Utilisation du détecteur à incendie

Le détecteur à incendie émet des impulsions électriques lorsqu’il détecte de la fumée. La valeur de ces impulsions ne nous intéresse pas. Tout ce dont nous avons besoin, c’est de pouvoir identifier un changement d’état sur le détecteur de fumée. Ainsi, une valeur booléenne est suffisante. Lorsque de la fumée est détectée, l’état du détecteur passe continuellement de TRUE à FALSE. Sinon, elle vaut toujours TRUE. Ainsi, il suffit de compter le nombre de changement d’état pour savoir si nous sommes face à une alerte incendie. Nous utilisons le terme « impulsion » dans la suite de ce document pour désigner un changement d’état tel que « TRUE 🡪 FALSE » ou « FALSE 🡪 TRUE ».

Dans notre le monde physique et tangible, une alarme doit être activée pour pouvoir lancer une alerte. Elle peut aussi être désactivée. Dans ce dernier cas, elle ne tient pas compte des valeurs renvoyées par les capteurs auxquelles elle est raccordée. L’alarme incendie suit la même logique. Elle dispose de trois méthodes dédiées à cet usage :

* **Public void activer()** 🡪 active l’alarme pour la capture des impulsions
* **Public void desactiver()** 🡪 désactive l’alarme. Les impulsions ne sont pas prises en comptes.
* **Public void cloturer()** 🡪 libère les ressources et l’accès à l’interface kit du phidget. Cette méthode doit être appelée durant la phase de clôture du programme. Toute utilisation en amont dans le code provoquera une erreur dans le logiciel en cas d’appel ultérieur sur l’objet représentant l’alarme incendie.

Un objet « DetecteurIncendie » contient un compteur pour le nombre d’impulsions nommé nbrImpulsions et une limite nommée limiteImpulsions au-delà de laquelle l’alarme est lancée. Ces deux valeurs sont des nombres entiers strictement positifs. Ils peuvent être modifiés par leurs couples de setters et getters, respectivement :

* **Public int getNbrImpulsions()**
* **Public void setNbrImpulsions(int nbrImpulsions)**
* **Public int getLimitImpulsions()**
* **Public void setLimitImpulsions( int limitImpulsions)**

Ces deux compteurs sont la base des trois modes de fonctionnement pour le détecteur d’incendie :

* **Mode 1 :** lorsque la limite d’impulsion est atteinte, l’alarme est déclenchée. nbrImpulsion est ici un simple compteur accumulatif. À chaque fois que la limite est franchie, l’alarme est déclenchée.
* **Mode 2 :** lorsque la limite d’impulsion est atteinte, l’alarme est déclenchée et le compteur nbrImpulsion est réinitialisé à zéro.
* **Mode 3 :** lorsque la limite d’impulsion est atteinte, l’alarme est déclenchée, le compteur d’impulsion est réinitialisé et l’alarme est désactivée.[[1]](#footnote-1)

Le mode peut être spécifié lors de l’appel au constructeur en spécifiant le paramètre et en cours de route avec le couple de getter et setter correspondant, respectivement :

* Public int getMode()
* Public setMode( int mode)

Lorsque l’alarme est déclenchée, la méthode « alarmeIncendie » d’un objet quelconque implémentant l’interface « Pompier » est appelée. C’est à l’utilisateur d’un objet alarme incendie de définir un objet implémentant l’interface pompier pour effectuer les actions de son choix lorsque l’alarme se déclenche. La référence à l’objet pompier peut être spécifie lors de l’instanciation et pendant l’exécution avec le couple de getters et setters correspondants :

* **Public void setPompier( Pompier pompier)**
* **Public pompier getPompier()**

Un objet détecteur incendie peut être créé selon trois constructeurs différents :

* **public DetecteurIncendie(Pompier pom, int limiteImpulsions, int mode)**
* **public DetecteurIncendie(int limiteImpulsions, int mode)**
* **public DetecteurIncendie(int limiteImpulsions)**
* **public DetecteurIncendie()**

1. On comprend ainsi qu’une désactivation de l’alarme ne remet pas le compteur nbrImpulsion à zéro. [↑](#footnote-ref-1)