Universidad de Granada

Facultad de Ingeniería Informática

Practica 3: Pruebas software

UNIVERSIDAD DE GRANADA

DS: Grupo 1.7

Emanuel Giraldo Herrera
Thomas Lang
Timur Sorokin
Alejando Iborra Morán

(2023-2024)

Contents

1 Planificación

1.1 "Creador de personajes"

- **Test 1**: CFG de ClaseBuilder
  Propósito: Verificar que la función loadCFG() de ClaseBuilder funcióna correctamente.
- **Test 2**: CFG de PersonajeBuilder
  Propósito: Confirmar que la función loadCFG() de PersonajeBuilder funcióna correctamente.
- **Test 3**: Creación correcta del personaje
  Propósito: Asegurar que la creación de un personaje utilizando PersonajeBuilder se realiza correctamente.
- **Test 4**: Exportación Correcta del personaje
  Propósito: Verificar que la exportación del personaje se realiza correctamente.
- **Test 5**: funciónamiento del director
  Propósito: Confirmar que el director puede crear un personaje correctamente.
- **Test 6**: funciónamiento de la fachada
  Propósito: Verificar que la fachada funcióna como se espera.

1.2 "Gestor de personajes"

- **Test 1**: Añadir Personaje
  Propósito: Confirmar que se pueden agregar personajes a la lista del gestor.
- **Test 2**: Eliminar Personaje
  Propósito: Verificar que se pueden eliminar personajes de la lista del gestor.
- **Test 3**: Ordenar la lista según el nombre
  Propósito: Confirmar que la lista ordenada de la lista segun el nombre funcióna correctamente.
- **Test 4**: Ordenar la lista según la clase
  Propósito: Confirmar que la lista ordenada de la lista según el clase funcióna correctamente.
- **Test 5**: Ordernar la lista según la raza
  Propósito: Confirmar que la lista ordenada de la lista según el raza funcióna correctamente.
- **Test 6**: Ordernar la lista según el atributo
  Propósito: Confirmar que la lista ordenada de la lista según el atributo funcióna correctamente.
- **Test 7**: Filtrar la lista según la clase
  Propósito: Confirmar que el filtrado de la lista segun el clase funcióna correctamente.
- **Test 8**: Filtrar la lista según la raza
  Propósito: Confirmar que el filtrado de la lista según el raza funcióna correctamente.

2 Análisis

2.1 Creador de personajes

2.2 CFG de ClaseBuilder

Elemento a Probar:

La función loadCFG() de la clase ClaseBuilder.

Condiciones para la Prueba:

Se requiere que exista un archivo de configuración válido para la clase en cuestión. La función loadCFG() debe cargar correctamente los valores del archivo de configuración.

Datos Necesarios para la Prueba:

Un archivo de configuración válido que contenga los atributos de la clase con sus respectivos valores. Valores esperados para cada atributo de la clase según el archivo de configuración.

2.3 CFG de PersonajeBuilder

Elemento a Probar:

La función loadCFG() de la clase PersonajeBuilder.

Condiciones para la Prueba:

Se requiere que exista un archivo de configuración válido para el personaje en cuestión. La función loadCFG() debe cargar correctamente los valores del archivo de configuración.

Datos Necesarios para la Prueba:

Un archivo de configuración válido que contenga los atributos del personaje con sus respectivos valores. Valores esperados para cada atributo del personaje según el archivo de configuración.

2.4 Creación correcta del personaje

Elemento a Probar:

El proceso de creación de un personaje utilizando PersonajeBuilder.

Condiciones para la Prueba:

Se deben cargar correctamente los valores de los archivos de configuración de la clase y el personaje. Los atributos del personaje creado deben coincidir con los valores esperados según las configuraciones de clase y personaje.

Datos Necesarios para la Prueba:

Archivos de configuración válidos para la clase y el personaje. Valores esperados para los atributos del personaje basados en las configuraciones de clase y personaje.

2.5 Exportación Correcta del personaje

Elemento a Probar:

El proceso de exportación de un personaje.

Condiciones para la Prueba:

El personaje debe haber sido creado correctamente. La función de exportación debe generar un archivo con la estructura correcta y los datos del personaje.

Datos Necesarios para la Prueba:

Un personaje creado y válido. Valores esperados para los atributos del personaje en el archivo de exportación.

2.6 funciónamiento del director

Elemento a Probar:

El proceso de creación de un personaje a través del director.

Condiciones para la Prueba:

El director debe ser capaz de crear un personaje utilizando un constructor específico.

Datos Necesarios para la Prueba:

Un constructor válido para el tipo de personaje deseado. Valores esperados para los atributos del personaje creado.

2.7 funciónamiento de la fachada

Elemento a Probar:

El funciónamiento de la fachada en una operación específica.

Condiciones para la Prueba:

La fachada debe ser capaz de realizar la operación deseada correctamente.

Datos Necesarios para la Prueba:

Datos de entrada necesarios para la operación. Valores esperados para el resultado de la operación.

2.8 Gestor de personajes

2.9 Añadir personajes

Elemento a Probar:

La función addPersonaje(Personaje) de GestorPersonajes

Condiciones para la Prueba:

Se requiere que exista un personaje valido ya creado y la instancia de GestorPersonajes

Datos Necesarios para la Prueba:

Un personaje valido ya creado y la instancia de GestorPersonajes

2.10 Eliminar Personajes

Elemento a Probar:

La función remPersonaje(indice) de la clase PersonajeBuilder.

Condiciones para la Prueba:

Se requiere que en GestorPersonajes se haya incluido previamente un personaje valido.

Datos Necesarios para la Prueba:

La instancia de GestorPersonajes con al menos un personaje valido a eliminar. El índice de la posicion del personaje a eliminar de GestorPersonajes

2.11 Ordenar lista de personajes por nombre

Elemento a Probar:

La función ordenarPersonaje(descendente, tipo, atributo) de GestorPersonaje cuando se quiere ordenar la lista por nombre.

Condiciones para la Prueba:

Se requiere que en GestorPersonajes haya al menos 2 personajes con distinto nombre.

Datos Necesarios para la Prueba:

La instancia de GestorPersonajes con al menos 2 personajes de distinto nombre. Una variable de opcion para que el orden sea ascendente o descendente.

2.12 Ordenar lista de personajes por clase

Elemento a Probar:

La función ordenarPersonaje(ascendente, tipo, atributo) de GestorPersonaje cuando se quiere ordenar la lista por clase.

Condiciones para la Prueba:

Se requiere que en GestorPersonajes haya al menos 2 personajes con distinta clase.

Datos Necesarios para la Prueba:

La instancia de GestorPersonajes con al menos 2 personajes de distinto clase. Una variable de opción para que el orden sea ascendente o descendente.

2.13 Ordenar lista de personajes por raza

Elemento a Probar:

La función ordenarPersonaje(descendente, tipo, atributo) de GestorPersonaje cuando se quiere ordenar la lista por raza.

Condiciones para la Prueba:

Se requiere que en GestorPersonajes haya al menos 2 personajes con distinta raza.

Datos Necesarios para la Prueba:

La instancia de GestorPersonajes con al menos 2 personajes de distinto raza. Una variable de opcion para que el orden sea ascendente o descendente.

2.14 Ordenar lista de personajes por atributo

Elemento a Probar:

La función ordenarPersonaje(descendente, tipo, atributo) de GestorPersonaje cuando se quiere ordenar la lista por un atributo.

Condiciones para la Prueba:

Se requiere que en GestorPersonajes haya al menos 2 personajes con el atributo a ordenar distinto.

Datos Necesarios para la Prueba:

La instancia de GestorPersonajes con al menos 2 personajes con distinto valor en el atributo a ordenar. Una variable de opcion para que el orden sea ascendente o descendente. El atributo por el que se quiere ordenar la lista.

2.15 Filtrar lista de personajes por raza

Elemento a Probar:

La función filtrarPorRaza(raza) de GestorPersonaje cuando se quiere filtrar la lista por una raza específica.

Condiciones para la Prueba:

Se requiere que en GestorPersonajes haya al menos un personaje de la raza a filtrar y, al menos, un personaje que no pertenezca a esa raza.

Datos Necesarios para la Prueba:

La instancia de GestorPersonajes con al menos un personaje de la raza a filtrar y, al menos, un personaje que no pertenezca a esa raza.

2.16 Filtrar lista de personajes por clase

Elemento a Probar:

La función filtrarPorClase(clase) de GestorPersonaje cuando se quiere filtrar la lista por una clase especifica.

Condiciones para la Prueba:

Se requiere que en GestorPersonajes haya al menos un personaje de la clase a filtrar y, al menos, un personaje que no pertenezca a esa clase.

Datos Necesarios para la Prueba:

La instancia de GestorPersonajes con al menos un personaje de la clase a filtrar y, al menos, un personaje que no pertenezca a esa clase.

3 Diseño

3.1 Creador de personajes

3.1.1 Grupo: Archivos de configuración

TLDR: Estos casos de prueba aseguran que los archivos de configuración necesarios para construir personajes estén presentes en el sistema y puedan ser accedidos correctamente por el programa.

Casos de prueba

- Configuración de clase: verificar que los archivos de configuración de clase existen en la ruta especificada
- Configuración de raza: verificar que los archivos de configuración de raza existen en la ruta especificada

Entorno de prueba

- No se requieren herramientas ni infraestructuras adicionales.

Datos de prueba

- Se requiere ClaseBuilder debidamente inicializado.
- Se requiere PersonajeBuidler debidamente inicializado.
- Se requieren rutas donde se ubican los archivos de configuración.

Condición base
Los archivos lib/cfg/* existen y son accesibles.

3.1.2 Grupo: ClaseBuilder CFG

TLDR: Verifica que la función `loadCFG()` de ClaseBuilder carga correctamente los atributos de clase.

Casos de prueba:

- Lectura de atributos de clase: verificar que la función `loadCFG()` de ClaseBuilder funcióna correctamente y carga los atributos de clase esperados.

Entorno de prueba:

- No se requieren herramientas ni infraestructuras adicionales.

Datos de prueba:

- Se requiere ClaseBuilder debidamente inicializado.

Condición base:

- ClaseBuilder está inicializado correctamente y ha cargado los valores desde el archivo de configuración correctamente.

3.1.3 Grupo: PersonajeBuilder CFG

TLDR: Verifica que la función `loadCFG()` de PersonajeBuilder carga correctamente los atributos de personaje.

Casos de prueba:

- Lectura de atributos de personaje: verificar que la función `loadCFG()` de PersonajeBuilder funcióna correctamente y carga los atributos de personaje esperados.

Entorno de prueba:

- No se requieren herramientas ni infraestructuras adicionales.

Datos de prueba:

- Se requiere PersonajeBuilder debidamente inicializado.

Condición base:

- PersonajeBuilder está inicializado correctamente y cha cargado los valores desde el archivo de configuración correctamente.

3.1.4 Grupo: Creación de Personaje

TLDR: Verifica que la creación de un personaje utilizando los constructores de ClaseBuilder y PersonajeBuilder produce un personaje con los atributos esperados.

Casos de prueba:

- Verificar que los atributos del personaje son los esperados después de su creación.

Entorno de prueba:

- No se requieren herramientas ni infraestructuras adicionales.

Datos de prueba:

- Se requieren ClaseBuilder y PersonajeBuilder debidamente inicializados.

Condición base:

- ClaseBuilder y PersonajeBuilder están inicializados correctamente y la construcción de los atributos en el builder da el mismo resultado que `loadCFG()`.

3.1.5 Grupo: Exportación del estado

TLDR: Verifica que la exportación del estado de un personaje se realiza correctamente y que se genera el archivo esperado.

Casos de prueba:

- Verificar que el archivo de exportación del estado del personaje se genera correctamente en la ubicación especificada.

Entorno de prueba:

- No se requieren herramientas ni infraestructuras adicionales.

Datos de prueba:

- Se requiere un personaje correctamente construido y una ubicación de archivo de exportación válida.

Condición base:

- El personaje está correctamente construido y la ubicación de archivo de exportación es válida. Finalmente que el archivo efectivamente se ha creado.

3.1.6 Grupo: funciónamiento Director

TLDR: Verifica que el director pueda crear un personaje correctamente y que el personaje creado tenga el nombre esperado.

Casos de prueba:

- Verificar que el director pueda crear un personaje con el nombre especificado.
- Verificar que el nombre del personaje creado coincida con el nombre especificado.

Entorno de prueba:

- No se requieren herramientas ni infraestructuras adicionales.

Datos de prueba:

- Se requiere un director correctamente inicializado y un nombre de personaje válido.

Condición base:

- El director está correctamente inicializado y puede crear un personaje con el nombre especificado.

3.1.7 Grupo: funciónamiento de la Fachada

TLDR: Verifica que la fachada pueda crear un personaje correctamente con los constructores proporcionados y que el personaje creado tenga el nombre esperado.

Casos de prueba:

- Verificar que la fachada pueda crear un personaje utilizando los constructores de ClaseBuilder y PersonajeBuilder.
- Verificar que el nombre del personaje creado coincida con el nombre especificado.

Entorno de prueba:

- No se requieren herramientas ni infraestructuras adicionales.

Datos de prueba:

- Se requiere una fachada correctamente inicializada, un constructor de ClaseBuilder y PersonajeBuilder válidos, y un nombre de personaje válido.

Condición base:

- La fachada está correctamente inicializada y puede crear un personaje utilizando los constructores proporcionados.

3.2 Gestor personajes

Se encarga de verificar el funciónamiento adecuado del gestor de personajes en el sistema. Este gestor gestiona una la lista de personajes, permitiendo añadir, eliminar y realizar operaciones como ordenar y filtrar por diferentes criterios, como nombre, clase, raza o atributo.

3.2.1 Añadir Personaje

TLDR: Este caso de prueba verifica que el gestor de personajes pueda añadir correctamente un personaje a la lista de personajes.

Casos de prueba:

- Se añade un personaje a la lista del gestor de personajes.

Entorno de prueba:

- Se necesita un gestor de personajes inicializado y un personaje válido.

Datos de prueba:

- Gestor de personajes correctamente inicializado.
- Personaje válido.

Condición base:

- El gestor de personajes está correctamente inicializado y la lista de personajes está vacía.

3.2.2 Eliminar Personaje

TLDR: Este caso de prueba verifica que el gestor de personajes pueda eliminar correctamente un personaje de la lista de personajes.

Casos de prueba:

- Se elimina un personaje de la lista del gestor de personajes.

Entorno de prueba:

- Se necesita un gestor de personajes inicializado y un personaje válido en la lista del gestor.

Datos de prueba:

- Gestor de personajes correctamente inicializado.
- Personaje válido dentro del gestor.

Condición base:

- El gestor de personajes está correctamente inicializado y el personaje esta en la lista del gestor.

3.2.3 Ordenar lista por nombre

TLDR: Este caso de prueba verifica que el gestor de personajes puede ordenar la lista de personajes segun el nombre

Casos de prueba:

- Se desea ordenar la lista del gestor segun el nombre de los personajes

Entorno de prueba:

- Se necesita un gestor de personajes inicializado con varios personajes en su lista y de distinto nombre.

Datos de prueba:

- Gestor de personajes correctamente inicializado.
- Variable para ordenar la lista de forma ascendente y descendete
- Personajes validos dentro de la lista con nombres diferentes

Condición base:

- El gestor de personajes está correctamente inicializado y los personajes estan en su lista

3.2.4 Ordenar lista por raza

TLDR: Este caso de prueba verifica que el gestor de personajes puede ordenar la lista de personajes segun el raza

Casos de prueba:

- Se desea ordenar la lista del gestor segun el raza de los personajes

Entorno de prueba:

- Se necesita un gestor de personajes inicializado con varios personajes en su lista y de distinto raza.

Datos de prueba:

- Gestor de personajes correctamente inicializado.
- Variable para ordenar la lista de forma ascendente y descendete
- Personajes validos dentro de la lista con razas diferentes

Condición base:

- El gestor de personajes está correctamente inicializado y los personajes estan en su lista

3.2.5 Ordenar lista por clase

TLDR: Este caso de prueba verifica que el gestor de personajes puede ordenar la lista de personajes segun el clase

Casos de prueba:

- Se desea ordenar la lista del gestor segun el clase de los personajes

Entorno de prueba:

- Se necesita un gestor de personajes inicializado con varios personajes en su lista y de distinto clase.

Datos de prueba:

- Gestor de personajes correctamente inicializado.
- Variable para ordenar la lista de forma ascendente y descendete
- Personajes validos dentro de la lista con clase diferentes

Condición base:

- El gestor de personajes está correctamente inicializado y los personajes estan en su lista

3.2.6 Ordenar lista por atributo

TLDR: Este caso de prueba verifica que el gestor de personajes puede ordenar la lista de personajes segun el atributo

Casos de prueba:

- Se desea ordenar la lista del gestor segun el atributo elegido de los personajes

Entorno de prueba:

- Se necesita un gestor de personajes inicializado con varios personajes en su lista y que tengan el atributo elegido de distinto valor

Datos de prueba:

- Gestor de personajes correctamente inicializado.
- Variable para ordenar la lista de forma ascendente y descendete
- Atributo a elegir para la lista ordenada
- Personajes validos dentro de la lista con con el valor del atributo diferentes

Condición base:

- El gestor de personajes está correctamente inicializado y los personajes estan en su lista

3.2.7 Filtrar lista por raza

TLDR: Este caso de prueba verifica que el gestor de personajes puede filtrar la lista de personajes segun la raza

Casos de prueba:

- Se desea filtrar la lista del gestor segun el raza de los personajes

Entorno de prueba:

- Se necesita un gestor de personajes inicializado con varios personajes en su lista, al menos uno de la raza a filtrar y al menos otro de distinta raza.

Datos de prueba:

- Gestor de personajes correctamente inicializado.
- Personajes validos dentro de la lista, con al menos un personaje de la raza a filtrar y otro de distinta raza.

Condición base:

- El gestor de personajes está correctamente inicializado y los personajes estan en su lista.

3.2.8 Filtrar lista por clase

TLDR: Este caso de prueba verifica que el gestor de personajes puede filtrar la lista de personajes segun la clase

Casos de prueba:

- Se desea filtrar la lista del gestor segun el clase de los personajes

Entorno de prueba:

- Se necesita un gestor de personajes inicializado con varios personajes en su lista, al menos uno de la clase a filtrar y al menos otro de distinta raza.

Datos de prueba:

- Gestor de personajes correctamente inicializado.
- Personajes validos dentro de la lista, con al menos un personaje de la clase a filtrar y otro de distinta clase.

Condición base:

- El gestor de personajes está correctamente inicializado y los personajes estan en su lista.

4 Conclusiones

Para realizar las pruebas, hemos integrado dos nuevas funciónalidades en nuestra aplicación:

- Gestor de personajes: Este componente se encarga de manejar a los personajes en nuestro sistema. Ofrece una interfaz que permite agregar, eliminar y consultar personajes en una lista. Además, ofrece funciónalidades avanzadas como filtrado y ordenamiento sobre el conjunto de personajes.
- Exportación de estado: Implementamos un método en la clase Personaje para exportar su estado a un archivo de texto. Aunque no especificamos la estructura del archivo en esta etapa, la principal meta era demostrar la capacidad de serializar los datos de un personaje.

Cabe mencionar que previamente hemos realizado dos iteraciones sobre este ejercicio. En la primera iteración, durante practica 1, ofreimos una implementación en Java. En la segunda , migramos a Flutter y reescribimos todo el código en Dart. Durante esta migración nos hemos enfrentado con bastantes desafios que finalmente han sido superados. Como resultado de este esfuerzo logramos obtener un código solido y estable que cumple con los requisitos planteados en las practicas anteriores. Debido a esto, durante las pruebas propuestas para esta practica no tuvimos que recurrir a la depuración ni tampoco a realizar cambios en el codigo base.

A pesar de no haber surgido problemas durante las pruebas de esta práctica, identificamos algunos puntos que requerían atención y que han sido mitigados a tiempo durante la practica 2 que resumimos a continuación:

- Builder: Observamos que se reutilizaba la referencia de Personaje en lugar de crear nuevas instancias durante la construcción de personajes. Esto causaba problemas cuando intentábamos guardar varios personajes en una lista, ya que las modificaciones en una referencia afectaban a otras. La solución fue garantizar que cada construcción generara una instancia única del personaje.
- Archivos de configuración: Detectamos que la ausencia de archivos de configuración causaba fallos en la creación de atributos de personajes. El problema era el uso de caminos absolutos. La solución fue utilizar rutas de archivos relativas.

Aunque el código base demostró ser robusto y confiable, consideramos que la implementación de pruebas y la depuración podrían haber mejorado el proceso de desarrollo. Definir un conjunto claro de requisitos funciónales desde el principio habría agilizado la identificación y resolución de problemas, lo que resultaría en un desarrollo más eficiente y seguro.