UNIVERSIDAD TÉCNICA NACIONAL

SEDE SAN CARLOS

CARRERA

INGENIERÍA DEL SOFTWARE

PROYECTO I

CALIDAD DE SOFTWARE

PORFESOR

JOSÉ DAVID CARVAJAL JIMÉNEZ

ELABORADO POR

JOSÉ LUIS SEQUEIRA GÓNGORA

KEVIN ANDRÉS VANEGAS MEDINA

28/03/2022

Contenidos

Propósito

El proyecto Video Club es un sistema de alquiler de películas pensando para un pequeño local que necesita un programa para gestionar estas ventas, para su correcto funcionamiento se debe de crear un rango de pruebas que se necesitan para verificar su calidad. Este programa no se realizó con anterioridad por lo que se debe de aplicar un estudio exhaustivo para saber el comportamiento interno del mismo.

Para realizar las pruebas unitarias se utilizará el lenguaje de programación Java en su última versión (12), también para hacer las pruebas unitarias se necesitará la tecnología TestNG y para realizar Mocks se utilizará el Framework Mockito en su última versión.

Casos de Prueba

Mockito Test:

ID	Descripción	Resultado esperado
1	Prueba del método registrarNuevoProveedor() , este método recibe un dato tipo Proveedor, no retorna nada es un método VOID.	El método recibe un Proveedor, se debe registrar en el sistema.

```
@Test
    public void testRregistrarNuevoProveedor() throws IOException,
ClassNotFoundException {
         //Arrange
        Videoclub videoclub = new Serializacion().deserializarModelo();
        ProveedorController proveedorcontroller = new
ProveedorController(videoclub);
        ProveedorController proveedorspy =
Mockito.spy(proveedorcontroller.getClass());
        Long telefono=Long.parseLong("2844389");
        Proveedor proveedor=new Proveedor("CDCMX",
telefono,"jose@gmial.com");

doNothing().when(proveedorspy).registrarNuevoProveedor(isA(Proveedor.class));
        //Act
        proveedorspy.registrarNuevoProveedor(proveedor);
        //Assert
        verify(proveedorspy, times(1)).registrarNuevoProveedor(proveedor);
    }
```

ID	Descripción	Resultado esperado
2	Prueba del método AgregarPeliculaMockito ejecuta el método de registrar una nueva película	Se espera que el método intente el registro de una nueva película

Codificado:

```
@Test
    public void AgregarPeliculaMockito() throws IOException, ClassNotFoundException{

        PeliculaController peliculaController = mock(PeliculaController.class);

        Genero genero = new Genero("Drama","Peliculas de drama");
        Pelicula pelicula = new
Pelicula("32322","IT",Long.valueOf(2019),Long.valueOf(180),genero,"Pelicula de payaso");
        Pelicula pelicula1 = new
Pelicula("32325342","CODA",Long.valueOf(2022),Long.valueOf(190),genero,"Pelicula basada en
hechos reales");
        doNothing().when(peliculaController).registrarNuevaPelicula(pelicula1);
        peliculaController.registrarNuevaPelicula(pelicula);
        verify(peliculaController,times(1)).registrarNuevaPelicula(pelicula);
```

ID	Descripción	Resultado esperado
3	Prueba del método AgregarBonoMockito ejecuta el método de registrar un nuevo bono	Se espera que el método intente el registro de un nuevo bono

Codificado:

```
      @Test
          public void AgregarBonoMockito() throws IOException,
      ClassNotFoundException{

              BonoController bonoController = mock(BonoController.class);

              Bono bono = new Bono(12500,45,10000.0);
              Bono bono1 = new Bono(5000,43,9000.0);

              doNothing().when(bonoController).registrarNuevoBono(bono1);
              bonoController.registrarNuevoBono(bono);
              verify(bonoController,times(1)).registrarNuevoBono(bono);

          }
```

ID	Descripción	Resultado esperado
4	Prueba del método AgregarClienteMockito ejecuta el método de registrar un nuevo cliente	Se espera que el método intente el registro de un nuevo cliente

Codificado:

```
@Test
    public void AgregarClienteMockito() throws IOException,
ClassNotFoundException{

        ClienteController clienteController =
mock(ClienteController.class);

        Cliente cliente = new
Cliente(Long.valueOf("1233"),"Vanegas","Kevin","vanegasmk@gmail.com",Long.
valueOf("1232324"));
        Cliente cliente1 = new
Cliente(Long.valueOf("12333"),"Guerra","Armando","guerra@gmail.com",Long.v
alueOf("12323243343"));
```

ID	Descripción	Resultado esperado
5	Prueba del método eliminarProveedor() , este método recibe un dato tipo int[] con los indexes a borrar, no retorna nada es un método VOID.	El método recibe un array tipo int[] con el index 0, se debe borrar el proveedor en el index 0.

```
    @Test
    public void testEliminarProveedor() throws IOException,
ClassNotFoundException {
        //Arrange
        Videoclub videoclub = new Serializacion().deserializarModelo();
        ProveedorController proveedorcontroller =
mock(ProveedorController.class);
        int[] eliminar = {0};

doNothing().when(proveedorcontroller).eliminarProveedor(isA(int[].class));
        //Act
        proveedorcontroller.eliminarProveedor(eliminar);
        //Assert
        verify(proveedorcontroller, times(1)).eliminarProveedor(eliminar);

    }
```

Tests:

ID	Descripción	Resultado esperado
6	Prueba del método buscarClientePorDNIValido buscará validar si no existe el DNI dentro el archivo de texto.	Se espera que el método devuelva true, que lo que significa que el DNI no existe en la base de información y se puede utilizar.

Codificado:

```
@Test
    public void buscarClientePorDNIValido() throws IOException,
ClassNotFoundException, Exception {
        //Arrange
        Videoclub videoclub = new Serializacion().deserializarModelo();

        //Act
        ClienteController clienteController = newClienteController(videoclub);
        long id = 12345;
        boolean result = true;

        boolean resultExpected = clienteController.dniClienteValido(id);
        //Assert
        Assert.assertEquals(result, resultExpected);
    }
```

ID	Descripción	Resultado esperado
7	Prueba del método buscarClientePorDNInvalid o buscará validar si existe dentro el archivo de texto.	Se espera que el método retorne false, que significa que si existe información relacionado al DNI y no se puede utilizar.

Codificado:

```
@Test
    public void buscarClientePorDNIInvalido() throws IOException,
ClassNotFoundException, Exception {
        //Arrange
        Videoclub videoclub = new Serializacion().deserializarModelo();

        //Act
        ClienteController clienteController = new ClienteController(videoclub);
        long id = 12344;

        boolean result = clienteController.dniClienteValido(id);
        //Assert
        Assert.assertTrue(result, "El DNI del cliente ya está registrado");
    }
```

ID	Descripción	Resultado esperado
8	Prueba del método buscarClientePorIndex devuelve el cliente que se elije mediante los jframe.	Se espera que el método retorne el mismo cliente que se elige manualmente.

Codificado:

```
@Test
    public void buscarClientePorIndex() throws IOException, ClassNotFoundException,
Exception {
        // Arrange
        Videoclub videoclub = new Serializacion().deserializarModelo();

        //Act
        ClienteController clienteController = new ClienteController(videoclub);
        int index = 0;
        Cliente result = clienteController.obtenerCliente(0);

        Cliente resultExpected = clienteController.obtenerCliente(index);
        //Assert
        Assert.assertEquals(result, resultExpected);

    }
```

ID	Descripción	Resultado esperado
9	Prueba del método obtenerBonosPorCliente muestra todos los bonos que se tiene vinculados en por cada cliente.	Se espera que el método retorne los id de los bonos que se tienen vinculados en los clientes.

Codificado:

```
@Test
    public void obtenerBonosPorCliente() throws IOException, ClassNotFoundException, Exception
{
        // Arrange
        Videoclub videoclub = new Serializacion().deserializarModelo();

        //Act
        ClienteController clienteController = new ClienteController(videoclub);
        Cliente cliente = new Cliente(Long.valueOf("1233"), "Vanegas", "Kevin",
"vanegasmk@gmail.com", Long.valueOf("1232324"));

        ArrayList<Integer> result = clienteController.obtenerBonosDelCliente(cliente);

        ArrayList<Integer> resultExpected = clienteController.obtenerBonosDelCliente(cliente);
        //Assert
        Assert.assertEquals(result, resultExpected);

    }
```

ID	Descripción	Resultado esperado
10	Prueba del método obtenerSansionesPorClientes busca mirar si se tiene vinculados sanciones en los clientes.	Se espera que el método devuelva si el cliente tiene sanciones ya sea que venga vacío o no.

Codificado:

```
  @Test
      public void obtenerSansionesPorCliente() throws IOException, ClassNotFoundException,
  Exception {
          // Arrange
          Videoclub videoclub = new Serializacion().deserializarModelo();

          //Act
          ClienteController clienteController = new ClienteController(videoclub);
          Cliente cliente = new Cliente(Long.valueOf("1233"), "Vanegas", "Kevin",
  "vanegasmk@gmail.com", Long.valueOf("1232324"));

          ArrayList<Sancion> result = new ArrayList<>();

          ArrayList<Sancion> resultExpected = clienteController.obtenerSanciones(cliente);
          //Assert
          Assert.assertEquals(result, resultExpected);

      }
```

ID	Descripción	Resultado esperado
11	Prueba del método buscarBonoPorIndex devuelve el bono que se elije mediante los jframe.	Se espera que el método retorne el mismo cliente que se elige manualmente.

Codificado:

```
@Test
    public void buscarBonoPorIndex() throws IOException, ClassNotFoundException,
Exception {
        // Arrange
        Videoclub videoclub = new Serializacion().deserializarModelo();

        //Act
        BonoController bonoController = new BonoController(videoclub);
        int index = 0;
        Bono result = bonoController.obtenerBono(index);

        Bono resultExpected = bonoController.obtenerBono(index);
        //Assert
        Assert.assertEquals(result, resultExpected);

    }
```

ID	Descripción	Resultado esperado
12	Prueba del método clientePuedeAlquilar deja si un cliente puede alquilar alguna película	Se espera que el método retorne true dando a entender que no tiene deudas y puede contratar una pelicula

Codificado:

```
@Test
    public void clientePuedeAlquilar() throws IOException, ClassNotFoundException,
Exception {
        // Arrange
        Videoclub videoclub = new Serializacion().deserializarModelo();

        //Act
        AlquilerController alquilerController = new AlquilerController(videoclub);
        Cliente cliente = new Cliente(Long.valueOf("1233"), "Vanegas", "Kevin",
"vanegasmk@gmail.com", Long.valueOf("1232324"));
        boolean result = true;

        boolean resultExpected = alquilerController.clientePuedeAlquilar(cliente);
        //Assert
        Assert.assertEquals(result, resultExpected);

    }
```

ID	Descripción	Resultado esperado
13	Prueba del método disponible(), este método recibe un dato tipo Película para confirmar si hay disponibles.	Se espera que el método reciba un dato tipo Película y retorne true ya que existe 1 película en stock

Codificado:

```
@Test
    public void testDisponible() throws IOException, ClassNotFoundException {
        //Arrange
        Videoclub videoclub = new Serializacion().deserializarModelo();
        PeliculaController peliculaController = new
PeliculaController(videoclub);
        Pelicula pelicula = peliculaController.obtenerPelicula(0);
        //Act
        boolean result = peliculaController.disponible(pelicula);

        //assert
        Assert.assertEquals(result, true);
    }
verify(peliculaController,times(1)).registrarNuevaPelicula(pelicula);

    }
```

ID	Descripción	Resultado esperado
14	Prueba del método getPeliculaCount, este método Este método no recibe ningún dato y retorna un dato tipo int.	Se espera que retorne un dato tipo int con el valor de 1 que corresponde al número de películas.

Codificado:

```
    @Test
    public void testGetPeliculaCount() throws IOException,
ClassNotFoundException {
        //Arrange
        Videoclub videoclub = new Serializacion().deserializarModelo();
        PeliculaController peliculaController = new
PeliculaController(videoclub);
        //Act
        int result = peliculaController.getGeneroCount();
        //assert
        Assert.assertEquals(result, 1);
    }
```

ID	Descripción	Resultado esperado
15	Prueba del método obtenerPelicula, Este método recibe un dato tipo int [] y devuelve una película con sus características.	Se espera que el método reciba un dato tipo int[] y retorne un dato tipo Película que corresponde a índice enviado

Codificado:

```
     @Test
    public void testObtenerPelicula() throws IOException,
ClassNotFoundException {
        //Arrange
        Videoclub videoclub = new Serializacion().deserializarModelo();
        PeliculaController peliculaController = new
PeliculaController(videoclub);
        //Act
        Pelicula result = peliculaController.obtenerPelicula(0);
        //Assert
        Assert.assertEquals(result.getClass(), Pelicula.class);
    }
```

ID	Descripción	Resultado esperado
16	Prueba del método codigoPeliculaValido(), este método recibe un String con el código y retorna un Boolean.	Se espera que el método reciba un dato tipo String con el código y retorne true.

Codificado:

```
        @Test
    public void testCodigoPeliculaValido() throws IOException,
ClassNotFoundException {
        //Arrange
        Videoclub videoclub = new Serializacion().deserializarModelo();
        PeliculaController peliculaController = new
PeliculaController(videoclub);
        String codigo = "009933-EF99";
        //Act
        boolean result = peliculaController.codigoPeliculaValido(codigo);
        //Assert
        Assert.assertEquals(result, true);
```

ID	Descripción	Resultado esperado
17	Prueba del método nombreGeneroValido() este método recibe un String con el código y retorna un Boolean.	Se espera que el método reciba un dato tipo String con el código y retorne true.

Codificado:

```
    @Test
    public void testNombreGeneroValido() throws IOException,
ClassNotFoundException {
        //Arrange
        Videoclub videoclub = new Serializacion().deserializarModelo();
        PeliculaController peliculaController = new
PeliculaController(videoclub);
        String nombreGenero = "SUSPENSO";
        //Act
        boolean result = peliculaController.nombreGeneroValido(nombreGenero);
        //Assert
        Assert.assertEquals(result, true);
```

ID	Descripción	Resultado esperado
18	Prueba del método buscarEjemplarPorCodigo() este método recibe un String con el código y retorna un valor tipo Ejemplar	Se espera que el método reciba un dato tipo String con el código y retorne un valor tipo Ejemplar.

Codificado:

```
@Test
    public void testBuscarEjemplarPorCodigo() throws Exception {
        //Arrange
        Videoclub videoclub = new Serializacion().deserializarModelo();
        PeliculaController peliculaController = new
PeliculaController(videoclub);
        //Act
        String codigo = "009933-EF99";
        Ejemplar result = peliculaController.buscarEjemplarPorCodigo(codigo);
        //Assert
        Assert.assertEquals(result.getClass(), Ejemplar.class);
    }
```

ID	Descripción	Resultado esperado
19	Prueba del método buscarProveedorPorNombre() este método recibe valor tipo String y retorna un valor tipo Proveedor	Se espera que el método reciba un dato tipo String con el código y retorne un valor tipo Proveedor.

Codificado:

```
@Test
    public void testBuscarProveedorPorNombre() throws IOException,
ClassNotFoundException {
     //Arrange
        Videoclub videoclub = new Serializacion().deserializarModelo();

        ProveedorController proveedorcontroller = new
ProveedorController(videoclub);

        Proveedor proveedor = new Proveedor();

        Object result = proveedorcontroller.buscarProveedorPorNombre("JOSE
LUIS");
         //Act
        Object expectecProveedor =
proveedorcontroller.buscarProveedorPorNombre("JOSE LUIS");

        //assert
        Assert.assertEquals(result, expectecProveedor);
    }
```

ID	Descripción	Resultado esperado
20	Prueba del método nombreProveedorValido() este método recibe valor tipo String y retorna un valor boolean.	Se espera que el método reciba un dato tipo String con el código y retorne un valor el valor true.

Codificado:

```
    @Test
    public void testNombreProveedorValido() throws IOException,
ClassNotFoundException {
     //Arrange
        Videoclub videoclub = new Serializacion().deserializarModelo();

        ProveedorController proveedorcontroller = new
ProveedorController(videoclub);

        Proveedor proveedor = new Proveedor();

     //Act
        boolean result = proveedorcontroller.nombreProveedorValido("jose
luis");

      //assert
        Assert.assertEquals(result, true);
    }

    }
```

Resultados Obtenido

ID	Detalle	Evidencia
1	Pasó	Controladores.ProveedorControllerNGTest.testBuscarProveedorPorNombre passed (0.049 s)
2	Pasó	Controladores.PeliculaControllerNGTest.AgregarPeliculaMockito passed (0.577 s)
3	Pasó	Controladores.BonoControllerNGTest.AgregarBonoMockito passed (0.56 s)
4	Pasó	Controladores.ClienteControllerNGTest.AgregarClienteMockito passed (0.592 s)
5	Pasó	Controladores.ProveedorControllerNGTest.testEliminarProveedor passed (0.516 s)
6	Pasó	Controladores.ClienteControllerNGTest.buscarClientePorDNIValido passed (0.005 s)
7	Pasó	Controladores.ClienteControllerNGTest.buscarClientePorDNIInvalido passed (0.044 s)
8	Pasó	Controladores.ClienteControllerNGTest.buscarClientePorIndex passed (0.007 s)
9	Pasó	Controladores.ClienteControllerNGTest.obtenerBonosPorCliente passed (0.005 s)
10	Pasó	Controladores.ClienteControllerNGTest.obtenerSansionesPorCliente passed (0.005 s)
11	Pasó	Controladores.BonoControllerNGTest.buscarClientePorIndex passed (0.046 s)

12	Pasó	Controladores.AlquilerControllerNGTest.clientePuedeAlquilar passed (0.042 s)
13	Pasó	Controladores.PeliculaControllerNGTest.testDisponible passed (0.004 s)
14	Pasó	Controladores.PeliculaControllerNGTest.testGetPeliculaCount passed (0.005 s)
15	Pasó	Controladores.PeliculaControllerNGTest.testObtenerPelicula passed (0.004 s)
16	Pasó	Controladores.PeliculaControllerNGTest.testCodigoPeliculaValido passed (0.005 s)
17	Falló	Controladores.PeliculaControllerNGTest.testNombreGeneroValido Failed: java.lang.AssertionError: expected [true] but found [false]
18	Pasó	Controladores.PeliculaControllerNGTest.testBuscarEjemplarPorCodigo passed (0.043 s)
19	Pasó	Controladores.ProveedorControllerNGTest.testBuscarProveedorPorNombre passed (0.041 s)
20	Pasó	Controladores.ProveedorControllerNGTest.testNombreProveedorValido passed (0.006 s)

Calificación

1. *Equipo de trabajo*

Nombre del alumno	E-mail	Teléfono
José Luis Sequeira Góngora	jsequeiragon@est.utn.ac.cr	85022903
Kevin Vanegas Medina	**vanegasmk@gmail.com**	6309 1969

2. *Contenido del documento*

Aspecto del documento	Ponderación	Obtenida
1. Portada	5	
2. Tabla de contenidos	5	
3. Introducción 3.1. Propósito Proposito de las pruebas a realizar. Describir la funcionalidad de las clases a prueba. 3.2. Casos de pruebas Esta sección del Plan de Pruebas se debe presentar todos los casos de prueba identificados para asegurar la calidad del software. Tome en cuenta que debe utilizar el formato visto en clase.	15	
4. Resultados obtenidos Estado de la ejecución de los casos de prueba. (Generalmente esto lo brinda la herramienta de pruebas)	5	

3. *Código ejecutable de las pruebas.*

Rubro	**Ponderación**	**Obtenida**
Casos de prueba formato texto	10	
Casos de prueba codificados	10	
Métodos acorde a los requerimientos	10	
Uso del patrón AAA	10	
Uso del framework TestNG	10	
Uso correcto de anotaciones	10	
El Código se ejecuta sin errores	10	
Uso de Mocks (investigación, uso de la herramienta, implementación)	30	