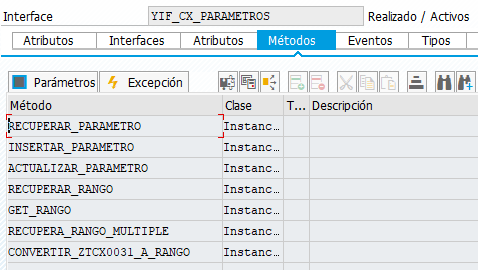
Se define una clase con sufijo CNF para manejo de la configuración de la aplicación y acceso a parámetros, que implemnta la anterior interfaz.

<L:Z:Y>CL\_<Módulo>\_<NombreClase>\_CNF

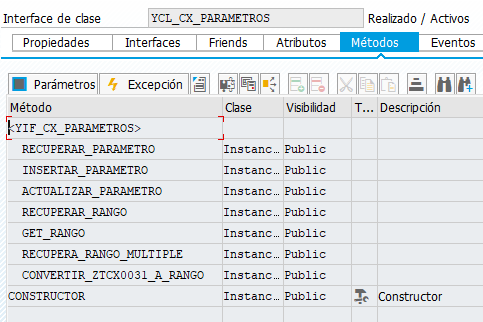
Para el caso de la clase de parámetros de la ZTCX0001 se utiliza como clase productiva la clase YCL\_CX\_PARAMETROS.

Ejemplo:

Interfaz para consulta de parámetros ZTCX0001

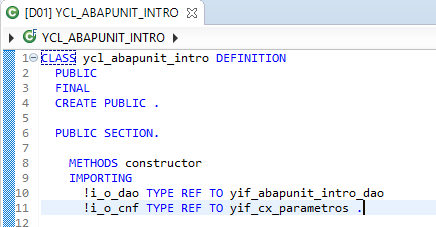
****

Clase productiva que implementa la interfaz

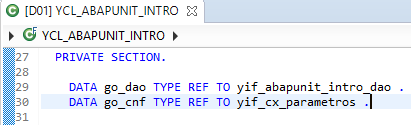


Ejemplo de inyección de dependencia para configuración y parámetros.

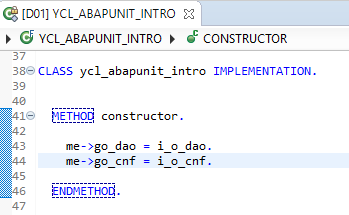
Se define por constructor que se requiere una referencia utilizando la interfaz YIF\_CX\_PARAMETROS.



Se asigna en el constructor la instancia recibida a un atributo global de la clase.

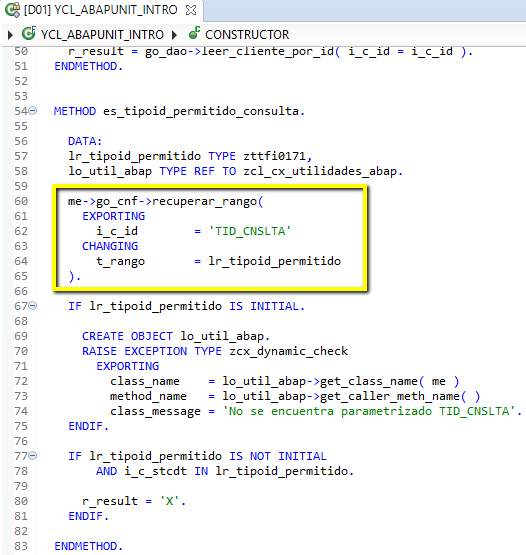


*Atributo global definido como tipo de la interfaz.*



*Asignación de atributo en el constructor.*

De esta manera se utiliza el acceso a los parámetros utilizando la referencia a la interfaz sin tener la referencia a la clase concreta.



## **Inyección de dependencias con método setter**

## 