

Documento de Pruebas de sistema

- Control de Versiones
 - Version 0.1.0 - Autores: Cavanagh Juan, Casabella Martin, Kleiner Matías - Fecha: 25/06/2017 - Descripción: primera versión del documento de pruebas.

1. Introducción

1.1 Propósito

El documento de pruebas tiene como finalidad poner a prueba los requerimientos de usuario y de sistema. Se presentan tres tipos de prueba para ello:

1. Pruebas unitarias.
2. Pruebas de integración.
3. Pruebas de sistema

2. Unit Test

2.1 Clase Asientos -- AsientosTest.java

@Before

```
public void setUp() {}
```

Descripción: Seteamos asientos de prueba como ocupados, para utilizar luego.

@Test

```
public void testBooking() {}
```

Descripción: Se prueba que efectivamente estará ocupado el asiento.

@Test

```
public void testAsientos() {}
```

Descripción: probamos el constructor de la clase.

@Test

```
public void testIsAvailable() {}
```

Descripción: Se testea la disponibilidad de asientos.

@Test

```
public void testUnbookingAll() {}
```

Descripción: se prueba si un asiento está ocupado, utilizando el método Unbooking() liberar el mismo.

@Test

```
public void testGetShowTime() {}
```

Descripción: Se comprueba la disponibilidad de horarios de función.

@Test

```
public void testUnbookingAll() { }
```

prueba el método para liberar todos los asientos ocupados.

2.2 Clase PeliculasModelTest -- PeliculasModelTest.java

@Before

```
public void setUp() {}
```

Descripción: Lee base de datos para su posterior uso.

@Test

```
public void testReadData() {}
```

Descripción: comprobamos los tipos de salas disponibles.

@Test

```
public void testGetAllMovies() {}
```

Descripción :se comprueba el funcionamiento del método obtener todos los títulos, obteniendo uno en particular y comparándolo con un string del mismo título.

@Test

```
public void testGetAllTheaters() {}
```

Descripción: comprueba de disponibilidad de sala seteada con anterioridad.

@Test

```
public void testGetCount() {}
```

Descripción: comprueba el funcionamiento del método cuenta de tickets.

2.3 Clase Peliculas -- PeliculasModelTest.java

@Before

```
public void setUp() {}
```

Descripción: Usa el constructor de Películas, para crear un objeto película con determinados parámetros a usar en los test siguientes.

@Test

```
public void testMovie() {}
```

Descripción: evalúa el funcionamiento del método, comparando los parámetros de la película creada en el SetUp().

@Test

```
public void testAddTheater() {}
```

Descripción: Se prueba el método adherir sala, pasando los parámetros correspondiente, y verificamos que se setean correctamente.

@Test

```
public void testsetTitle() {}
```

Descripción: se prueba el método agregar título. Se comprueba si el título es adherido correctamente.

@Test

```
public void testsetDescription() {}
```

Descripción: se prueba el método para agregar una descripción, Comparando una descripción agregada a la película creada anteriormente.

@Test

```
public void testgetTitle() {}
```

Descripción: se comprueba el metodo obtener título, comparando con un string el título que tiene la película de prueba.

2.4 Clase Salas -- SalasTest.java

@Before

```
public void setUp() {}
```

Descripción: se crea una nueva sala, con determinados parámetros, se agrega un horario de función a esa sala.

@Test

```
public void testSalas() {}
```

Descripción: se verifica el funcionamiento del constructor, creando una sala, y comparando todos los parámetros, demostrando que fueron seteados correctamente.

@Test

```
public void testAddShowTime() {}
```

Descripción: se prueba el método que adhiere una función a una determinada hora.

@Test

```
public void testGetPrice() {}
```

Descripción: Se prueba el método obtener precio de la función.

@Test

```
public void testGetLevel() {}
```

Descripción: Se prueba el método obtener tipo de sala.

@Test

```
public void testGetName() {}
```

Descripción: Se prueba el método obtener nombre de sala.

@Test

```
public void testGetRow() {}
```

Descripción: Se prueba el método obtener cantidad filas de asientos.

@Test

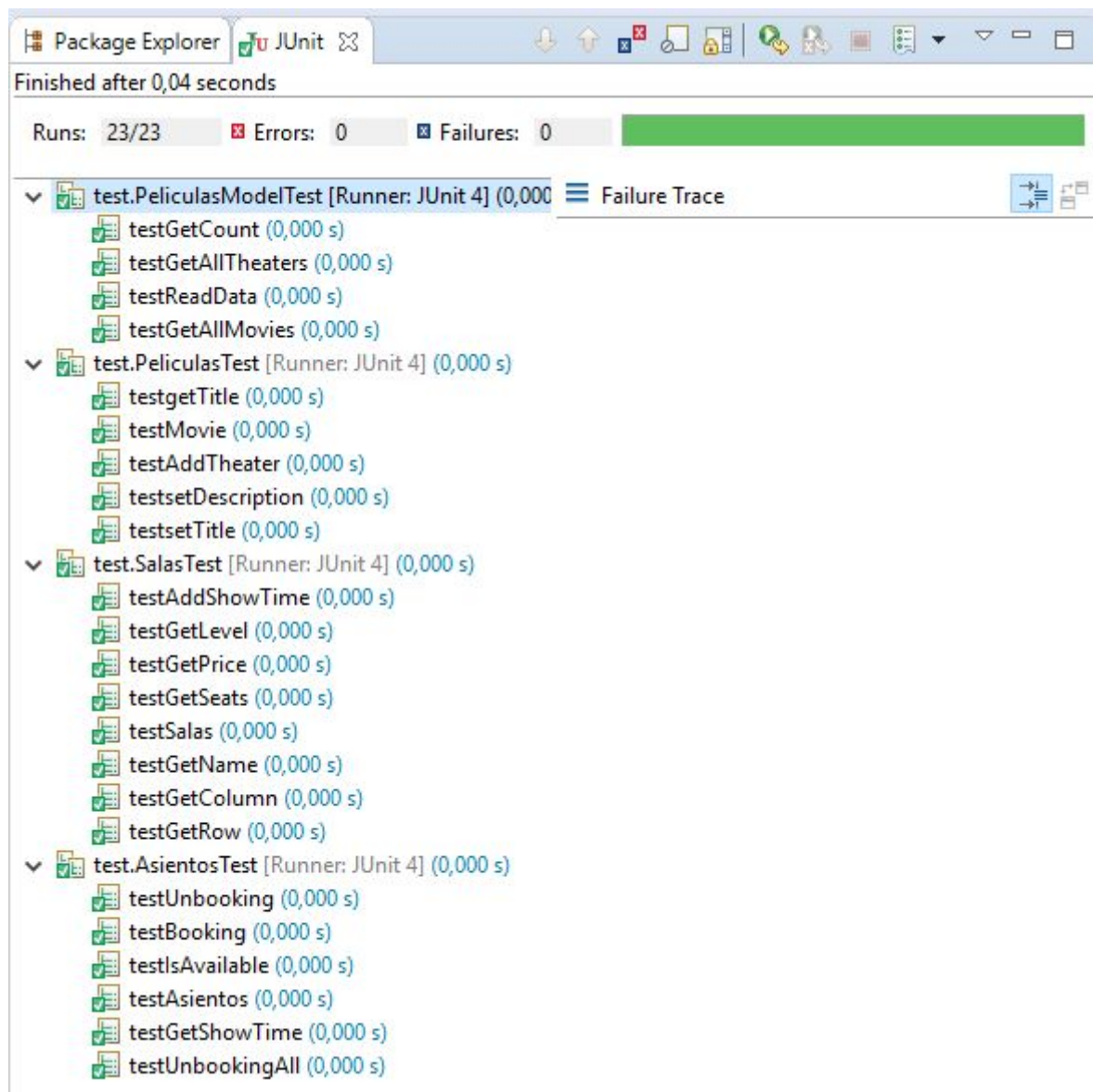
```
public void testGetColumn() {}
```

Descripción: Se prueba el método obtener cantidad columnas de asientos.

@Test

```
public void testGetSeats() {}
```

Descripción: se comprueba si hay asientos en esa hora.



Captura de todos los test superados exitosamente.

3. Pruebas del sistema

Para las pruebas de sistema abrimos el ejecutable .jar y vemos que el mismo da el resultado esperado.

The screenshot shows the BITicket v1.0 application window. On the left, a sidebar titled "Películas disponibles" lists: "Un don excepcional", "Logan", "Guardianes de la Galaxia", "Wonder Woman", and "Comunicaciones digitales". Below this list is a "Seleccionar" button. The main area has a header with "Película seleccionada:" and "Duración:" labels, each followed by a text input field. To the right of these are "Sala" and "Horarios de inicio" dropdown menus, and a clock showing "00:47:14". The center of the screen features a large movie poster placeholder. To the right of the poster are three input fields labeled "Cant. asientos", "Precio total", and "Precio x asiento". At the bottom right, there are "Reservar" and "Deshacer" buttons.

Como siguiente paso, seleccionamos una película y cliqueamos el botón seleccionar. Se espera que la operación muestre las salas y los horarios para la película seleccionada.

Películas disponibles

Un don excepcional

Logan

Guardianes de la Galaxia

Wonder Woman

Comunicaciones digitales

Seleccionar

Película seleccionada:

Un don excepcional

Resuelve muchas integrales

Duración:

200 mins

Sala

Sala B

Horarios de inicio

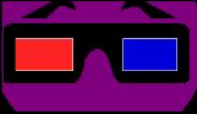
23:00

Cant. asientos

Precio total

Precio x asiento

140.0



Reservar

Deshacer

Vemos que el resultado es correcto. Ahora podemos seleccionar las salas y el horario junto con sus asientos.

Películas disponibles

Un don excepcional

Logan

Guardianes de la Galaxia

Wonder Woman

Comunicaciones digitales

Seleccionar

Película seleccionada:

Un don excepcional

Resuelve muchas integrales

Duración:

200 mins

Sala

Sala B

Horarios de inicio

23:00

00:53:51

Cant. asientos

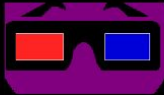
4

Precio total

560.0

Precio x asiento

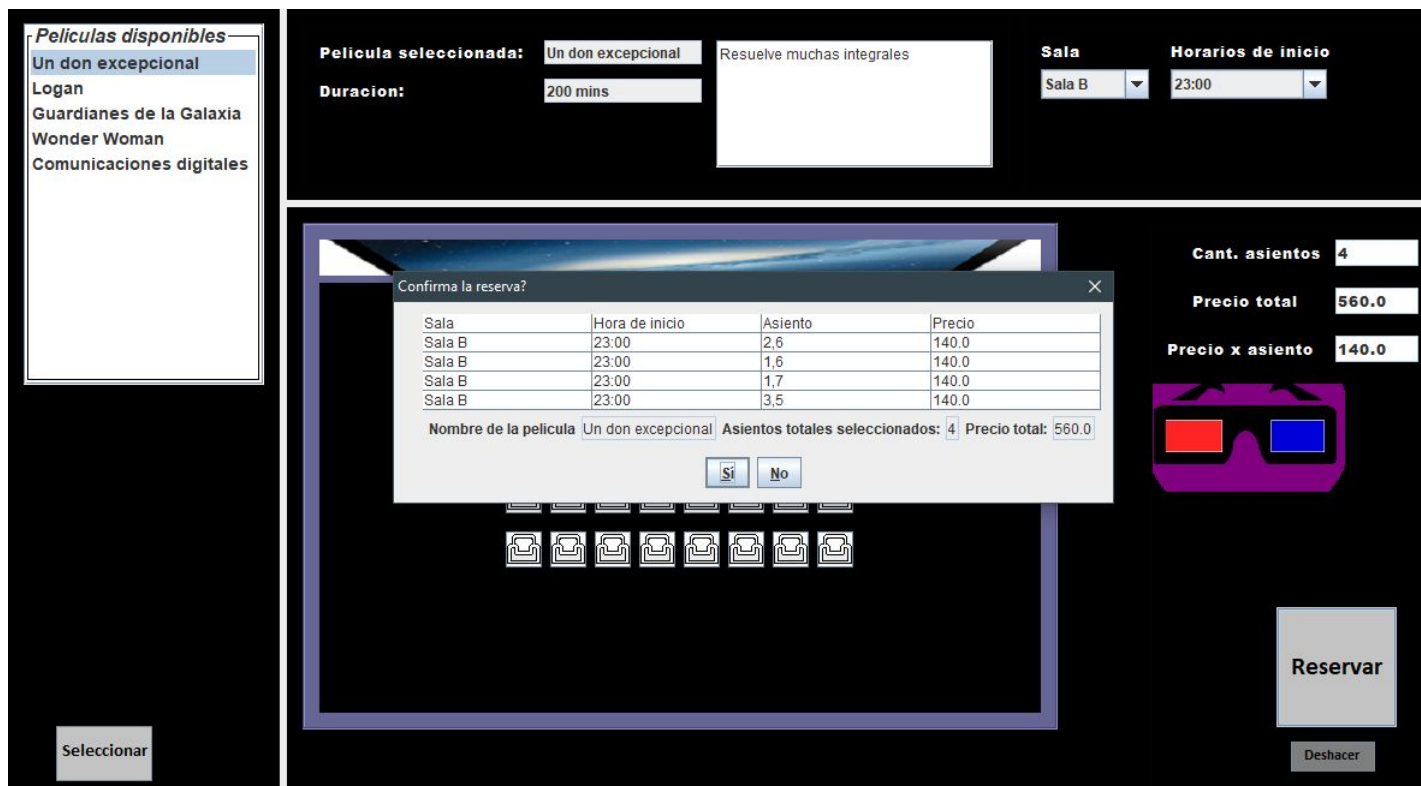
140.0



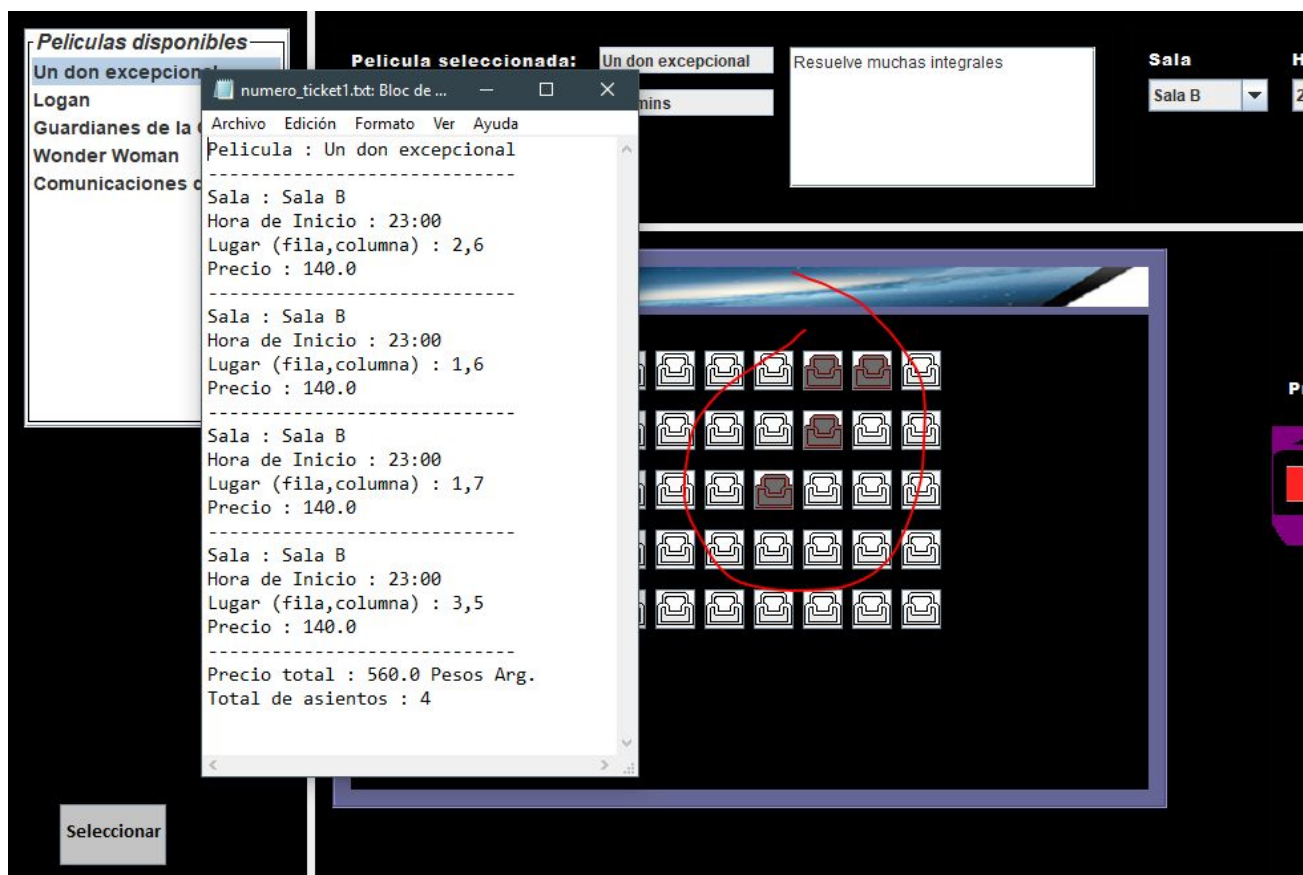
Reservar

Deshacer

Como era esperado, el resultado de la operación fue correcta. Proseguimos a reservar los asientos haciendo click en reservar. El resultado esperado es que se muestre un mensaje con el detalle de la operación y confirmación o cancelación del mismo.



Por último, si se confirma la operación deberíamos poder ver el ticket generado.



El ticket es generado correctamente y ahora los asientos se encuentran ocupados, no disponibles para su reelección.

3. Pass/Fail Ratio

Se hicieron diversas pruebas unitarias teniendo en cuenta que en las mismas no podemos instanciar un objeto distinto de la clase que se está probando, en caso de necesitar hacerlo se usaron mocks objects o stubs.

Como se puede observar, se cuenta con un 100% de pruebas superadas exitosamente, aunque hay que tener en cuenta que faltaron hacer las pruebas de integración del sistema, por lo que estamos en una etapa temprana como para poder brindar un porcentaje de pruebas pasadas/falladas.

4. Errores en la implementación

Con las pruebas del sistema notamos los siguientes errores dejando abierta la posibilidad de, en algún futuro, implementar su corrección. Los errores fueron:

- Al adquirir asientos (esto es, reservar un ticket) la aplicación deshabilita correctamente esos asientos para la función seleccionada evitando así que se pueda volver a adquirir los mismos. El error está cuando cerramos la aplicación, como pensamos a los asientos como objetos los mismos no quedan guardados en una base de datos volviendo a quedar habilitados esos asientos.
- La base de datos es local, sin posibilidad de un acceso remoto a la misma. En un futuro se implementará un manejo de la misma con acceso remoto.
- No hay una interfaz en la aplicación para los administradores del cine. Para cambiar la base de datos, hay que hacerlo desde un editor de texto o por ejemplo, Excel.