DOCUMENTO TÉCNICO DE ESPECIFICACIONES

Ingeniería del Software II

José Antonio Sousa Porto
Antonio Cadenas Sarmiento

Funcionalidad de la aplicación

Hemos creado una aplicación web basada en una fábrica de juguetes. La aplicación está dividida en varias partes:

- **Pantalla de login**: el usuario introduce su nombre y contraseña y accederá a su vista. La creación de cuentas de usuario deben ser solicitadas al administrador, que le proporcionará los datos necesarios para acceder.

- **Pantalla de administrador**: en la vista de administrador nos encontramos 3 tablas distintas:

  - ***Usuarios***: contiene información sobre todos los usuarios registrados, así como sus datos. Disponemos de un botón para crear usuarios nuevos, para actualizar sus datos y para eliminarlos del sistema.

  - ***Juguetes***: podemos crear un nuevo modelo de juguete que se almacenará en la base de datos, indicando su nombre y descripción. A continuación podemos generar nuevas unidades para ese modelo. Para ello, generamos una tarea que será enviada a un trabajador, que se encargará de crear las unidades del modelo que le hemos indicado. Una vez creados los juguetes dispondremos de un botón para distribuirlos.

    También incluimos un botón para desechar el modelo seleccionado y todo el 'stock' disponible del mismo, así como sus materiales asociados.

  - ***Materiales***: son necesarios para la creación de juguetes. A la hora de introducirlos debemos indicar su nombre, cantidad y juguete asociado. También podremos añadir nuevas unidades de cada modelo y eliminarlos.

- **Pantalla de trabajador**: cada trabajador tiene una tabla en la que aparecen las tareas pendientes que les ha enviado el administrador. Cada tarea tiene asociada un modelo de juguete. Cuando el trabajador pulse en el botón "Terminar Tarea", se generarán las cantidades indicadas previamente por el administrador para ese modelo de juguete y se sumarán a la cantidad total.

Iteraciones Realizadas

Sprint 1 (Preparativos)

Duración

- Desde el 2 de abril hasta el 2 de mayo.

Objetivo

- Establecer un modo de trabajo para el desarrollo de la aplicación, formas de actuación, que debe hacer cada integrante del equipo y dejar todo preparado para comenzar el desarrollo en el siguiente sprint.

Contenido de la pila

- 1. Definir la funcionalidad básica de la aplicación.
- 2. Decidir qué tecnologías utilizar para el desarrollo de la aplicación.
- 3. Tras decidir qué tecnologías y aplicaciones se van a utilizar, descargarlas e instalarlas en los ordenadores de los integrantes.
- 4. Debatir qué controlador de versiones utilizar.
- 5. Establecer la repartición del trabajo.
- 6. Generar archivo de documentación inicial.

Acta de planificación

- Durante la reunión de planificación de éste sprint, hemos decidido crear una aplicación web basada en una fábrica de juguetes. Dicha aplicación contará con un sistema de login y gestión de trabajadores, donde podremos ver la lista de trabajadores, asignarles tareas o cambiarles datos.
- Se podrán añadir, modificar y eliminar nuevos modelos de juguetes y materiales, así como mandar fabricarlos y distribuirlos.
- Hemos decidido utilizar las tecnologías explicadas en clase, ya que pensamos que nos permitirán avanzar de manera más sencilla en el proyecto. Serán las siguientes:
  - IDE NetBeans 8.2
  - GlassFish Server 4.1
  - Base de datos MySql
  - JPA y EJB
  - JDK 1.8
  - JSF 2.2

        - Primefaces 7

- En cuanto al sistema de control de versiones, utilizaremos un repositorio privado de GitHub donde iremos actualizando el código fuente y la documentación.

Acta de revisión

- Durante la sesión de revisión hemos aclarado un punto que no había quedado bien definido del todo. Los usuarios tendrán diferentes opciones en la aplicación tras hacer login en función del rol que les haya sido asignado. Por ejemplo, un trabajador de la fábrica no puede consultar los datos ni eliminar a otro compañero, pues éstas funciones están reservadas para el administrador del sistema.

Sprint 2 (Desarrollo y control de versiones)

Duración

- Desde el 3 de mayo hasta el 31 de mayo.

Objetivo

- Proceder con el desarrollo de la práctica siguiendo las especificaciones y tecnologías anteriormente definidas. Dejar constancia de nuestro trabajo en el programa de control de versiones y entregar el resultado final.

Contenido de la pila

- 1. Crear una primera versión del proyecto.
- 2. Continuar creando el código fuente.
- 3. Compartir el trabajo en GitHub.
- 4. Tras finalizar el desarrollo, comprobar que todo funciona correctamente.
- 5. Realizar la entrega de la versión final.
- 6. Crear pruebas de caja blanca y caja negra de un fragmento del código.

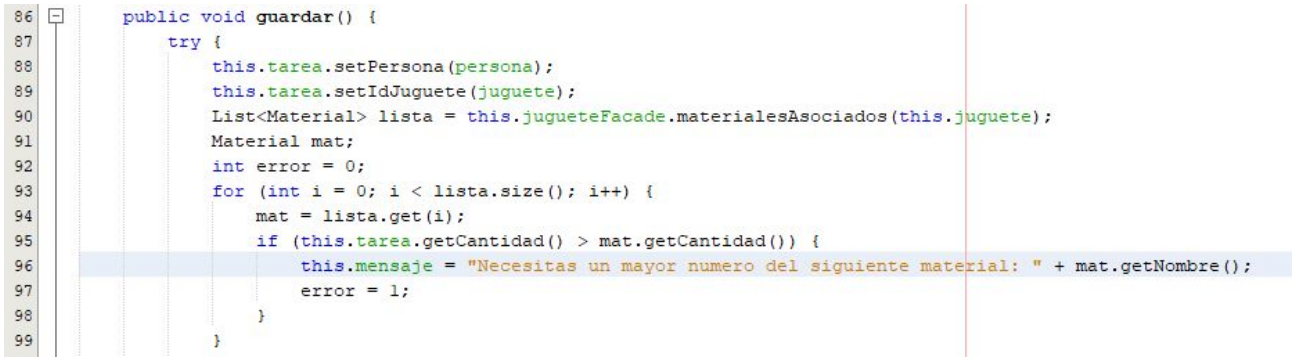
Acta de planificación

- Durante la reunión de planificación de éste sprint, hemos quedado para crear el esqueleto inicial de la aplicación. Creamos el proyecto en NetBeans, importamos todas las librerías y dependencias y levantamos el servidor.
- Establecimos que el código debería estar finalizado para un día antes del final del sprint, día en el cual nos reuniremos para perfeccionar detalles y documentación.

Acta de revisión

- El último día lo utilizamos para solucionar errores que teníamos en la aplicación. Algunos de esos errores eran que no se actualizaba la interfaz cuando se pulsaban los botones o que al sumar cantidad de materiales, el número total de materiales almacenados se sustituye por el número que acabamos introducimos.
- Otro detalle que nos hemos dado cuenta es que dependiendo del ordenador en el que estuviésemos, el tamaño de las tablas variaba. Por ello hemos decidido quitar algunas de las columnas de las tablas para facilitar la visualización, y que debe utilizarse Internet Explorer como navegador principal.

Pruebas de Caja Blanca y Caja Negra



```
86      public void guardar() {
87          try {
88              this.tarea.setPersona(persona);
89              this.tarea.setIdJuguete(juguete);
90              List<Material> lista = this.jugueteFacade.materialesAsociados(this.juguete);
91              Material mat;
92              int error = 0;
93              for (int i = 0; i < lista.size(); i++) {
94                  mat = lista.get(i);
95                  if (this.tarea.getCantidad() > mat.getCantidad()) {
96                      this.mensaje = "Necesitas un mayor numero del siguiente material: " + mat.getNombre();
97                      error = 1;
98                  }
99              }
```

Para la realización de nuestras pruebas de Caja Blanca y Caja Negra, hemos escogido este trozo de código de nuestra clase *TareasController.java.*

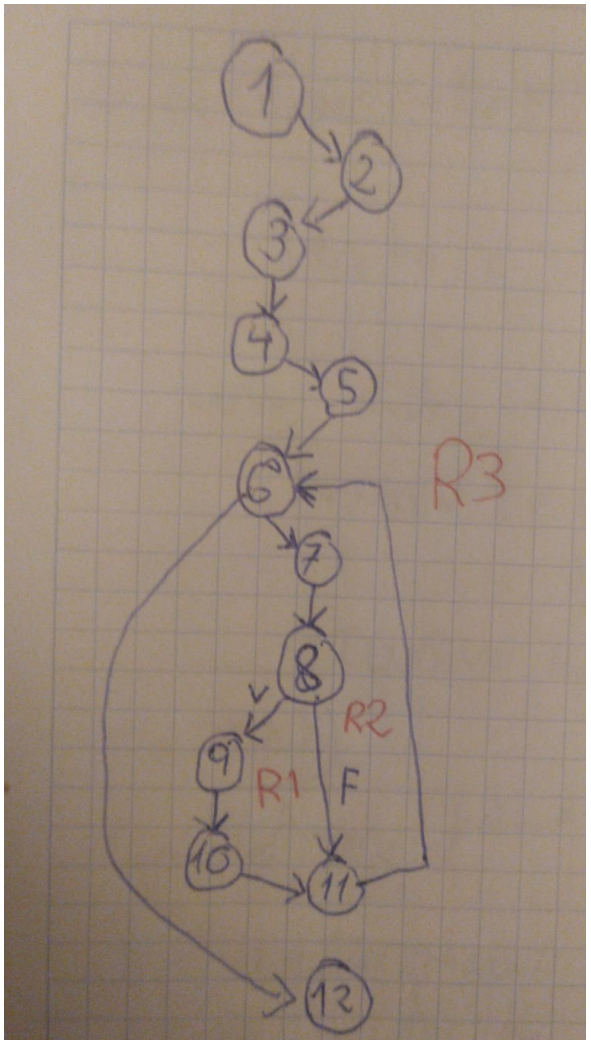
Este método nos permite enviar tareas a nuestros trabajadores para que fabriquen juguetes pero, para ello, debemos comprobar primero que contamos con los suficientes materiales para dicha fabricación. Para realizar esta comprobación, necesitamos una lista de los materiales empleados para la fabricación del juguete deseado, y una vez obtenida, comprobar la disponibilidad de los materiales de 1 en 1. En caso de no contar con suficientes, enviaremos un mensaje de error, y la tarea no se llevará a cabo, aunque esa parte está más adelante en el método.

Hemos seleccionado este trozo de código por contar con una secuencia iterativa (bucle for) y una selectiva (if).

Prueba Caja Blanca

```
public void guardar() {
 try {
1.          this.tarea.setPersona(persona);
2.          this.tarea.setIdJuguete(juguete);
3.          List<Material> lista = this.jugueteFacade.materialesAsociados(this.juguete);
4.          Material mat;
5.          int error = 0;
6.          for (int i = 0; i < lista.size(); i++) {
7.                      mat = lista.get(i);
8.              if (this.tarea.getCantidad() > mat.getCantidad()) {
9.                      this.mensaje = "Necesitas un mayor numero del siguiente material: " + mat.getNombre();
10.                     error = 1;
11.             }
12.         }
```

Diagrama de flujo:



Complejidad ciclomática:

1) **V(G) = R** = **3**
2) **V(G) = E - N + 2** = 13 - 12 + 2 = **3**
3) **V(G) = P + 1** = 2 + 1 = **3**

Caminos independientes:

Camino 1: 1-2-3-4-5-6-12 Aristas Nuevas Recorridas: 1-2, 2-3, 3-4, 4-5, 5-6, 6-12

Camino 2: 1-2-3-4-5-**6-7-8-9-10-11-6**-12 Ar. Nuevas Recorridas: 6-7, 7-8, 8-9, 9-10, 10-11, 11-6

Camino 3: 1-2-3-4-5-6-7-**8-11**-6-12 Aristas Nuevas Recorridas: 8-11

Escenarios de prueba:

Camino 1: 1-2-3-4-5-6-12

lista.size() == 0

Caso de prueba: lista.size() = 0

Resultado: “Almacenado con éxito. ” (En este caso, se asume que el juguete no necesita de ningún material independiente para ser fabricado, y se crea la tarea sin error)

Camino 2: 1-2-3-4-5-6-7-8-9-10-11-6-12

lista.size() > 0 y mat.getCantidad() > this.tarea.getCantidad() para todos los materiales de la lista

Caso de prueba: lista = {(1,x, 5, x), (2,x, 10, x)}, material = {(1,x,20, x), (2,x,50,x)}
x son valores irrelevantes en este caso, pero no deben ser nulos.

Resultado: "Almacenado con éxito."; material={(1,x,15,x), (2,x,40,x)}; error = 0

Camino 3: 1-2-3-4-5-6-7-8-11-6-12

lista.size() > 0 y mat.getCantidad() < this.tarea.getCantidad() en por lo menos un caso

Caso de prueba: lista = {(1,x, 5, x), (2,x, 10, x)}, material = {(1,x,20, x), (2,'Madera',5,x)}
x son valores irrelevantes en este caso, pero no deben ser nulos.

Resultado: "Necesitas un mayor número del siguiente material:" + 'Madera'; material={(1,x,20,x), (2,x,5,x)}; error = 1

Prueba Caja Negra

1. Identificar los valores de los datos de entrada

Sec.	Condición de Entrada	Tipo	Clases Válidas		Clases No Válidas	
			Entrada	Código	Entrada	Código
1	IdTarea	Valor	Cualquier número	CEV<01>	Un valor no numérico	CENV<01>
2	IdJuguete	Valor	Cualquier número	CEV<02>	Un valor no numérico	CENV<02>
3	Cantidad	Valor	Cualquier número	CEV<03>	Un valor no numérico	CENV<03>
4	IdPersona	Valor	Cualquier número	CEV<04>	Un valor no numérico	CENV<04>
5	tarea.getCantidad()	Rango	tarea.getCantidad() > mat.getCantidad()	CEV<05>	tarea.getCantidad <= mat.getCantidad()	CENV<05>

|| CONDICIONES DE ENTRADA ||

ID CP	Escenario	Id Tarea	Id Juguete	Cantidad	Id Persona	Cantidad superior a material	Resultado esperado
CP1	Escenario 1	V	V	V	V	V	Mensaje "Almacenado con éxito"
CP2	Escenario 2	V	V	V	V	NV	Mensaje "Necesitas un mayor número del siguiente material: + "mat. getNombre()
CP3	Escenario 3	NV	V	V	V	V	Mensaje "IdTarea no puede ser nulo"
CP4	Escenario 4	V	NV	V	V	V	Mensaje "IdJuguete no puede ser nulo"
CP5	Escenario 5	V	V	NV	V	V	Mensaje "Cantidad no puede ser nulo"
CP6	Escenario 6	V	V	V	NV	V	Mensaje "IdPersona no puede ser nulo"

2. Generar los caso de prueba.

|| CONDICIONES DE ENTRADA ||

ID CP	Clases de equivalencia	Id Tarea	Id Juguete	Cantidad	Id Persona	Cantidad superior a material	Resultado esperado
CP1	CEV<01>, CEV<02>, CEV<03>, CEV<04>, CEV<05>	V	V	V	V	V	Mensaje “Almacenado con éxito”
CP2	CEV<01>, CEV<02>, CEV<03>, CEV<04>, CNEV<05>	V	V	V	V	NV	Mensaje “Necesitas un mayor número del siguiente material: + “mat. getNombre()
CP3	CNEV<01>, CEV<02>, CEV<03>, CEV<04>, CEV<05>	NV	V	V	V	V	Mensaje “IdTarea no puede ser nulo”
CP4	CEV<01>, CNEV<02>, CEV<03>, CEV<04>, CEV<05>	V	NV	V	V	V	Mensaje “IdJuguete no puede ser nulo”
CP5	CEV<01>, CEV<02>, CNEV<03>, CEV<04>, CEV<05>	V	V	NV	V	V	Mensaje “Cantidad no puede ser nulo”
CP6	CEV<01>, CEV<02>, CEV<03>, CNEV<04>, CEV<05>	V	V	V	NV	V	Mensaje “IdPersona no puede ser nulo”

Ramas del Controlador de Versiones

En nuestro proyecto utilizamos el controlador de versiones GitHub donde íbamos subiendo las sucesivas versiones de nuestra aplicación. Iniciamos en rama '**master**' donde incluimos las primeras implementaciones y documentación.

A continuación creamos dos ramas: '**anto**' y '**jose**', donde cada integrante subía las clases en las que estaba trabajando. Al terminar dichas clases, hicimos merge de las dos ramas y continuamos escribiendo en la rama master hasta finalizar.