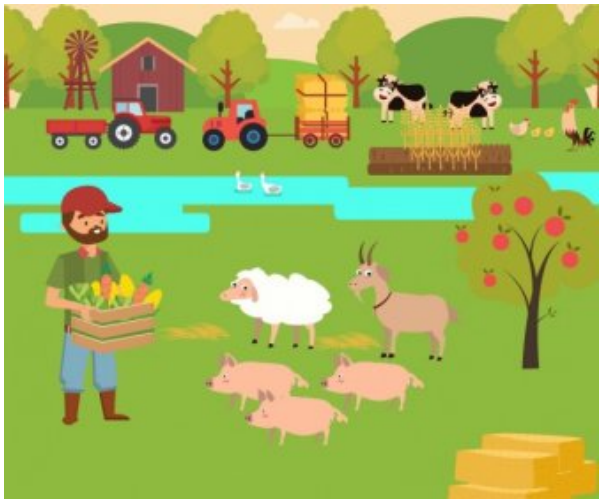
Proyecto

SmartFarm - PARTE 2 (10 %)



Objetivos:

Practicar de manera más autónoma los conceptos trabajados en la asignatura especialmente sobre jerarquías, clases abstractas, concretas e interfaces, polimorfismo, dispatching casting, instanceof, etc.

Herramientas que vamos a utilizar:

- Herramienta de diseño StarUML
- Entorno de desarrollo Eclipse: JUnit, JavaDoc y Debugger

Entrega Se realizará una entrega y se nombrará el fichero comprimido con el siguiente formato: *apellido1_apellido2_nombre*. Recordad que la entrega es individual y se subirá a eGela en la fecha indicada.

- UML solo de la jerarquía de clases derivadas de **`FarmAnimal`**: la imagen del UML en formato jpg/jpeg. Y, si se ha usado StarUML, añadir también el documento uml (o la extensión que corresponda a otra versión de StartUML).
- Documentación generada de todo el proyecto, debiendo aparecer la carpeta doc dentro del proyecto.
- Proyecto exportado (apellido1_apellido2_nombre_parte2.zip) con todas las tareas realizadas. El nombre de proyecto debe ser apellido1_apellido2_nombre_parte2
- Documento con los siguientes apartados:
  - Informe de todo lo realizado, **tiempo empleado para realizar el proyecto (si os acordáis poner el tiempo de cada una de las partes. Poner horas reales)** y los problemas que se han tenido en su desarrollo.
  - Explicaciones de aquellas partes de la entrega de las que se duda de su correcto funcionamiento o que directamente no funcionan correctamente
  - Estudio de casos de prueba considerados para la elaboración de los JUnit para los que no se ha dado el estudio de casos.
  - Captura de una ejecución completa**.**

Fecha límite de entrega: Domingo 16 de mayo a las 23:55

¡¡ATENCIÓN!!, para que se corrija la práctica se deben respetar todos los nombres indicados (ficheros, proyectos, clases etc.). Además, el proyecto no podrá tener errores de compilación y su ejecución no terminará con ningún tipo de excepción.

Caracterización de los animales de la granja

Siguiendo con el mundo de las granjas inteligentes, nos han pedido diversificar el negocio. Por ello, vamos a trabajar con granjas que tienen varios tipos de animales, lo que implica cambios en la implementación y programación de nuevas funcionalidades. En este proyecto nos vamos a centrar en los aspectos de mantenimiento de los animales (alimentación).

En la granja habrá diferentes tipos de **`FarmAnimal`**. Los animales que vamos a crear tienen una temperatura habitual de 39ºC sin embargo, las gallinas suben hasta los 42º. Se deberá definir el método `getUsualTemperature()` para poder obtener esta información. En las gallinas será necesario sobrescribe el método**.**

Además se añadirá el método `possibleSick()` que determinará (boolean) si un animal está posiblemente enfermo. Si la temperatura media registrada supera en 1,5ºC la habitual o es inferior en 1,5ºC a la habitual, se considera que está enfermo.

Los animales se siguen caracterizando como en la primera parte del proyecto, pero además debe tener información de su edad de salida de la granja (`departureAge, int`) y su coste medio anual de alimento (`annualFoodPrice,` double). Estos valores se irán trasmitiendo a través de la constructora por cada subclase según el nivel y la información que tenga. El método `departure()` indicará (`boolean`**`)`** si para el animal es el momento de salir de la granja. Para los animales en general el momento de salir de la granja es cuando ya tiene la edad estipulada de salida, pero no sucede así con los caballos y los cerdos

De forma general, en todas las clases se añadirán los getters necesarios para acceder a su información privada y se sobrescribirá el método `toString` de manera incremental para aquellas clases que tengan alguna información privada que aportar (bastará incluir dicha información separada por un espacio). También, se desea obtener en un String el camino de herencia en la jerarquía de un objeto (`inheritancePath`). Por ejemplo, si tengo un objeto de tipo oveja Latxa (Latxa l;) convenientemente inicializado y quiero obtener su camino de herencia: `System.out.println(l.inheritancePath());` debería escribir:
" Latxa < Sheep < Producer < FarmAnimal".

Para alcanzar el objetivo de mantenimiento, es necesario calcular cuál es el coste anual (double) en piensos y forrajes para los animales de la granja, `calculateAnnualCost`**.** En general devolverá el coste de su alimentación anual.

En la granja tendremos caballos, cerdos y animales productores. Los animales productores, **`Producer,`** son animales que producen algún tipo de producto como "milk", "eggs", etc. Estos animales productores se caracterizarán precisamente por lo que producen (`product`). Entre los animales productores se encuentran las gallinas ponedoras **`Chicken`** (consume 547,5€ de pienso al año), **`Cow`** (su consumo es de 492,75 € anual) y **`Sheep`** (consume 133,23 € de forraje al año). En nuestra granja además tenemos dos tipos de ovejas, ovejas lecheras que serán las ovejas **`Latxa`** y las ovejas de lana **`WoolSheep`**. La edad de salida de los Producer es su edad tope de producción. Las gallinas ponedoras tienen una edad tope de producción de 4 años, las vacas lecheras de 20 y las ovejas viven 12 años.

Por su parte los otros tipos de animales son los caballos y los cerdos. El coste en forraje para este tipo de animales depende del peso del animal. Los caballos(**`Horse`**) se caracterizarán por su tipo que indicará si su cría está destinada a exhibiciones ("show"), a paseos o aprendizaje para su monta ("ride") o para trabajo ("work"). En cualquier caso, son animales para su venta en vivo. El coste anual de su forraje es de 1027,75 € multiplicado por el 1% de su peso. La edad de salida de los caballos es 5 años. Para los cerdos (**`Pig`**), el coste del pienso es de 134,32€ por el 3,5% del peso del animal. Un cerdo estará en disposición de salir para su venta si tiene 1 año o ha alcanzado ya los 100 kg. Para los caballos y cerdos será condición

necesaria para salir de la granja (`departure`) no estar enfermos.

Además, los animales de la granja pueden ser de producción ecológica. Se considera que la granja tiene suficiente base territorial y agricultura para poder considerarse ecológica. En nuestra granja los animales productores y los cerdos pueden ser seleccionados para su producción de forma ecológica. Que una clase de animal sea ecológica no significa que todos los animales de la clase sean ecológicos. Es decir, un cerdo puede ser ecológico y otro no. En el momento de creación de un animal de posible producción ecológica, se indicará si es o no ecológico. Todos los animales ecológicos deben implementar el método sin parámetros **`isEcological`** que indicará si ese animal es o no ecológico**.** Si un animal es ecológico el coste anual en piensos y forrajes es superior. En el caso de los animales productores, el incremento es de un 20% si son ecológicos. Y en el caso de los cerdos ecológicos un 15%.

Actividades a realizar

1. **Diseña el diagrama de clases sólo para la jerarquía** de **`FarmAnimal`** (**con StarUML o a mano)** y considera las clases abstractas, concretas e interfaces surgidas. Presta atención a las características (atributos) y funcionalidades (métodos) a asociar en cada clase. Considera cómo deben ser los métodos, abstractos o concretos, y sitúalos explícitamente sólo en aquellas clases que lo precisen. Diseña todas las clases en un paquete `packFarmAnimal` que estará ubicado dentro del paquete `packFarm`**.** Este diseño te servirá de guía en la implementación.
2. Crea un nuevo proyecto a partir del proyecto de la primera parte y crea dentro del paquete **`packfarm`** el paquete **`packfarmanimal`** donde iremos creando la jerarquía de clases de los animales de la granja. Sitúa la clase **`FarmAnimal`** en el paquete **`packfarmanimal.`**
3. Modifica la clase **`Sensor`**. Los métodos **`collectValues`** y **`collect`** deben tener un parámetro temperatura cada uno. En la implementación de **`collectValues`** en la llamada al método **`collect`** se le pasará el valor del parámetro temperatura. Y finalmente, se modificará parcialmente la implementación del método **`collect`** sustituyendo la referencia a la temperatura 101.5, por el nombre del parámetro temperatura que hayáis puesto. Debe quedar:

```
temperature = temp+r.nextDouble()*2;
```

4. Completa la jerarquía de la clase **`FarmAnimal`** con toda la información dada y siguiendo el diseño UML creado. Algunas indicaciones en cuanto a la clase **`FarmAnimal.`**
   4.1. Haz que la clase implemente la interfaz **`Comparable`**. Usad el identificador del animal para realizar la ordenación.
   4.2. Actualiza la constructora con todos los parámetros que tenía anteriormente. No recibirá como parámetro el sensor, recibirá también la edad de salida de la granja y el coste anual de alimentación.
   4.3. Comprueba que tienen un método setter para añadir el sensor al animal.
   4.4. El método **`register`** se modificará en su llamada a **`collectValues`**, que le deberá pasar la temperatura habitual del animal.
   4.5. Añadir en general todos los métodos que se deriven de la descripción de la nueva granja.

5. Modifica y amplía la información de la clase **`Farm`** según se indica:
   5.1. Tiene como constante un Sensor general de identificador “ID00”, que sirve para monitorizar a los animales pequeños como las gallinas. Es un sensor único, constante y siempre está disponible.
   5.2. **`addFarmAnimal`**: A diferencia del método en la versión anterior en lugar de los tres parámetros con la información del animal, sólo se pasará un parámetro **`FarmAnimal`**. Se le añadirá un sensor disponible como lo hacíamos antes, pero sólo si el animal no es una gallina. Si el animal es una gallina no le añade un sensor de su lista, sino que le añade el sensor general “ID00”. Todas las

gallinas tienen asociado el mismo sensor ya que se trata de un sensor general que se encuentra en el exterior y contacta con cada unidad de ellas.

5.3. **obtainFarmAnimal, removeFarmAnimal,** sustituye en ambos métodos el parámetro **String** id, por **FarmAnimal** animal, y corrige todo para que funcione correctamente. Cambia el nombre de la excepción (refactor) **UnknownIdFarmAnimalException** por **UnknownFarmAnimalException.**

5.4. **farmAnimalsDeparture**: ahora este método ya no tiene parámetros y depende de las características de salida de cada animal (método departure). El sensor de las gallinas es único, siempre está disponible y no hay que recuperarlo. Sin embargo, para el resto de animales que salen de la granja si se deberá recuperar su sensor. Se devolverá una lista con los animales que salen de la granja.

5.5. **obtainPossiblySick**. Modifica el método de tal manera que ahora obtenga una lista de aquellos animales de la granja que estén enfermos (**possibleSick**).

5.6. **sortFarm**, que ordena los animales de la granja según los identificadores de sus animales.

5.7. **whoIsInFarm** que escriba por pantalla la información de todos los animales que están en la granja. Observa en el siguiente ejemplo:

```
Animals in the Farm:
CHI34 1 1.5 ID00 4 547.5 eggs true
SHE40 5 45.0 ID56 12 133.23 milk true
…
```

5.8. **calculateAnnualCost,** que obtiene el coste total anual de todos los animales de la granja.

5.9. **farmEcologicalAnimalAccount,** obtiene la cantidad de animales ecológicos que hay en la granja

6. Crea una nueva clase **FarmSimulatorHierarchy** para poder probar si todo lo anterior funciona correctamente. Para ello:

6.1. Comprueba que tienes definido un atributo de clase para el objeto que representa la granja y poder acceder a él desde cualquier parte del simulador.

6.2. Copia el método **loadSensors** hecho en la primera parte y llámalo desde el main. Imprime después cuántos sensores disponibles hay en la granja. OJO, tienes un nuevo fichero de sensores `availableSensors12.txt`.

6.3. Copia de eGela el método **initializeFarmAnimals.** Este método crea un conjunto de animales y los añade a la granja. Completa la granja con otros animales según se indica con comentarios especiales en el código: "//TODO ...". Revisa y adapta el código, ya que el orden de los parámetros de las constructoras puede no coincidir con el tuyo. Llama desde el main a ese método.

6.4. Escribe el camino de herencia de todos los animales que están en este momento en la granja.

6.5. Ordena los animales de la granja en base a su identificador**.** Muestra de nuevo los animales de la granja y comprueba si verdaderamente están en orden.

6.6. Solicita 7 veces el registro de los valores fisiológicos de todos los animales de la granja.

6.7. Comprueba y escribe cuántos y qué animales de la granja están posiblemente enfermos. Vuelve a pedir el registro de los valores fisiológicos de esos animales.

6.8. Obtén e imprime cuál es el gasto anual en piensos y forrajes de todos los animales de la granja.

6.9. Imprime cuántos animales ecológicos hay en la granja.

6.10. Imprime por pantalla cuántos sensores disponibles hay en la granja.

6.11. Obtén y elimina los animales que salgan de la granja. Imprime cuántos animales han abandonado la granja y cuáles son sus identificadores.

6.12. Vuelve a imprimir cuántos sensores disponibles hay en la granja para comprobar si se han recuperado los sensores que corresponde.