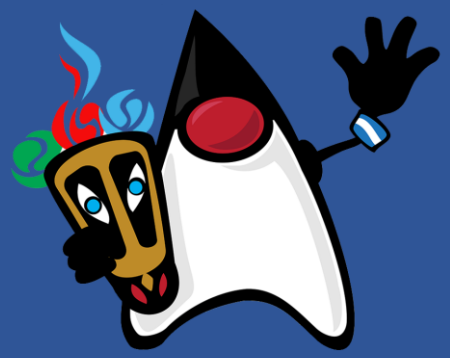
MOCKS en PRUEBAS UNITARIAS

PARA QUE SIRVE Y CUANDO USARLO

por
Francisco Contreras



Francisco Contreras

Membro del Core del Java User Group de Nicaragua

CTO y co-fundador de GInIEm S.A

Master en Informática Empresarial

Desarrollador Java con mas de 10 años de experienca

Conferencista Internacional

Apasionado de las tecnologías

/fcontreras @Frank_JCG /in/fjcontrerasg/

TEMAS

Conceptos básicos

Ejemplo de pruebas unitarias con Junit

Ejemplo de pruebas unitarias con Junit – Mockito

Conclusiones

¿QUÉ SON PRUEBAS UNITARIAS?

Una **prueba unitaria** es una forma/método que permite comprobar de que una **unidad/pieza funcional** de un sistema, bajo un **entorno determinado**, produce los **resultados esperados**.

CAPAS DE UN SISTEMA

Vistas

Controladores

Capa de Servicios

Lógica de Negocios

Capa de acceso a datos

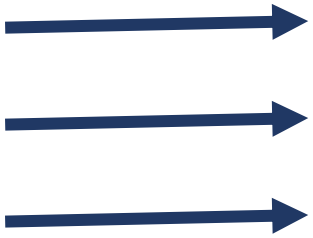
Repositorio - Data Access Object

Base de Datos

HABLEMOS DE PRUEBAS UNITARAS

CONCEPTO DE CAJA NEGRA

Entradas



Salida

CONCEPTO DE CAJA NEGRA

```
a = 3
b = 4
```

```
public int suma(final int a, final int b) {
    return a + b;
}
```

```
resultado: 7
```

CONCEPTO DE CAJA NEGRA

```
a = 3
b = 4
```

Entradas

```
public int suma(final int a, final int b) {
    return a + b;
}
```

```
resultado: 7
```

CONCEPTO DE CAJA NEGRA

```
a = 3
b = 4
```

Entradas

```
public int suma(final int a, final int b) {
    return a + b;
}
```

Sujeto de pruebas

```
resultado: 7
```

CONCEPTO DE CAJA NEGRA

```
a = 3
b = 4
```

Entradas

```
public int suma(final int a, final int b) {
    return a + b;
}
```

Sujeto de pruebas

Valor esperado

```
resultado: 7
```

CONCEPTO DE CAJA NEGRA

Dado que...

Entradas

Cuando...

Lógica de función

Entonces...

Verificaciones y revisiones

HORA DE CODIFICAR PRUEBAS UNITARAS

VERIFICACIONES MAS USADAS

assertEquals

Verifica que dos objetos o primivitos sean **iguales**

assertTrue

Verifica que el valor esperado evalúe **true**

assertFalse

Verifica que el valor esperado evalúe **false**

assertNotNull

Verifica que el valor esperado no sea **null**

VERIFICACIONES MAS USADAS

assertNull

Verifica que el valor esperado sea **null**

assertSame

Verifica que dos referencias de objeto apunten al mismo objeto

assertNotSame

Verifica que dos referencias de objeto no apunten al mismo objeto

assertArrayEquals

Verifica que dos arreglos sean iguales

ANOTACIONES RELEVANTES

@RunWith

Determina con **que motor** se van a ejecutar las pruebas

@Test

Le indica al motor que **métodos** deben considerarse **pruebas**

@BeforeClass

Se utiliza para ejecutar una funciona **antes de todas las pruebas** de las clase

@AfterClass

Se utiliza para ejecutar una funciona **despues de todas las pruebas** de las clase

ANOTACIONES RELEVANTES

@Before

Le indica al motor que debe ejecutar un método **antes de cada prueba**

@After

Le indica al motor que debe ejecutar un método **despues de cada prueba**

@Ignore

Le indica al motor que el metodo o clase anotada debe **ignorarse**

CONTROL DE ENTORNOS Y COMPONENTES

Entradas

Salida

CONTROL DE ENTORNOS Y COMPONENTES

CONTROL DE ENTORNOS Y COMPONENTES

```
public class ServicioDeProductos {

    private RepositorioDeImpuestos impuestosRep;
    private RepositorioDeProductos productosRep;

    public BigDecimal calcularMontoAPagarPorProducto(
        final int idDeProducto, final Localidad lugarDeCompra) {

        Producto producto = this.productosRep.obtenerPorId(idDeProducto);
        BigDecimal impuestoUnitario = this.impuestosRepositorio
            .obtenerImpuestosPorLugar(producto.categoria, lugarDeCompra);

        return producto.precioUnitario.add(impuestoUnitario);

    }
}
```

CONTROL DE ENTORNOS Y COMPONENTES

```
public class ServicioDeProductos {

    private RepositorioDeImpuestos impuestosRep;
    private RepositorioDeProductos productosRep;

    public BigDecimal calcularMontoAPagarPorProducto(
        final int idDeProducto, final Localidad lugarDeCompra) {

        Producto producto = this.productosRep.obtenerPorId(idDeProducto);
        BigDecimal impuestoUnitario = this.impuestosRepositorio
            .obtenerImpuestosPorLugar(producto.categoria, lugarDeCompra);

        return producto.precioUnitario.add(impuestoUnitario);

    }
}
```

CONTROL DE ENTORNOS Y COMPONENTES

```
public class ServicioDeProductos {

    private RepositorioDeImpuestos impuestosRep;
    private RepositorioDeProductos productosRep;

    public BigDecimal calcularMontoAPagarPorProducto(
        final int idDeProducto, final Localidad lugarDeCompra) {

        Producto producto = this.productosRep.obtenerPorId(idDeProducto);
        BigDecimal impuestoUnitario = this.impuestosRepositorio
            .obtenerImpuestosPorLugar(producto.categoria, lugarDeCompra);

        return producto.precioUnitario.add(impuestoUnitario);

    }
}
```

CONTROL DE ENTORNOS Y COMPONENTES

```
public class ServicioDeProductos {

    private RepositorioDeImpuestos impuestosRep;
    private RepositorioDeProductos productosRep;

    public BigDecimal calcularMontoAPagarPorProducto(
        final int idDeProducto, final Localidad lugarDeCompra) {

        Producto producto = this.productosRep.obtenerPorId(idDeProducto);
        BigDecimal impuestoUnitario = this.impuestosRepositorio
            .obtenerImpuestosPorLugar(producto.categoria, lugarDeCompra);

        return producto.precioUnitario.add(impuestoUnitario);

    }
}
```

CONTROL DE ENTORNOS Y COMPONENTES

```
public class ServicioDeProductos {

    private RepositorioDeImpuestos impuestosRep;
    private RepositorioDeProductos productosRep;

    public BigDecimal calcularMontoAPagarPorProducto(
        final int idDeProducto, final Localidad lugarDeCompra) {

        Producto producto = this.productosRep.obtenerPorId(idDeProducto);
        BigDecimal impuestoUnitario = this.impuestosRepositorio
            .obtenerImpuestosPorLugar(producto.categoria, lugarDeCompra);

        return producto.precioUnitario.add(impuestoUnitario);

    }
}
```

CONTROL DE ENTORNOS Y COMPONENTES

CONTROL DE ENTORNOS Y COMPONENTES

Dado que...

Entradas y **Entorno**

Cuando...

Lógica de función

Entonces...

Verificaciones y revisiones

¿QUÉ SON LOS MOCKS?

Un MOCK es un objecto que **simula** ser otro para **suplantarlo** en un entorno determinado. El comportamiento de un mock debe ser programado previamente.

CAPAS DE UN SISTEMA

Vistas

Controladores

Capa de Servicios

Lógica de Negocios

Capa de acceso a datos

Repositorio - Data Access Object

Base de Datos

HORA DE CODIFICAR
CONTROLEMOS EL ENTORNO CON
MOKITO

ANOTACIONES RELEVANTES

@MockBean

Permite la creacion de objectos falsos para crear entornos controlados

@Autowired

Crea e injecta objetos tomando en cuenta los MOCKS creados en el contexto

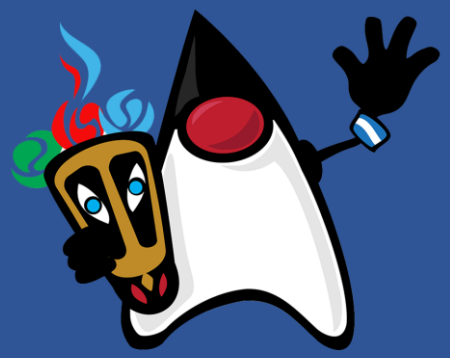
CONCLUSIONES

Ayuda a garantizar la calidad del código

Ayuda a encontrar problemas en las etapas iniciales del desarrollo

Contribuye a la correcta utilización de los patrones de diseño

Reduce costos



Francisco Contreras

Membro del Core del Java User Group de Nicaragua

CTO y co-fundador de GInIEm S.A

Master en Informática Empresarial

Desarrollador Java con mas de 10 años de experienca

Conferencista Internacional

Apasionado de las tecnologías

/fcontreras @Frank_JCG /in/fjcontrerasg/

¿CÓMO INTEGRARSE?

slack

https://jugnicaragua.slack.com

En la WEB

http://javanicaragua.org/

Telegram

https://t.me/jugnicaragua