Universitat d'Alacant
Universidad de Alicante
Departament de Llenguatges i Sistemes Informàtics
Departamento de Lenguajes y Sistemas Informáticos

Práctica 2
Programación dirigida por eventos

Objetivos

- Aprender a crear una aplicación que haga uso de los conceptos vistos en el tema *Programación Dirigida por Eventos*.
- Aprender a diferenciar entre los conceptos de evento y manejador (*callback)*.
- Continuar aprendiendo a usar Git.

Entrega

La entrega de esta práctica consiste en el directorio de la solución `hada-p2`, junto con todo su contenido, comprimido en un fichero llamado `hada-p2.zip`.

Lugar y fecha de entrega: La entrega se realizará en http://pracdlsi.dlsi.ua.es. No se admitirá ningún otro método de entrega.

Fecha límite: 08/03/2020

Requisitos técnicos

Requisitos que tiene que cumplir este trabajo práctico para ser evaluado (si no se cumple alguno de los requisitos la calificación será **cero**):

- La solución entregada no contiene archivos compilados, por lo que, **antes de comprimir, debes limpiar la solución (Compilar > Limpiar solución).**
- El archivo entregado se llama `hada-p2.zip` (**todo en minúsculas**).
- Al descomprimir el archivo `hada-p2.zip` se crea un directorio de nombre `hada-p2` (**todo en minúsculas**).
- Dentro del directorio `hada-p2` hay un archivo de nombre `hada-p2.sln`.
- Dentro del directorio `hada-p2` (*directorio de la solución*) hay dos directorios: `hada-p2` y `.git`.
- El directorio `hada-p2/hada-p2` contiene los archivos con el código de la práctica y se llaman como se indica en el enunciado (**respetando en todo caso** el uso de mayúsculas y minúsculas).
- Los nombres de espacios de nombres, clases y métodos implementados, así como sus argumentos, se llaman como se indica en el enunciado (**respetando en todo caso** el uso de mayúsculas y minúsculas).

- Los mensajes producidos siguen el formato especificado en el enunciado (**respetando en todo caso** el uso de mayúsculas y minúsculas).
- Se han realizado **al menos 3 commits** y en cada uno de ellos los cambios o adiciones realizados **demuestran avances** en el desarrollo de la práctica. Cada uno de estos commits deberá contener, tu DNI/NIE en el comentario del commit y un mensaje explicativo.
- El código debe compilar y ejecutarse sin problemas en los ordenadores del laboratorio de prácticas de Hada.

Guía de evaluación

- La creación de las clases `Vehiculo` y `Autovia`, así como de todos y cada uno de sus componentes (constructores, propiedades, métodos, etc…) trabajando de forma correcta supondrá hasta el 90% de la nota.
- Los distintos commits realizados a lo largo de la creación de la práctica supondrán hasta el 10% de la nota.

Descripción

En esta segunda práctica vamos a llevar a cabo un ejercicio sencillo de programación dirigida por eventos. Para ello implementaremos una pequeña aplicación que simula el recorrido de varios vehículos por una autovía. Para cada vehículo conoceremos el porcentaje de combustible que le queda en el depósito, la velocidad a la que circula y la temperatura del motor. Cuando el depósito del coche esté próximo a vaciarse saltará un evento informando de esto, lo mismo sucederá cuando la temperatura del motor supere la temperatura máxima permitida y cuando el vehículo exceda la velocidad máxima permitida.

Sigue los pasos indicados, **respeta el uso de mayúsculas y minúsculas** así como el **nombre** de las **carpetas, archivos, espacios de nombres, clases, métodos y argumentos** y el **formato** de **mensajes** que se te indique.

Crea un proyecto de C# del tipo "Aplicación de Consola (.NET framework)" llamado **hada-p2** dentro de una solución de nombre **hada-p2** y un repositorio git ubicado en la carpeta de la solución y **ve haciendo commits** en él de todo lo que vayas haciendo. En el **comentario de cada commit** debes incluir tu **DNI/NIE** además de un mensaje explicativo (ej: git commit -m "*12345678A Añadido método M1*").

Clases empleadas

Clase `Vehiculo` (fichero `Vehiculo.cs`)

Esta clase pertenece al espacio de nombres **`Hada`** ***(esto es: el namespace del proyecto debe llamarse Hada)*** y hará uso de las siguientes propiedades **públicas** y **estáticas** *(en este caso son propiedades implementadas automáticamente*) para establecer, respectivamente, la velocidad máxima permitida, la temperatura máxima permitida, el umbral mínimo de combustible y el generador de número aleatorios a usar por los métodos que así lo requieran:

- `int maxVelocidad`
- `int maxTemperatura`
- `int minCombustible`
- `Random rand`
    - **Importante**, debéis usarlo de este modo:
      `rand.Next(minValue,maxValue + 1)`

El *getter* de `rand` deberá ser **privado**.

Además hará uso de la siguiente propiedad **pública**, pero cuyo *setter* deberá ser **privado**, para guardar el nombre del vehículo:

- `string nombre`

También declarará las siguientes propiedades **privadas** *(en este caso son propiedades con campos de respaldo*):

- `int velocidad`: guardará la velocidad que lleva el vehículo. En el *setter* deberá comprobar si el valor a asignar es mayor que `maxVelocidad` y en caso afirmativo disparará el evento `velocidadMaximaExcedida` (véase más abajo). Si el valor a asignar es menor que cero, se asignará el valor cero.
- `int temperatura`: guardará la temperatura del motor del vehículo. En el *setter* deberá comprobar si el valor a asignar es mayor que `maxTemperatura` y en caso afirmativo disparará el evento `temperaturaMaximaExcedida` (véase más abajo).
- `int combustible`: guardará el porcentaje de combustible que tiene el vehículo en su depósito. En el *setter* deberá comprobar si el valor a asignar es menor que `minCombustible` y en caso afirmativo disparará el evento `combustibleMinimoExcedido` (véase más abajo). Como se trata de un porcentaje, si el valor a asignar es menor que cero, se asignará el valor cero, y si es mayor que 100, se asignará el valor 100.

Los métodos que deberá implementar esta clase son los siguientes métodos (todos ellos **públicos**):

- `Vehiculo (string nombre, int velocidad, int temperatura, int combustible)`: Constructor de la clase. Establece el nombre y la velocidad, temperatura y porcentaje de combustible iniciales.
- `void incVelocidad()`: Incrementará la velocidad del vehículo en un valor aleatorio entre 1 y 7 km/h. El valor aleatorio lo generará usando el método pertinente de la propiedad de clase (estática) `rand`.
- `void incTemperatura()`: Incrementará la temperatura del motor del vehículo en un valor aleatorio entre 1 y 5 ºC. El valor aleatorio lo generará usando el método pertinente de la propiedad de clase (estática) `rand`.
- `void decCombustible()`: Decrementará el porcentaje de combustible restante en un valor aleatorio entre 1 y 5 puntos porcentuales. El valor aleatorio lo generará usando el método pertinente de la propiedad de clase (estática) `rand`.
- `bool todoOk()`: El estado de un vehículo es correcto si la temperatura de su motor es menor o igual que la temperatura máxima permitida (`maxTemperatura`) y su nivel de combustible (el porcentaje que le queda en el depósito) es mayor o igual que el límite permitido (`minCombustible`).
- `void mover()`: Si el estado del coche es correcto (`todoOk`), moverá el coche, lo cual implica incrementar la velocidad y la temperatura, y decrementar la cantidad de combustible (**en este orden**). Para esto hará uso de los métodos definidos anteriormente.
- `string ToString()`: Este método sobreescribe la implementación del mismo que ofrece la clase `Object`. Para esto deberéis usar la palabra reservada `override`. Devuelve una cadena con el mismo formato que el que se aprecia en el siguiente ejemplo:
  [nombre] Velocidad: 128 km/h; Temperatura: 97 ºC; Combustible: 2 %; Ok: False

Por último esta clase declarará los siguientes **eventos públicos**:

- `event EventHandler<VelocidadMaximaExcedidaArgs> velocidadMaximaExcedida`: evento que se produce cuando se supera la velocidad máxima establecida (`maxVelocidad`).
- `event EventHandler<TemperaturaMaximaExcedidaArgs> temperaturaMaximaExcedida`: evento que se produce cuando se supera la temperatura máxima establecida (`maxTemperatura`).
- `event EventHandler<CombustibleMinimoExcedidoArgs> combustibleMinimoExcedido`: evento que se produce cuando el porcentaje de combustible es menor que el mínimo establecido (`minCombustible`).

Hay que tener en cuenta que también se deberán crear las clases que representan los argumentos de estos tipos de eventos: `VelocidadMaximaExcedidaArgs`, `TemperaturaMaximaExcedidaArgs` y

`CombustibleMinimoExcedidoArgs`. Los constructores de estas clases recibirán como argumento el valor (de tipo `int`) que ha provocado que se lance el evento y utilizarán una propiedad para permitir el acceso a éste. El nombre de esta propiedad será `velocidad` para `VelocidadMaximaExcedidaArgs`, `temperatura` para `TemperaturaMaximaExcedidaArgs` y `combustible` para `CombustibleMinimoExcedidoArgs`. Se deja como ejercicio para el alumno determinar cómo implementar estas clases. **Importante: Estas clases se crearán dentro del fichero Vehiculo.cs** (en un mismo fichero podemos tener varias clases).

Clase `Autovia` (fichero `Autovia.cs`)

Esta clase también pertenece al espacio de nombres **`Hada`** y será la encargada de crear los vehículos y moverlos, y de conectar los **eventos** que se disparan desde la clase `Vehiculo` con los manejadores que deberán tratarlos.

Esta clase implementará los siguientes métodos **públicos**:

- `Autovia(int nc)`: Constructor de la clase. Crea una autovía con capacidad para `nc` vehículos. Crea `nc` vehículos, todos con un nombre distinto a elección del alumno y un valor de temperatura, velocidad y combustible de 50. Además de crear los vehículos deberá conectar los manejadores (véase más abajo) de los eventos para así poder tratarlos cuando se produzcan.
- `bool moverCoches()`: mueve todos los coches de la autovía cuyo estado sea correcto (`todoOk`). Devuelve `true` si al menos uno de los vehículos pudo ser movido y `false` en caso contrario.
- `void moverCochesEnBucle()`: Mueve todos los coches mientras queden coches cuyo estado sea correcto.
- `List<Vehiculo> getCochesExcedenLimiteVelocidad()`: Devuelve una lista con los vehículos que han excedido el límite de velocidad.
- `List<Vehiculo> getCochesExcedenLimiteTemperatura()`: Devuelve una lista con los vehículos que han excedido el límite de temperatura.
- `List<Vehiculo> getCochesExcedenMinimoCombustible()`: Devuelve una lista con los vehículos con menos combustible que el mínimo de combustible establecido.
- `string ToString()`: Este método sobreescribe la implementación del mismo que ofrece la clase `Object`. Para esto deberéis usar la palabra reservada `override`. Devuelve una cadena formada por varias líneas, la primera de ellas contendrá información general sobre el estado de los vehículos en la autovía y el resto serán los datos concretos de cada uno de los vehículos. El siguiente ejemplo ilustra el formato de esta cadena:
  [AUTOVÍA] Exceso velocidad: 0; Exceso temperatura: 2; Déficit combustible: 1

```
[coche_0] Velocidad: 103 km/h; Temperatura: 96 ºC; Combustible: 4 %; Ok: False
[coche_1] Velocidad: 107 km/h; Temperatura: 95 ºC; Combustible: 0 %; Ok: False
[coche_2] Velocidad: 115 km/h; Temperatura: 94 ºC; Combustible: 4 %; Ok: True
[coche_3] Velocidad: 97 km/h; Temperatura: 98 ºC; Combustible: 5 %; Ok: False
[coche_4] Velocidad: 111 km/h; Temperatura: 94 ºC; Combustible: 7 %; Ok: True
```

Además la clase `Autovia` implementará los siguientes métodos **privados** cuya función es **manejar** el **evento** correspondiente cuando se produzca:

- `void cuandoVelocidadMaximaExcedida(...)`: Cuando sea invocado imprimirá por pantalla tres líneas como éstas:
  ```
  ¡¡Velocidad máxima excedida!!
  Vehículo: coche_4
  Velocidad: 131 km/h
  ```
- `void cuandoTemperaturaMaximaExcedida(...)`: Cuando sea invocado imprimirá por pantalla tres líneas como éstas:
  ```
  ¡¡Temperatura máxima excedida!!
  Vehículo: coche_2
  Temperatura: 97 ºC
  ```
- `void cuandoCombustibleMinimoExcedido(...)`: Cuando sea invocado imprimirá por pantalla tres líneas como éstas:
  ```
  ¡¡Combustible mínimo excedido!!
  Vehículo: coche_1
  Combustible: 0 %
  ```

En cada caso se indica el nombre del vehículo que ha lanzado el evento y el valor de velocidad, temperatura o combustible, según sea el caso, que lo ha provocado. Estos métodos deberán además gestionar las estructuras de datos adecuadas que permitan a los métodos `getCochesExcedenLimiteVelocidad`, `getCochesExcedenLimiteTemperatura` y `getCochesExcedenMinimoCombustible` devolver los datos indicados más arriba para cada uno de ellos.

Se deja como ejercicio para el alumno determinar qué parámetro o parámetros tienen que tener los métodos utilizados para el manejo de eventos.