Informe sobre SmartFarm

Markel Ferro

24 de abril de 2021

Junto a este informe se entrega todo lo solicitado en la entrega, este detallará los detalles que requieren de explicaciones.

1. Cambios al UML

En el transcurso de programar el proyecto SmartFarm encontré que el mi diagrama UML disponía de algunos fallos o faltaban algunos métodos. Todos los cambios descritos a continuación están corregidos en la versión adjunta a esta parte del trabajo:

1. En el método *equals* de la clase *FarmAnimal* su único parámetro tenía el nombre y el tipo cambiados de orden.

2. En ese mismo método, este no devolvía nada cuando debería devolver una variable de tipo *boolean*.

3. Se ha añadido un método *toString* a la clase *Farm*, aunque debo admitir que me he tomado cierta libertad a la hora de programar este método porque este no es requerido por nadie y no venía especificado, pero al principio del enunciado se especifica que todas las clases deben tener este método sobreescrito.

4. En la clase *Farm*, cuando se hacía referencia a la temperatura y el peso se usaban *int*s, estos han sido reemplazados por *double*s en los métodos *obtainPossiblySick* y *farmAnimalsDeparture*.

5. Finalmente, en la clase *FarmAnimal* se ha añadido un getter para el atributo *arrayPos* para poder realizar mejores pruebas en JUnit.

También quiero mencionar, que se usó el UML antiguo para generar un esqueleto del proyecto, lo cual ha ayudado en gran medida a detectar estos errores. Eso sí, también ha conllevado trabajo, ya que la traducción UML Java no es perfecta.

2. Posible funcionamiento incorrecto

El trabajo no ha sido entregado siendo consciente de ningún error.

3. Problemas durante el desarrollo

Como cabe de esperar de cualquier proyecto de cierta envergadura se han presentado algunos problemas por el camino, aunque hay que admitir que el proceso de programar la *SmartFarm* no ha sido particularmente intrincado. A pesar de ello, estas son algunas de las dudas provenientes del proyecto:

1. A la hora de probar el método *register* surgían algunos problemas con la excepción *NullPointerException*, ya que desconocía la razón de que apareciese. En un momento de inspiración me di cuenta de que el animal no tenía ningún sensor adjudicado (todavía la clase *Farm* no estaba implementada), por lo tanto, estaba usando un sensor inexistente.

2. El formato *data\\file.txt* no funcionaba correctamente en Linux, por ello tras investigar online descubrí que podía usar la variable *File.separator* para la creación de archivos.

3. No tenía la teoría de lectura de archivos muy interiorizada, por lo que tuve que volver al proyecto para hacer un uso adecuado de los *FileReaders* y *Scanners*.

4. El crear datos de prueba para la clase *FarmSimulator* presento problemas al necesitar un volumen decente de datos, acabando con una combinación de datos de Excel y contraseñas comunes de Roblox.

5. Y para el final, el mayor problema que he tenido, al haber finalizado el simulador, los sensores antes de eliminar los animales + los animales eliminados me daba uno más que la cantidad de sensores que disponía al final. Tardé en darme cuenta de que entre medias se añadía al animal con código XXXX reduciendo el número de sensores en 1.

4. Casos de prueba considerados sin implementar

Algunos casos de prueba no han sido implementados, ya sea porque no aportan demasiado o por dificultades técnicas:

1. No se realiza un caso donde se haga *removeFarmAnimal* para todos los animales, ya que no aporta demasiado.

2. En el caso de *register* no se comprueba el fichero es creado correctamente a partir de la séptima lectura.

3. No se comprueba que *farmAnimalsDeparture* elimine a los animales que debe ya que resultaría prácticamente en una repetición de código dentro de JUnit.