Programación 2
Curso 2016/2017

Práctica 3
La granja del tío Owen (orientado a objetos)
versión 2 (3/5/2017)

Cambios con respecto a la versión anterior:

- Se ha añadido un nuevo método público en la clase `Android` para poner a 0 el número de horas de trabajo, `resetHoursWorked`
- Se ha corregido una errata en el método `setStatus` (faltaba `void`)

Esta práctica consiste en implementar, utilizando programación orientada a objetos, una parte de la práctica 2, eliminando todo lo relacionado con ficheros.

1. Normas generales

1.1. Entrega

1. El último día para entregar esta práctica es el **miércoles 24 de mayo, hasta las 23:59**. No se admitirán entregas fuera de plazo.
2. La práctica debe tener varios ficheros llamados "`Android.cc`" y "`Androd.h`", "`Field.cc`" y "`Field.h`", "`Farm.cc`" y "`Farm.h`", "`Util.cc`" y "`Util.h`", y "`Makefile`". Para intentar evitar confusiones, se debe entregar todo el código (incluyendo el `Makefile` correspondiente) en un archivo comprimido `prog2p3.tgz`, que se debe crear con la orden:

   ```
   tar cvfz prog2p3.tgz *.cc *.h Makefile
   ```

   Para poder generar un ejecutable y realizar pruebas, en la página web de la asignatura se publicará un fichero "`uncleOwen-p3.cc`", que no debería ser modificado y que no es necesario entregar. Para corregir la práctica se utilizará ese fichero, esté o no presente en el `prog2p3.tgz`.

1.2. Detección de plagios/copias

En el Grado en Ingeniería Informática, la Programación es una materia fundamental, que se aprende programando y haciendo las prácticas de las diferentes asignaturas (y otros programas, por supuesto). Si alguien pretende aprobar las asignaturas de programación sin programar (copiando), obviamente tendrá serios problemas en otras asignaturas o cuando intente trabajar. Concretamente, en Programación 2 es muy difícil que alguien que no ha hecho las prácticas supere el examen de teoría, por lo que copiar una práctica es una de las peores decisiones que se puede tomar.

La práctica debe ser un trabajo original de la persona que la entrega; en caso de detectarse indicios de copia de una o más prácticas se tomarán las medidas disciplinarias correspondientes, informando a la dirección de la EPS por si hubiera lugar a otras medidas disciplinarias adicionales.

1.3. Otras normas

1. La práctica se debe entregar exclusivamente a través del servidor de prácticas del departamento de Lenguajes y Sistemas Informáticos, al que se puede acceder desde la página principal del departamento (`www.dlsi.ua.es`, "Entrega de prácticas") o directamente en la url `http://pracdlsi.dlsi.ua.es`.

   - No se admitirán entregas por otros medios (correo electrónico, Campus Virtual, etc.).
   - El usuario y contraseña para entregar prácticas es el mismo que se utiliza en el Campus Virtual.
   - La práctica se puede entregar varias veces, pero sólo se corregirá la última entrega (las anteriores entregas no se borran).

2. El programa debe poder ser compilado sin errores con el compilador de C++ existente en la distribución de Linux de los laboratorios de prácticas; si la práctica no se puede compilar su calificación será 0. Se recomienda compilar y probar la práctica con el autocorrector inmediatamente antes de entregarla.

3. La corrección de la práctica se hará de forma automática, por lo que es imprescindible respetar estrictamente los textos y los formatos de salida que se indican en este enunciado.

4. Al principio de todos los ficheros fuente entregados se debe incluir un comentario con el nombre y el NIF (o equivalente) de la persona que entrega la práctica, como en el siguiente ejemplo:

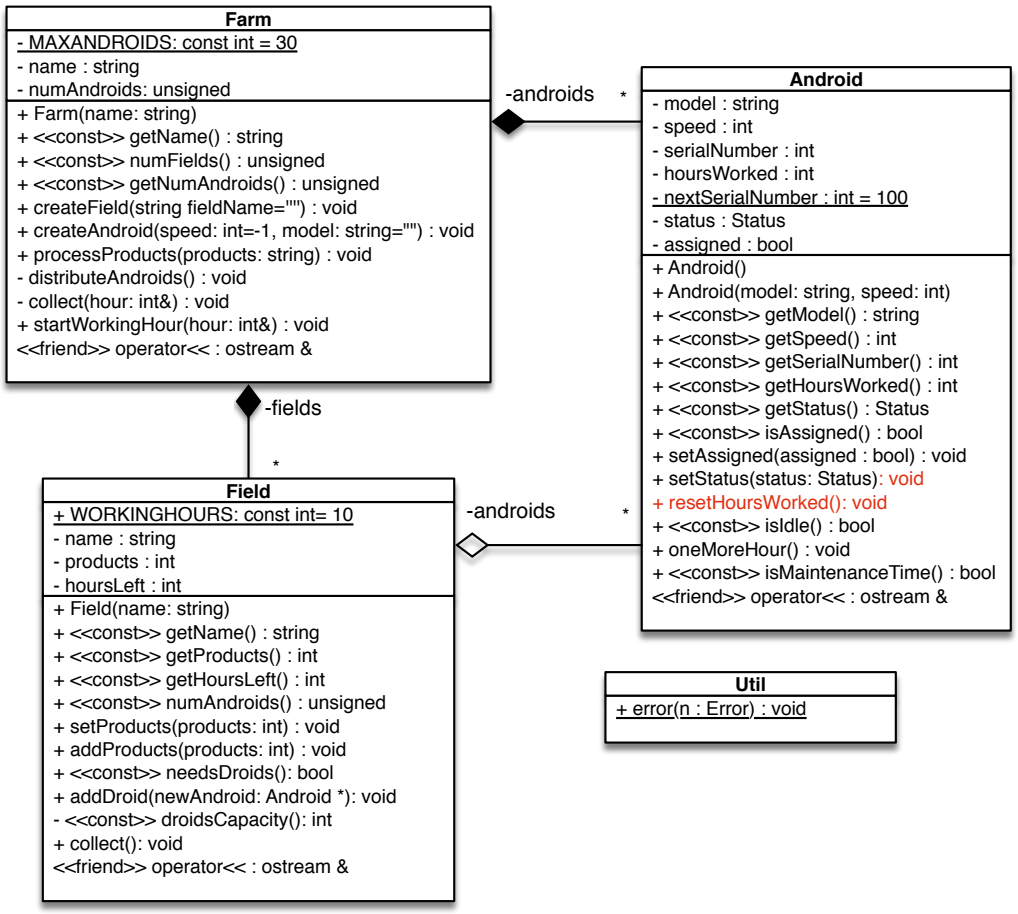
   ```
   // NIF   12345678X  GARCIA GARCIA, JUAN MANUEL
   ```

5. El cálculo de la nota de la práctica y su influencia en la nota final de la asignatura se detallan en las transparencias de la presentación de la asignatura.

2. Introducción

En esta práctica se implementarán con programación orientada a objetos las opciones del menú relacionadas con la gestión de la jornada de trabajo en la granja del tío Owen. Las opciones relacionadas con ficheros binarios y de texto no se implementarán en esta práctica.

En la siguiente página tienes un diagrama UML con las clases que es necesario implementar.



3. Clases y métodos

En el diagrama UML se muestran las clases que se deben implementar, con sus atributos y métodos. Además, tendrás que utilizar más métodos (privados siempre) y quizá atributos (también privados), según sea necesario para implementar la práctica.

A continuación se describen los métodos más importantes de cada clase; los constructores y *getters/setters* no se explican, salvo que lo que tengan que hacer no sea evidente. Algunos de ellos no es necesario utilizarlos en la práctica, pero se utilizarán en las pruebas unitarias para la corrección de la práctica.

Importante: para facilitar la corrección con pruebas unitarias, los atributos y métodos privados deben declararse con la palabra reservada `protected` en vez de con `private`.

3.1. Android

Esta clase se utiliza para almacenar los datos de un androide. Con respecto a la práctica anterior, se ha añadido un atributo más, `assigned` que indica si el androide está asignado a un campo o no (está en la granja). En esta práctica,

los androides nunca se borran del vector de androides de la granja, lo que se almacena en el campo es una referencia (puntero) al androide, y la forma de saber si está en un campo o en la granja es mediante este atributo. Para acceder a este atributo puedes usar los métodos `isAssigned` o `setAssigned`.

Los métodos más relevantes de esta clase son:

`Android` : constructor por defecto, sin parámetros. Debe inicializar el modelo a una cadena vacía y la velocidad a -1. Es necesario que exista para declarar el vector de androides de la granja que ya no es un `vector<>` y pasa a ser un *array* de tamaño fijo, como se explica más adelante) y no debe invocarse nunca.

`Android` : constructor para crear un androide con los datos indicados en los argumentos. Si el modelo que se pasa como parámetro no es correcto, o bien si la velocidad es incorrecta, se debe lanzar la excepción `invalid_argument`, pasando como parámetro de la excepción la cadena `"wrong android model"` o `"wrong speed"`, respectivamente. Si ambos datos son incorrectos solamente se lanzará la excepción de modelo incorrecto. Para poder usar la excepción `invalid_argument` es necesario incluir el fichero `stdexcept`, tanto en este fichero como en el que se captura la excepción:

```
#include <stdexcept>
```

Es posible que durante la ejecución normal de la práctica no se generen las dos posibles excepciones, pero el constructor debe implementarlas por si se usa en las pruebas unitarias.

`isIdle` : devuelve `true` si el estado del androide es `ST_IDLE`, o `false` en caso contrario.

`resetHoursWorked` : pone a 0 el número de horas trabajadas del androide.

`isMaintenanceTime` : devuelve `true` si el androide ha alcanzado el número de horas de trabajo indicadas para realizar el mantenimiento según su modelo, o `false` en caso contrario.

`oneMoreHour` : incrementa en 1 las horas de trabajo del androide.

`operator<<` : operador de salida, imprimirá los datos de un androide como la función `printAndroid` de las prácticas anteriores, pero añadiendo el atributo `assigned`, como en este ejemplo:

```
[x-75 sn=100 s=100 hw=0 st=1 a=0]
```

Notas sobre implementación:

- La definición del tipo enumerado `Status` debe ir en el fichero `Android.h`, puesto que se usa en los demás ficheros del programa.
- Las constantes `NUMDROIDMODELS` y `ANDROIDDATA` deberían aparecer en el fichero `Android.cc` antes de los métodos de la clase, ya que solamente se utilizan en esta clase.
- El atributo de clase `nextSerialNumber` se utiliza como en las prácticas anteriores para asignar a cada androide un número de serie único. También como en las prácticas anteriores, debe inicializarse a 100.

3.2. Field

Esta clase implementa los campos de cultivo de la granja. Como en la práctica anterior debe almacenar los androides utilizando un vector.Al tratarse de una relación de agregación, el vector almacenará punteros a Android, en vez de objetos de la clase Android. Los métodos más importantes son:

`Field` : constructor que inicializa nombre, productos y horas restantes.

`setProducts` : *setter* para asignar el valor del atributo `products`. Si después de asignar ese valor el atributo `products` es menor que 0, debe ponerse a 0.

`addProducts` : añade productos al campo. Si después de añadir los productos el atributo `products` es menor que 0, debe ponerse a 0.

`needsDroids` : devuelve `true` si el campo necesita más androides para recolectar los productos que le quedan, y `false` si con los androides que tiene es suficiente. Si no tiene productos debe devolver `false`.

`addDroid` : añade un nuevo androide al vector de androides del campo, cambiando antes su estado a `ST_WORKING`, y su atributo `assigned`.

`droidsCapacity` : devuelve el número de productos que pueden recolectar los androides del campo en una hora de trabajo. Es un método privado que sólo puede usarse dentro de la clase.

`collect` : recolecta los productos del campo en una hora de trabajo, como en la práctica 2. Actualiza las horas trabajadas de los androides, las horas restantes del campo, envía a mantenimiento los androides que lo necesiten, y si ha acabado la jornada de trabajo devuelve los restantes androides a la granja. La forma de "devolver" los androides a la granja es diferente con respecto a la práctica anterior: en esta práctica para devolver un androide simplemente tiene que modificar el atributo `assigned` del androide (usando el *setter*) al mismo tiempo que modifica el estado, y borrarlo de su vector de androides.

`operator<<` : muestra los datos del campo siguiendo el mismo formato que en la práctica anterior (similar a la función `printField`).

3.3. Farm

En esta clase se implementan los métodos relacionados con la granja. Con respecto a las prácticas anteriores, el vector de androides se implementará en esta práctica usando un *array* convencional:

```
Android androids[MAXANDROIDS];
```

Por tanto, habrá espacio para `MAXANDROIDS` androides como mucho; para controlar cuántos androides se han introducido en la granja se debe usar el atributo `numAndroids`, que inicialmente valdrá 0 y se irá incrementando cuando se añadan androides a la granja. Como se ha explicado antes, los androides no se mueven del vector de la granja, simplemente se asignan a los campos, que almacenan un puntero al androide en su vector de androides (que sería un `vector<Android *>`).

`Farm` : crea una granja con su nombre, e inicializa el número de androides.

`createField` : se debe comprobar si el parámetro `name` es una cadena vacía, en cuyo caso debe pedir por pantalla el nombre del campo; si no es la cadena vacía, se toma ese parámetro como nombre del campo. A continuación se intentará añadir el campo a la granja, si el nombre no está repetido. En caso contrario emite el error `ERR_NAME`, como en la práctica anterior.

`createAndroid` : si se ha alcanzado el número máximo de androides, se debe emitir el error `ERR_MAXANDROIDS` y no hacer nada más. Si el parámetro `speed` tiene el valor por defecto (-1), pide por pantalla la velocidad del androide, y si es correcta pide el modelo y en caso contrario emite el error `ERR_WRONG_SPEED`. Si no se ha producido ese error (porque la velocidad introducida es correcta o bien el parámetro `speed` no es -1), intenta crear un androide nuevo con los parámetros que se le pasan y añadirlo a la granja. Si al crear el androide se produce una excepción debe capturarla y emitir el mensaje de error oportuno (`ERR_WRONG_MODEL` o `ERR_WRONG_SPEED`)

`processProducts` : procesa una cadena con datos de campos y productos, con el mismo formato de la práctica anterior, y añade los datos (si son correctos) a la granja. En caso de producirse algún error se emite el mensaje de error adecuado, como en la práctica anterior.

`distributeAndroids` : distribuye los androides por los distintos campos de la granja, siguiendo el mismo algoritmo que en la práctica anterior (método privado). Los androides se añaden al campo usando el método `addDroid` pasándole como parámetro un puntero a un androide del vector de androides de la granja.

`collect` : simula la recolección de los productos de los campos de la granja durante una hora de trabajo, como en la práctica anterior. Al final de la jornada simula la realización del mantenimiento, como en la práctica anterior (método privado).

`startWorkingHour` : simula una hora de trabajo, como en la práctica anterior. Si es la primera hora de trabajo de la jornada pide por pantalla los productos y los procesa, también como en la práctica anterior.

`operator<<` : operador de salida, muestra los datos de la granja siguiendo el mismo formato que en la práctica anterior (similar a la función `printFarm`).

3.4. Util

Esta clase implementa el método para emitir mensajes de error:

`error` : se usa para emitir mensajes de error. El tipo enumerado con los posibles errores debe estar en el fichero `Util.h`

En esta práctica hay un nuevo mensaje de error, `ERR_MAXANDROIDS`, por lo que habría que añadir ese valor al `enum Error`, y un `case` más al método error (con respecto a la práctica anterior):

```
case ERR_MAXANDROIDS:
  cout << "Error, no more androids allowed in the farm" << endl;
  break;
```

4. Programa principal

El programa principal estará en el fichero "`uncleOwen-p3.cc`" que se publicará en la web de la asignatura. Este fichero no debería modificarse y no puede entregarse con la práctica (la práctica se compilará con la versión publicada en la web antes de corregirla).

5. Implementación

Para implementar la práctica debes tener en cuenta, además de todo lo comentado anteriormente:

1. Es probable que puedas aprovechar código de las prácticas 1 y 2, y es buena idea que lo hagas, pero tampoco intentes aprovecharlo todo; a veces se tarda menos en escribir el código de nuevo que en intentar adaptar un código ya existente.
2. Como en las prácticas anteriores, el resto de decisiones en la implementación de la práctica quedan a tu criterio.