Diseño Software – Grado en Ingeniería de Robótica Software

Práctica 4: Programación orientada a objetos (I)

La multinacional Carsarus-Brou Inc. está diseñando el "parking del futuro", una idea revolucionaria que pretende utilizar superficies en desuso en las ciudades, aprovechando al 100% el espacio disponible para aparcar vehículos por medio de un sofisticado sistema que incluye una grúa robotizada, de forma que cuando un vehículo precisa ser estacionado, el usuario lo coloca en un espacio único reservado para la entrega y sale del vehículo. A continuación, la grúa robotizada lo coge en volandas y lo eleva a modo de ascensor, llevándolo a su planta correspondiente y dejándolo perfectamente estacionado en el hueco asignado. De esta forma se evita la necesidad de dejar espacio libre en cada planta para que los vehículos maniobren (entren/salgan/aparquen/besen columnas), ya que cuando el usuario vuelve para recoger el vehículo la grúa realiza la operación inversa y deposita el vehículo en un espacio único reservado para la recogida.

La empresa pretende desarrollar una aplicación para gestionar los espacios libres/ocupados del parking. Tras hablar con Mr. Quattropillars Objetoriented, reputado experto en diseño software, éste les ha aconsejado encarecidamente que utilicen la programación orientada a objetos, para poder gestionar todos los parkings que vayan instalando de forma sencilla y eficiente dentro de una única aplicación (los ejecutivos de Carsarus no han entendido muy bien de lo que les estaba hablando, pero al fin y al cabo Quattropillars es el experto…).

Las características solicitadas para la aplicación han sido las siguientes:

- Podrá gestionar un número ilimitado de parkings (las expectativas de crecimiento de la empresa son muy altas).
- Cada parking puede tener un número diferente de plantas (en función de la superficie sobre la que se construya), pero por restricciones legales y para facilitar el diseño de la grúa el número de plantas estará entre 1 y 4
- Se podrá consultar el estado de ocupación de cada planta en cada parking.
- El sistema asignará un identificador secuencial de 1 letra y 3 dígitos a cada vehículo que queda estacionado en el parking. La letra será asignada dependiendo del tipo de vehículo (A = Autobús, M = Motocicleta, F = Furgoneta, T = turismo). Para el número de 3 dígitos empezará asignando el 100, después el 101, etc…, hasta llegar al 999, asignando de nuevo el 100 al siguiente vehículo.
- El espacio disponible en cada planta será un área rectangular cuyas dimensiones se miden en "bloques básicos" (por ejemplo, una planta podría alojar 5x7 bloques básicos). Por cuestiones de diseño el tamaño máximo de una planta es 6x10. El espacio que ocupa cada tipo de vehículo estacionado, medido en bloques básicos es el siguiente:
    1. Motocicleta: 1 bloque básico

2. Turismo: 2 bloques básicos
3. Furgoneta: 3 bloques básicos
4. Autobús: 4 bloques básicos

- Debido a limitaciones en el diseño de la grúa (no tiene capacidad para girar vehículos en el aire), todos los vehículos deben ser colocados en una planta del parking en posición horizontal, mirando la planta del parking desde arriba

- El algoritmo que debe utilizar la aplicación para asignar los bloques básicos correspondientes a un vehículo que desee estacionar será el siguiente:

    1. Escoger la planta del parking con menor ocupación (mayor número de bloques básicos libres). En caso de empate, se escogerá la planta más baja posible

    2. En la planta elegida, buscar los bloques básicos contiguos libres necesarios (dependiendo del tipo de vehículo) siempre en horizontal, empezando por la última fila (la fila situada más abajo, mirando la planta del parking desde arriba) y acabando en la primera fila (la fila situada más arriba). Cada fila se recorre buscando a partir de la primera columna (la columna situada más a la izquierda, mirando la planta del parking desde arriba) y acabando en la última (la de más a la derecha).

        Ejemplo de visualización de una planta 4x8 (mirando desde arriba):

```
Primera fila →   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
Segunda fila →   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
Tercera fila →   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
Última fila  →   XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX
                  |                                     |
                  |                    Última columna <-|
                  |-> Primera columna
```

    3. Si no existen los bloques básicos libres necesarios en la planta elegida, el programa debe informar de que no es posible estacionar el vehículo en el parking, y no debe asignar ningún identificador al vehículo.

La empresa Carsarus-Brou consiguió que Mr. Objetoriented esbozara el diseño de las distintas clases que componen la aplicación, obteniendo varios ficheros .h a cambio de un 5J. El señor Objetoriented quedó tan encantado con el regalo que les envió además un pequeño programa de pruebas junto con la salida esperada como ejemplo. Después de terminarse el 5J, decidió también implementar una de las clases y la añadió al envío.

En esta práctica debemos completar el desarrollo de la aplicación descrita anteriormente. El código de partida son los ficheros de cabecera y el pequeño programa de pruebas proporcionados por el susodicho Quattropillars. Este código puede ser modificado/ampliado/no utilizado en absoluto, a voluntad.

Además de funcionar correctamente, el programa debe cumplir las siguientes normas de calidad con el objetivo de que el código final sea lo más modular/extensible/mantenible/reutilizable posible:

1. Dentro de la función “main()” sólo puede haber declaraciones de y asignaciones a variables locales y llamadas a otras funciones
2. El fichero “main.cpp” no puede contener ninguna otra función aparte de la función “main()”
3. Las distintas funciones/métodos a desarrollar deben contener un máximo de 30 líneas de código (sin contar los comentarios)
4. Las variables globales están prohibidas
5. Las estructuras de datos como tales están prohibidas, el programa debe ser diseñado usando exclusivamente orientación a objetos
6. El idioma a utilizar tanto en los mensajes de salida hacia el usuario como en los comentarios del código debe ser único
7. Se prohíbe el uso de las sentencias “continue” y “break” en los bucles del programa
8. Cada función debe tener un único punto de retorno (una sola aparición de la sentencia “return”)
9. Se deben definir las constantes necesarias para aumentar la legibilidad del código y reducir el tiempo de desarrollo, ante posibles cambios de valor de las mismas