Práctica 3 Desarrollo de Software

Memoria del desarrollo de la práctica.

5 de mayo de 2024

Alonso Doña Martinez

Jose Ramón Plaza Plaza

Jaime Parra Jiménez

Carlos Izquierdo Guzmán

Índice

1. Introducción

En este documento se va a realizar todo el proceso de planificación, diseño, implementación y prueba de una serie de tests definidos para mejorar y refinar todo el funcionamiento y desarrollo de nuestra aplicación.

Enlace al repositorio de github: **Desarrollo de Software**

2. Planificación de las pruebas del sistema

Nuestra aplicación, diseñada en Flutter/Dart, se basa en el desarrollo de un constructor de personajes que nos permite crear diferentes personajes con características variadas. Dentro de la estructura del programa, observamos dos patrones definidos:

- **Patrón Builder:** Gestiona la creación de personajes.
- **Patrón Decorator:** Encargado de la creación y personalización de la armadura del personaje.

Además de estos dos patrones que componen la estructura general de la aplicación, encontramos una lista dentro de la app que nos permite gestionar los personajes que vamos creando, permitiendo añadirlos, eliminarlos y mostrarlos de forma visual para el usuario.

Pruebas a realizar

Una vez analizados los tres bloques del programa pasamos a examinarlos a partir de la creación y desarrollo de tests, vamos a definir las pruebas a realizar:

1. **Métodos y funciones relacionados con el patrón Builder:** Se deben probar todas las funciones y métodos relacionados con la creación de personajes para garantizar que se construyan correctamente con todas las características especificadas.
2. **Métodos y funciones relacionados con el patrón Decorator:** Las pruebas deben cubrir la funcionalidad de personalización de la armadura del personaje, verificando que se apliquen correctamente los decoradores y que el resultado final sea el esperado.
3. **Gestión y funcionamiento de la lista de personajes:** Se deben realizar pruebas para asegurar que la lista de personajes se actualiza correctamente al agregar o eliminar personajes, así como verificar que se muestren correctamente para el usuario.

3. Análisis de prueba

Para la realización del análisis de prueba se ha seguido la tabla proporcionada para la práctica, realizándose en primer lugar un estudio de los requisitos funcionales y no funcionales de nuestra aplicación para pasar posteriormente a la realización del análisis.

[Pruebas software] ANÁLISIS
Especificar las distintas condiciones de prueba:

REQUSITOS FUNCIONALES
Dentro de nuestro sistema encontramos los siguientes requisitos.
Funcionalidad: Crear Personaje
- Condiciones a probar:
  - El personaje elegido se crea de forma correcta
  - El personaje final (junto a la armadura) se crea de forma correcta
  - Si intentamos mostrar las características del personaje aun sin crear da error
  - Todos los métodos get y set funcionan bien.
 	
Funcionalidad: Crear Armadura
- Condiciones a probar:
  - La armadura se crea de forma correcta
  - La armadura se puede cambiar en tiempo de ejecución sin problemas
  - Las respuestas de cada Armadura creada son correctas(string dar Apariencia)
  - Si intento pasarle al “setArmadura” un objeto armadura que no está bien creado da error.
 	
Funcionalidad: Gestión Lista
- Condiciones a probar:
  - Los personajes creados se añaden a la lista
  - Puedo acceder a los personajes de la lista.
  - Los personajes presentes en la lista se pueden eliminar.
REQUISITOS NO FUNCIONALES
Mantenibilidad
- Todo el código debe seguir una estructura que le permite ser escalable y fácil de mantener
Cohesión
- Todo el código respeta las diferentes cuestiones de estructura y composición de los patrones de diseño usados.
Seguridad
- Ninguna función ni componente pone en riesgo la integridad del sistema ej : Introducción de personajes erróneos en lista de personajes
Usabilidad
- Métodos necesarios para que el diseño de interfaz de usuario garantice una experiencia intuitiva y fácil de usar.

Figura 1: Análisis de requisitos para la aplicación.

Elemento a probar	Condiciones	Datos requeridos
Creación y Asignación de la armadura al personaje	La Armadura es creada de forma correcta. La Armadura se puede asignar dinámicamente al personaje.	El string “ArmaduraBásica” para pasarle a Armadura. El objeto Armadura que hay que pasarle como parámetro a la armadura decorada. El objeto personajeguerrero de tipo GuerreroBuilder.
Reasignación de la armadura de forma dinámica	La Armadura es modificada correctamente. La Nueva Armadura sustituye a la anterior.	Un objeto Armadura de un tipo diferente a la asignada. El objeto personaje con una armadura asignada con anterioridad.
Método dar apariencia de la clase Armadura	El mensaje correspondiente a la armadura creada es el correcto.	Un objeto armadura decorada ya sea de tipo planta o fuego.
Método setArmadura no acepta Armaduras incorrectas	Usuario puede asignar una armadura al personaje. Mensaje de error si la Armadura creada es incorrecta	Un objeto Personaje creado. Un objeto Armadura creado de forma incorrecta.
Creación de un tipo de personaje específico.	El usuario puede crear un personaje. Los atributos(arma y habilidad) del personaje creado serán acordes a los de su clase	Un objeto Director. El objeto PersonajeBuilder inicializado al tipo deseado ya sea GuerreroBuilder o MagoBuilder.
Creación de un personaje final con Armadura específica.	El personaje se debe crear con sus características correspondientes y con el tipo de armadura especificado. Usuario puede crear un personaje junto a una armadura y asignarse dinámicamente	Un personaje y una Armadura inicializada
Método getTipoPersonaje devuelve de forma correcta el tipo de personaje.	El personaje es creado correctamente. El método getTipoPersonaje devuelve el tipo correcto.	Un objeto director que permite crear el personaje correctamente. Un objeto PersonajeBuilder

Figura 2: Análisis de pruebas.

Método getArmadura devuelve la armadura que contiene el personaje.	El personaje es creado correctamente. El personaje es asignado con una armadura de forma correcta.	Un objeto director que permite crear el personaje correctamente. Varios objetos Armadura para comprobar distintos tipos. Un personajeBuilder.
Método mostrarPersonaje maneja excepción cuando el objeto Personaje aún no ha pasado por el Director.	Mensaje mostrado correctamente. Mensaje de error si el objeto Personaje aún no ha pasado por el director	Un objeto personaje. Un objeto guerreroBuilder/magoBuilder
Método addPersonaje permite añadir un personaje a la lista	El personaje ha sido creado correctamente. La lista ha sido creada correctamente.	El objeto Personaje a insertar La Lista de personajes donde se va a insertar.
Método removePersonaje permite eliminar un personaje de la lista	El personaje es eliminado correctamente. Mensaje de error si la lista no tiene ningún personaje creado. Mensaje de error si el personaje a borrar no está en la lista.	El objeto Personaje a eliminar. La lista de personajes de donde se van a eliminar los personajes.
Obtención personaje de la lista	Lista creada correctamente. Mensaje de error si se intenta acceder a un elemento que no está en la lista.	El objeto lista. Dos objetos PersonajeBuilder que se van a añadir a la lista. El director que va a inicializar los personajes.
Creación de la lista	Solo puede existir una única instancia de la clase Lista. Si se intenta crear otra se devuelve la misma instancia.	Un objeto lista.

Figura 3: Análisis de pruebas -continación anterior.

4. Diseño de pruebas

Una vez realizado el correspondiente análisis de pruebas pasamos al diseño de pruebas, este se realizará siguiendo la tabla proporcionada para la práctica.

[Pruebas software] DISEÑO
Especificar los distintos casos de prueba:
Casos de prueba: Para detectar y definir los distintos casos de prueba se ha realizado un análisis de los elementos y condiciones especificadas en el análisis, de esta forma hemos detectado los siguientes casos, que nos permitirán agrupar los tests a realizar. - Gestión de Armadura: engloba todos los tests relacionados con la creación, modificación y comprobación de métodos propios de la armadura así como de la gestión de errores en su proceso de creación. - Gestión del Personaje: engloba todos los tests relacionados con la creación de personajes incluyendo la asignación de una armadura ya definida previamente al personaje, además se tratarán todos los posibles errores en este proceso. - Gestión de la Lista : engloba todos los tests relacionados con la creación de la lista, respetando siempre su estructura ya se establece como un “singletone”. Además tenemos en cuenta los test para su modificación, es decir, añadir personajes, borrar personajes… Manteniendo siempre las precauciones y tratando todos los errores posibles. **Trazabilidad pruebas-casos:** para realizar este apartado hemos tenido en cuenta todas las diferentes condiciones que cada caso de prueba tiene en nuestra aplicación, consiguiendo a partir de un caso de uso agrupar todos los tests relacionados con él con sus respectivas condiciones.

Figura 4: Diseño de pruebas, especificación casos de prueba.

Mostramos la tabla completa.

Casos de pruebas(ordenados por prioridad)	**Entornos de prueba**	**Datos de prueba**	**Relación con las condiciones de prueba (trazabilidad)**
Gestión de la Armadura	Se hará mediante terminal.	El string "ArmaduraBásica" para pasarle a Armadura. El objeto Armadura que hay que pasarle como parámetro a la armadura decorada. Un objeto Armadura de un tipo diferente a la asignada. Un objeto Personaje creado. Un objeto Armadura creado de forma incorrecta. El objeto personajeguerrero de tipo GuerreroBuilder.	Armadura creada correctamente. Mensaje correspondiente a la armadura creada sea correcto. Armadura se asigna dinámicamente al personaje. Se puede modificar sustituyendo a la armadura anterior. Si se sustituye por una nula se muestra mensaje de error.
Gestión del Personaje	Se hará mediante terminal.	Una Armadura inicializada. Un objeto Director. El objeto PersonajeBuilder inicializado al tipo deseado ya sea GuerreroBuilder o MagoBuilder.	Personaje creado correctamente. Personaje con armadura específica se crea de forma correcta. Todos los métodos que devuelven características de la armadura funcionan Los atributos se asignan de forma correcta a cada tipo de personaje específico. El personaje puede mostrar sus atributos de forma correcta.

Figura 5: Diseño de pruebas.

			Mensaje de error si se intenta mostrar los atributos del personaje sin haber sido creado correctamente.
Gestión de la lista	Se hará mediante terminal.	El objeto lista. Objetos PersonajeBuilder para realizar las comprobaciones. El director que va a inicializar los personajes.	Se pueden añadir personajes de forma correcta. Se pueden eliminar personajes de forma correcta Se muestra excepción si el personaje añadido es nulo.

Figura 6: Diseño de pruebas-continuación.

5. Pruebas realizadas

Primero mostramos los group de cada test y el contenido del método setUp realizado para cada group.

```
group('Test Lista de personajes', () {

  late PersonajeLista lista;
  PersonajeBuilder personaje = GuerreroBuilder();
  Director director = Director(personaje);

  setUp((){
    director.buildPersonaje("guerrero");
    lista = PersonajeLista();
    lista.removePersonajes();
  });
```

Figura 7: Inicio del group con los test de la lista.

```
group('Test Patron Decorator', () {

  PersonajeBuilder personajeguerrero = GuerreroBuilder();

  Armadura armadura = ArmaduraSimple("Armadura Básica");
  Armadura armaduraFuego = FuegoDecorador(armadura);

  Armadura armadura2 = ArmaduraSimple("Armadura Básica");
  Armadura armaduraPlanta = PlantaDecorador(armadura2);


  setUp((){
    Director director = Director(personajeguerrero);
    director.buildPersonaje("guerrero");

    personajeguerrero.setArmadura(armaduraFuego);
  });
```

Figura 8: Inicio del group con los test del patrón decorator.

```
group('Test Patron Builder', () {

  late PersonajeBuilder personajeguerrero;
  late PersonajeBuilder personajemago;

  setUp((){
    personajeguerrero = GuerreroBuilder();
    personajemago = MagoBuilder();
  });
```

Figura 9: Inicio del group con los test del patrón builder.

5.1. Test para la lista de personajes

En estos test nos encargaremos de comprobar la lista la cual guarda los personajes, comprobando que estos se añadan correctamente, se puedan quitar, no haya varias instancias distintas y que los personajes devueltos son los correctos.

```
test('Prueba de que los personajes creados se añaden a la lista', () {
  lista.agregarPersonaje(personaje);

  expect(lista.personajes.contains(personaje), true);
});
```

Figura 10: El personaje creado se puede añadir a la lista correctamente

```
Run | Debug
test('Prueba de que los personajes contenidos en la lista se pueden eliminar', () {

  lista.agregarPersonaje(personaje);
  lista.eliminarPersonaje(personaje);

  expect(lista.personajes.contains(personaje), false);
});
```

Figura 11: Los personajes se pueden eliminar correctamente

```
test('Prueba de que el método para obtener un personaje devuelve el personaje que elegimos correctamente.', () {

  PersonajeBuilder personaje2 = GuerreroBuilder();
  Director director2 = Director(personaje2);
  director2.buildPersonaje('guerrero');

  lista.agregarPersonaje(personaje);
  lista.agregarPersonaje(personaje2);

  expect(lista.obtenerPersonajePorIndice(0), personaje);
  expect(lista.obtenerPersonajePorIndice(1), personaje2);
});
```

Figura 12: Los personajes que devuelve la lista son los correctos

```
test('Prueba de que solo se obtiene una instancia de la lista de personajes', () {
  PersonajeLista lista2 = PersonajeLista();
  expect(identical(lista, lista2), true);
});
```

Figura 13: Solo se puede tener una instancia del objeto lista

5.2. Tests para el patrón decorator

En estos test nos encargaremos de comprobar la funcionalidad correcta del patrón decorator utilizado en las armaduras, veremos que las armaduras se crean bien, se pueden decorar correctamente, se puedan cambiar y que solo acepte armaduras para poder asignarselos a los personajes.

```
test('Comprobar que la armadura que elige el usuario se crea y se le aigna dinamicamente al personaje.', (){

  expect(personajeguerrero.personaje?.getArmadura(), 'Armadura de Fuego');
});
```

Figura 14: La armadura se crea y se asigna dinámicamente

```
test('Prueba para comprobar que si asignamos una armadura y luego la cambiamos se refleja en el personaje.', (){

  expect(personajeguerrero.personaje?.getArmadura(), 'Armadura de Fuego');

  personajeguerrero.setArmadura(armaduraPlanta);
  expect(personajeguerrero.personaje?.getArmadura(), 'Armadura de Planta');
});
```

Figura 15: Comprobación de que si se cambia la armadura se refleja en el personaje

```
test('Comprobar que cuando creo una armadura el dar aparienza funciona a corde a dicho objeto.', (){
  expect(armaduraPlanta.darApariencia(), 'Armadura de Planta');
  expect(armaduraFuego.darApariencia(), 'Armadura de Fuego');
});
```

Figura 16: Funciona la función de dar apariencia de las distintas armaduras

```
test('Comprobar que si al set armadura se le pasa un objeto por parámetro que no es armadura correcta da error.', (){
  Armadura armaduraerror = ArmaduraSimple("ArmaduraBasica");

  expect(() => personajeguerrero.setArmadura(armaduraerror), throwsAssertionError);
});
```

Figura 17: Solo se le puede pasar como parametro a setArmaduras objetos armaduras

5.3. Test para patrón builder

Por ultimo, comprobaremos el correcto funcionamiento del patrón builder, comprobando que los personajes se crean correctamente, solo el director los crea, los tipos son correctos y las funciones get funcionan correctamente.

```
test('Comprobar que si creamos un personaje, se crea con sus características', (){
  Director director = Director(personajeguerrero);
  director.buildPersonaje("guerrero");

  expect(personajeguerrero.personaje?.getArma(), 'Espada');
  expect(personajeguerrero.personaje?.getArmadura(), 'Armadura Básica');
  expect(personajeguerrero.personaje?.getHabilidad(), 'Lucha');
});
```

Figura 18: El personaje creado tiene sus objetos correctos

```
test('Comprobar que si usamos mostrar personaje sin crearlo con el director nos da error.', (){
  expect(personajeguerrero.personaje?.mostrarPersonaje(), null);
});
```

Figura 19: Solo se puede mostrar el personaje si el director lo ha creado

```
test('Prueba para comprobar que el personaje final se crea correctamente', (){
  Director directorGuerrero = Director(personajeguerrero);
  directorGuerrero.buildPersonaje("guerrero");

  Armadura armadura = ArmaduraSimple("ArmaduraBasica");
  Armadura armaduraFuego = FuegoDecorador(armadura);

  personajeguerrero.setArmadura(armaduraFuego);

  expect(personajeguerrero.personaje?.mostrarPersonaje(), 'Armadura: Armadura de fuego, Arma: Espada, Habilidad: Lucha');

  Director directorMago = Director(personajemago);
    directorMago.buildPersonaje("mago");

  Armadura armadura2 = ArmaduraSimple("ArmaduraBasica");
  Armadura armaduraPlanta = PlantaDecorador(armadura2);

  personajemago.setArmadura(armaduraPlanta);

  expect(personajemago.personaje?.mostrarPersonaje(), 'Armadura: Armadura de planta, Arma: Varita, Habilidad: Magía');
});
```

Figura 20: El personaje final se ha creado correctamente

```
test('Prueba que getTipoPersonaje funciona correctamente.', () {

  Director directorMago = Director(personajemago);
  directorMago.buildPersonaje("mago");
  expect(personajemago.getTipoPersonaje(), 'mago');

  Director directorGuerrero = Director(personajeguerrero);
  directorGuerrero.buildPersonaje("guerrero");
  expect(personajeguerrero.getTipoPersonaje(), 'guerrero');
});
```

Figura 21: La funcion que devuelve el tipo de personaje funciona correctamente

```
test('Prueba par comprobar que el metodo para obtener la armadura del personaje devuelve la correcta.', (){
  Director directorGuerrero = Director(personajeguerrero);
  directorGuerrero.buildPersonaje("guerrero");

  Armadura armadura = ArmaduraSimple("ArmaduraBasica");
  Armadura armaduraFuego = FuegoDecorador(armadura);

  personajeguerrero.setArmadura(armaduraFuego);

  expect(personajeguerrero.getArmadura(), armaduraFuego);

  Director directorMago = Director(personajemago);
  directorMago.buildPersonaje("mago");

  Armadura armadura2 = ArmaduraSimple("ArmaduraBasica");
  Armadura armaduraPlanta = PlantaDecorador(armadura2);

  personajemago.setArmadura(armaduraPlanta);

  expect(personajemago.getArmadura(), armaduraPlanta);
});
```

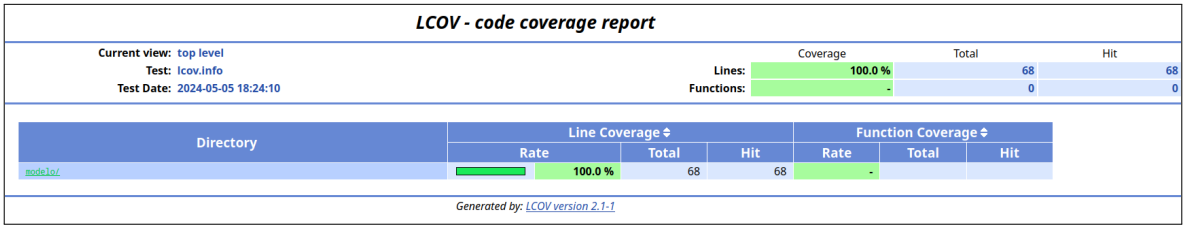
Figura 22: La funcion que devuelve la armadura funciona correctamente

6. Resultados tests

Una vez realizados todos los tests correctamente, los hemos exportado en formato html y abierto en el navegador, durante el proceso de elaboración de los tests esta herramienta ha sido muy útil para detectar errores y código sin revisar, finalmente tras realizar la totalidad de los tests podemos observar que se obtiene el cien por cien de comprobación de todo el código del programa.

```
joserpp@msi ~/Universidad/TERCERO/2CUATRI/DS/Desarrollo-de-Software/P2/ejercicio3 main flutter test --coverage
00:02 +13: All tests passed!
joserpp@msi ~/Universidad/TERCERO/2CUATRI/DS/Desarrollo-de-Software/P2/ejercicio3 main genhtml coverage/lcov.info -o coverage/html
Reading tracefile coverage/lcov.info.
Found 10 entries.
Found common filename prefix "/home/joserpp/Universidad/TERCERO/2CUATRI/DS/Desarrollo-de-Software/P2/ejercicio3/lib"
Generating output.
Processing file lib/modelo/guerrero_builder.dart
  lines=11 hit=11
Processing file lib/modelo/personaje.dart
```

Figura 23: Ejecución de los test en la terminal y creacion del html.



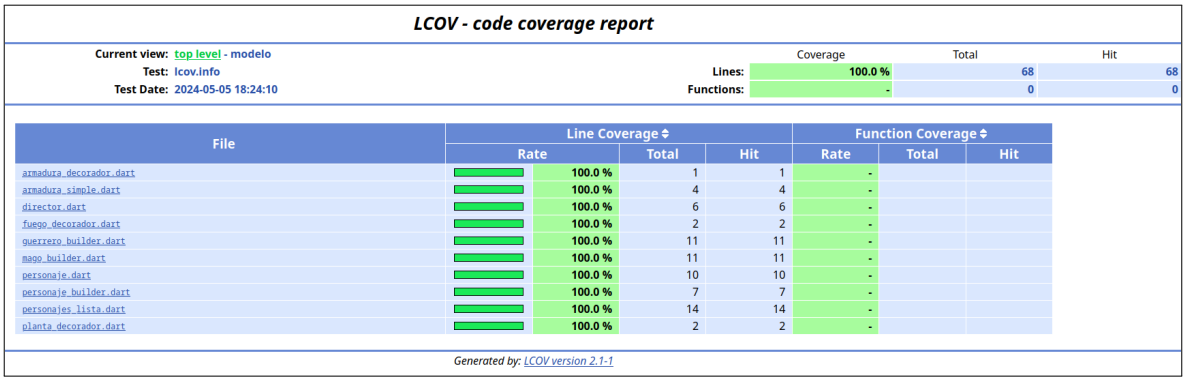
LCOV - code coverage report

		Coverage	Total	Hit
Current view:	top level			
Test:	lcov.info	Lines:	100.0 %	68	68
Test Date:	2024-05-05 18:24:10	Functions:	-	0	0

Directory	Line Coverage			Function Coverage		
	Rate	Total	Hit	Rate	Total	Hit
modelo/	100.0 %	68	68	-		

Generated by: LCOV version 2.1-1

Figura 24: Web principal donde se muestrán los test del modelo.



LCOV - code coverage report

		Coverage	Total	Hit
Current view:	top level - modelo			
Test:	lcov.info	Lines:	100.0 %	68	68
Test Date:	2024-05-05 18:24:10	Functions:	-	0	0

File	Line Coverage			Function Coverage		
	Rate	Total	Hit	Rate	Total	Hit
armadura_decorador.dart	100.0 %	1	1	-		
armadura_simple.dart	100.0 %	4	4	-		
director.dart	100.0 %	6	6	-		
fuego_decorador.dart	100.0 %	2	2	-		
guerrero_builder.dart	100.0 %	11	11	-		
mago_builder.dart	100.0 %	11	11	-		
personaje.dart	100.0 %	10	10	-		
personaje_builder.dart	100.0 %	7	7	-		
personajes_lista.dart	100.0 %	14	14	-		
planta_decorador.dart	100.0 %	2	2	-		

Generated by: LCOV version 2.1-1

Figura 25: Sección de la web donde vemos los test por clases.

7. Cambios Realizados

Durante la realización de esta práctica hemos encontrado ciertos aspectos e implementaciones que tras analizar el conjunto del programa y gracias a las funcionalidades comprobadas en los test hemos decidido modificar.

- **Eliminación de Constructor Innecesario en la Clase Personaje:** Se ha eliminado un constructor innecesario en la clase `Personaje`. Anteriormente, se diseñaba un constructor con argumentos que no se utilizaban en la implementación, lo que ponía en riesgo la integridad del patrón de diseño. Esta mejora simplifica la clase y elimina posibles fuentes de confusión.

- **Reestructuración de la Lista de Personajes:** La estructura de la lista de personajes ha sido completamente reestructurada. Anteriormente, toda la lógica de la lista se manejaba con un vector de `Personajes` incluido en el archivo `PaginaLista.dart`. Ahora, se ha decidido crear un archivo donde se ha definido la clase `Lista` con sus atributos y métodos específicos. Esta reestructuración ha permitido mejorar la abstracción, modularización y la facilidad para el desarrollo de pruebas.

- **Mejoras en el Método `mostrarPersonaje`:** Dentro del método `mostrarPersonaje`, se ha realizado una mejora significativa. En el caso de que no se haya utilizado el método `director` que asigna los diferentes elementos al objeto, en lugar de mostrar elementos nulos, ahora muestra un mensaje de error adecuado. Esta mejora aumenta la robustez y la claridad del código.

- **Método darApariencia en la clase ArmaduraDecorador :** Habíamos realizado la implementación de este método en la clase abstracta ArmaduraDecorador pero no lo usabamos en ningún momento por tanto hemos decidido eliminarlo.

- **Eliminación del método Widget MostrarPersonaje :** Eliminación de un método que creaba un texto como widget de los atributos del personaje, este se encontraba definido en la clase Personaje y finalmente no se usaba en el código.

- **Método setArmadura MagoBuilder y GuerreroBuilder :** Se ha modificado la función para comprobar si la armadura que le pasamos es correcta.

  Muchos de estos errores se han detectado al realizar la comprobación de los test en formato html y visualizar los porcentajes de los métodos no comprobados.