Hada Práctica 2: Programación dirigida por eventos

Departamento de Lenguajes y Sistemas Informáticos Universidad de Alicante

Objetivos de la práctica.

- 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 *señal* y *callback*.
- Hacer uso de clases, interfaces, herencia y paso de mensajes en C#.
- Continuar aprendiendo a usar git.

Programación dirigida por eventos

- En esta segunda práctica vamos a llevar a cabo un ejercicio sencillo de programación dirigida por eventos.
- 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 una solución de nombre **hada-p2** y un repositorio **git** en la carpeta de la solución y ve haciendo commits en él de todo lo que vayas haciendo. **Incluye un comentario con tu DNI/NIE en cada commit**.
- Al final del documento se indican las condiciones de entrega, los requisitos técnicos que debe cumplir la entrega para ser evaluada y una guía de evaluación de esta práctica.

Clases empleadas

- Usaremos la clase **`FlipFlop`** del ejemplo de la vivienda domótica como *clase base* de dos *tipos de sensores*, cada uno de ellos, a su vez, representado por una clase:
  - **`ProximitySensor`**: Generará un evento de proximidad cuando el sensor detecte que estamos *cerca* de un obstáculo.
  - **`LightSensor`**: Generará eventos de *poca* o *demasiada* luz en función de la luz ambiente detectada.
- Estas dos clases serán utilizadas por una tercera que representará un **`Robot`**. Un **`Robot`** hará uso de un *sensor de luz* y de un *sensor de proximidad*.
- Todas las clases empleadas en esta práctica pertenecen al *espacio de nombres* **`Hada`**. El código de cada clase estará en un archivo llamado como la clase (todo en minúsculas) y con extensión '.cs', p.e.: **`robot.cs`** .

Clase FlipFlop

- Ya la conocemos, se trata de la misma clase empleada en el ejemplo de la *vivienda domótica*.
- Representa un *biestable*, algo que puede estar *encendido* o *apagado*, *abierto* o *cerrado*, etc.
- Es una clase abstracta que puede generar el **evento** `statusChanged` para indicar que el *biestable* ha cambiado de estado.

Clase Robot - I

- Deriva de `FlipFlop` y contará con un sensor de luz y uno de proximidad propios.
- El sensor de luz tendrá unos valores mínimo y máximo de 20 y 70. respectivamente. Su valor inicial será de 50.
- El de proximidad tendrá un valor mínimo de 20 y su valor inicial será de 100.
- Los nombres de los sensores pueden ser cualquier cadena, excepto la cadena vacía.

Clase Robot - II

- Tendrá los siguientes métodos públicos:
  - **`public Robot (string name)`**: Constructor.
    - **`name`**: Es el nombre del robot.
  - **`public void on ()`**: Pone en marcha el robot.
  - **`public void off ()`**: Para el robot.
  - **`public void createLowLightCondition ()`**: Hace que el sensor de luz del robot genere un evento de *poca* luz.
  - **`public void createHighLightCondition ()`**: Hace que el sensor de luz del robot genere un evento de *demasiada* luz.
  - **`public void createProximityCondition ()`**: Hace que el sensor de proximidad del robot genere un evento de *proximidad*.

Clase Robot - III

- Para atender a los eventos que pueden generar sus dos sensores, dispondrá de estos métodos privados:
  - **`private void onLowLight (...)`**: Debes decidir qué parámetro o parámetros tiene que tener. Cuando sea invocado imprimirá por pantalla dos líneas como éstas:

    ```
    low-light alert!!
    light level: 5  /* es el valor que ha hecho generar el evento, puede ser otro */
    ```

  - **`private void onHighLight (...)`**: Debes decidir qué parámetro o parámetros tiene que tener. Cuando sea invocado imprimirá por pantalla dos líneas como éstas:

    ```
    high-light alert!!
    light level: 5  /* es el valor que ha hecho generar el evento, puede ser otro */
    ```

  - **`private void onProximity (...)`**: Debes decidir qué parámetro o parámetros tiene que tener. Cuando sea invocado imprimirá por pantalla dos líneas como éstas:

    ```
    proximity alert!!
    proximity level: 5  /* es el valor que ha hecho generar el evento, puede ser otro */
    ```

Clase LightSensor - I

- Deriva de `FlipFlop`.
- Tendrá los siguientes métodos públicos:
  - **`public LightSensor (string n, float l, float minl, float maxl)`**: Constructor.
    - **`n`**: Es el nombre del sensor.
    - **`l`**: Es el nivel de luz actual.
    - **`minl`**: Es el nivel mínimo de luz. Si el sensor detecta un nivel de luz menor que éste, entonces genera un evento de tipo `lowLightCondition`.
    - **`maxl`**: Es el nivel máximo de luz. Si el sensor detecta un nivel de luz superior a éste, entonces genera un evento de tipo `highLightCondition`.
  - **`public float level`**: Es una propiedad que permitirá consultar y modificar el nivel de luz de este sensor.

Clase LightSensor - II

- Tendrá además los siguientes eventos:
  - **`public event EventHandler<LowLightArgs> lowLightCondition`**: El evento que se produce cuando se detecta poca luz.
  - **`public event EventHandler<HighLightArgs> highLightCondition`**: El evento que se produce cuando se detecta demasiada luz.
- Ten en cuenta que también deberás crear las clases que representan los argumentos de estos tipos de eventos:
  - **`LowLightArgs`**
  - **`HighLightArgs`**
- Piensa qué constructor necesitas para estas clases y qué método(s) o propiedades necesitarás para acceder a su parte privada.

Clase ProximitySensor - I

- Deriva de `FlipFlop`.
- Tendrá los siguientes métodos públicos:
  - **`public ProximitySensor (string n, float p, float minp)`**: Constructor.
    - **`n`**: Es el nombre del sensor.
    - **`p`**: Es el nivel de proximidad actual.
    - **`minp`**: Es el nivel mínimo de proximidad. Si el sensor detecta un nivel de proximidad menor que éste, entonces genera un evento de tipo `proximityCondition`.
  - **`public float proximity`**: Es una propiedad que permitirá consultar y modificar el nivel de proximidad de este sensor.

Clase ProximitySensor - II

- Tendrá además el siguiente evento:
  - **`public event EventHandler<ProximityArgs> proximityCondition`**: El evento que se produce cuando se detecta que estamos cerca de un obstáculo.
- Ten en cuenta que también deberás crear la clase que representa los argumentos de este tipo de eventos:
  - **`ProximityArgs`**
- Piensa qué constructor necesitas para esta clase y qué método(s) o propiedades necesitarás para acceder a su parte privada.

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.tgz`.
  - Este archivo lo puedes crear así en el terminal: `tar cfz hada-p2.tgz hada-p2`
- **Lugar y fecha de entrega**: La entrega se realizará en http://pracdlsi.dlsi.ua.es en las fechas allí publicadas.
- **No se admitirá ningún otro método de entrega**.

Requisitos técnicos I.

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

- El archivo entregado se llama `hada-p2.tgz` (todo en minúsculas).
- Al descomprimir el archivo `hada-p2.tgz` 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` 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).

Requisitos técnicos II.

- 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, al menos, tu DNI/NIE en el comentario del commit.

Guía de evaluación.

- La creación de cada una de las clases `Robot`, `LightSensor`, `ProximitySensor`, `LowLightArgs`, `HighLightArgs`, `ProximityArgs` 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.