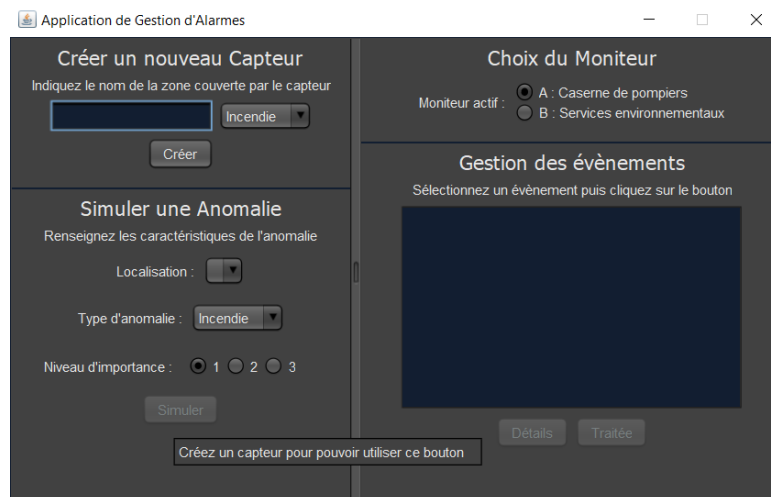


Rapport de projet

INFO641 : Conception et Programmation Orientée Objet

Mini-Projet Alarme



Lien GitHub : <https://github.com/SimonGuilbert/Alarm>

Enseignants :

Cimpan Sorana

Mauffret Etienne

Vernier Flavien

Simon Guilbert

Introduction - Présentation du projet

Ce projet est une application codée en Java et qui permet principalement de générer une interface graphique divisée en deux fenêtres.

L'une de ces fenêtres permet de simuler des alarmes provoquées par des anomalies telles que des incendies, des fuites de gaz et un taux élevé de radiations dans des bâtiments d'un laboratoire de physique.

L'autre fenêtre sert à gérer ces alarmes. L'utilisateur peut demander des détails sur l'anomalie à l'origine d'une alarme qui apparaît à l'écran et une fois que cette alarme a été traitée, il a la possibilité de la faire disparaître.

Le fonctionnement du programme pour simuler et gérer les alarmes repose sur le principe du modèle à événements. Simuler une anomalie revient à créer un événement qui sera écouté par des moniteurs appelés A (une caserne de pompiers) et/ou B (les services environnementaux).

I. Fonctionnement du programme

Classe Main : point d'entrée du code

Au lancement du programme, les lignes de code de la classe Main s'exécutent et elles servent principalement à faire deux choses :

- appeler la méthode statique *darkMode()* qui permet de modifier les couleurs d'arrière plan et de police d'écriture des JFrame (et autres fenêtres).
- créer une instance de la classe Fenetre. Cette classe sert de gestionnaire des différentes fenêtres JFrame qui apparaîtront à l'écran par la suite.

Classe Fenetre : gère les différentes interfaces graphiques

Dans le constructeur de la classe Fenetre, une instance de FenetreGauche et une autre instance de FenetreDroite sont créées. Dans le même temps, une barre de menu est ajoutée en haut de chaque fenêtre et permet de fermer toutes les fenêtres visibles à l'écran en cliquant sur Fermer puis sur Tout fermer.

Classe FenetreGauche : simulation d'une alarme

La fenêtre de gauche sert à créer des capteurs d'incendie, de gaz ou de radiations et à simuler des alarmes qui seront déclenchées par ces capteurs.

The screenshot shows a window titled 'Simulation alarmes' with a 'Fermer' button in the top right. The window is divided into two main sections. The top section, 'Créer un nouveau Capteur', has a subtitle 'Indiquez le nom de la zone couverte par le capteur' and a text input field. Below the input field is a dropdown menu currently set to 'Incendie' and a 'Créer' button. The bottom section, 'Simuler une Anomalie', has a subtitle 'Renseignez les caractéristiques de l'anomalie'. It contains three controls: a 'Localisation :' dropdown menu, a 'Type d'anomalie :' dropdown menu set to 'Incendie', and a 'Niveau d'importance :' section with three radio buttons labeled '1', '2', and '3'. At the bottom of this section is a 'Simuler' button.

A chaque nouveau lancement du programme, aucun capteur n'existe et on ne peut donc pas encore simuler une anomalie puisqu'aucun capteur ne pourra la détecter. C'est pour ça qu'au début, la liste déroulante des localisation possibles d'une anomalie est vide et que le bouton Simuler (en bas de la fenêtre) est désactivé. Si l'utilisateur essaie néanmoins de cliquer dessus, le tooltip suivant apparaîtra :

Créez un capteur pour pouvoir utiliser ce bouton

Une fois que l'utilisateur aura créé au moins un capteur via le bouton Créer, la liste déroulante des localisations sera automatiquement alimentée et le bouton Simuler sera activé.

Pour créer un capteur de fuite de gaz couvrant la zone du laboratoire, il faut renseigner ces informations puis cliquer sur le bouton Créer :

This screenshot shows the 'Créer un nouveau Capteur' section of the 'Simulation alarmes' window. The text input field now contains 'Laboratoire' and the dropdown menu next to it is set to 'Gaz'. The 'Créer' button is still present below these fields.

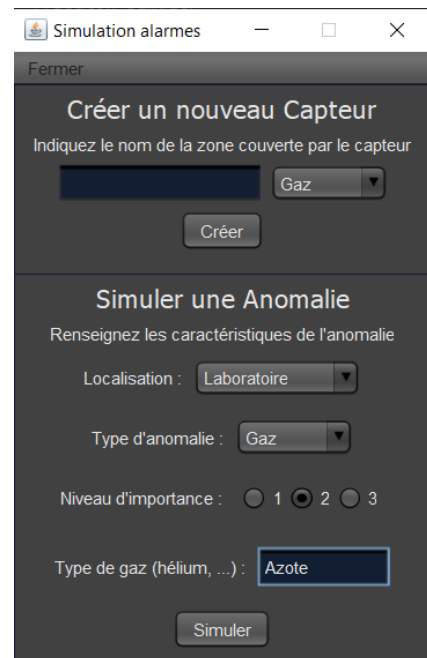
Le capteur sera alors ajouté à l'attribut de la classe Fenetre dicoCapteurs et par la même occasion dans la liste déroulante en face du mot Localisation sur la fenêtre de gauche.

Sur l'image ci-contre, une fuite d'azote de niveau 2 est sur le point d'être simulée dans le Laboratoire. Pour que l'anomalie soit détectée, il faut qu'un capteur de type Gaz couvrant la

zone Laboratoire ait déjà été créé. Si c'est le cas, ce capteur apparaîtra dans la liste déroulante en face de Localisation.

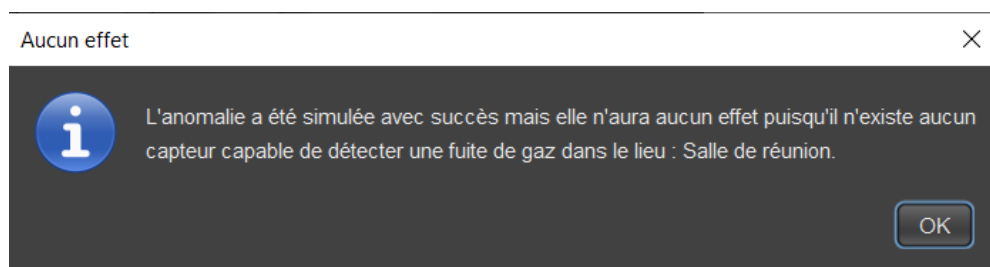
Une fois le lieu choisi, l'utilisateur devra sélectionner le mot Gaz dans la seconde liste déroulante. A ce moment là, la zone de texte apparaîtra pour lui permettre d'indiquer le type de gaz (hélium,...). Enfin, l'utilisateur indiquera le niveau d'importance de la fuite de gaz en sélectionnant le bouton radio dont l'étiquette vaut 1, 2 ou 3.

Une fois le bouton Simuler cliqué, le programme va déclencher l'alarme en appelant la méthode `simulerFuiteGaz()` de la classe `GazCapteur`, tout en veillant à ce qu'aucune exception ne soit levée.



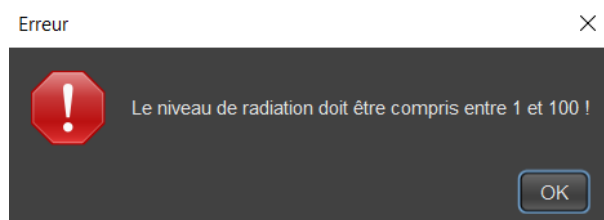
Classe TypesDifferentException (dérive de Exception) : type anomalie ≠ type capteur

Étant donné qu'un capteur est spécifique à une anomalie (un capteur à incendie ne pourra détecter que des incendies et pas des gaz ni des radiations), il faut que le type d'anomalie choisi dans les 2 listes déroulantes concordent. Autrement dit, si on simule un incendie dans la salle de réunion, il faut qu'un capteur à incendie existe déjà dans la salle de réunion pour pouvoir le détecter. Dans le cas contraire, la boîte de dialogue suivante apparaîtra et l'utilisateur sera convié à créer le capteur manquant ou à choisir un autre type d'anomalie :



Classe ValeurIncorrecteException (dérive de RuntimeException) : nombre <1 ou >100

Cette exception ne peut être déclenchée que dans le cas d'une simulation d'accident nucléaire lorsqu'une valeur du niveau de radiation inférieure à 1 ou bien supérieure à 100 a été renseignée. L'erreur est détectée au niveau du constructeur du `RadiationEvent`.



Classe abstraite Capteur (mère de IncendieCapteur, GazCapteur, RadiationCapteur)

Une fois ces exceptions traitées et les informations correctement renseignées par l'utilisateur, la première méthode appelée pour déclencher tout le processus d'événements est la méthode `simulerFuiteGaz()` de la classe `GazCapteur`. Bien sûr, dans le cas d'un incendie ce sera la méthode `simulerIncendie()` de la classe `IncendieCapteur` et pareil pour un accident nucléaire.

Classe abstraite AnomalieEvent (mère de IncendieEvent, GazEvent, RadiationEvent)

La première chose que va faire la méthode `simulerFuiteGaz()` c'est créer l'événement de type `GazEvent`. Quelque soit la nature du `GazEvent`, il dérivera toujours de la classe abstraite `AnomalieEvent` possédera toujours au moins les 3 attributs suivants :

- `date` au format `GregorianCalendar`. Cette date correspond au moment où l'utilisateur clique sur le bouton `Simuler`
- `localisation` : une chaîne de caractères précisant le lieu de l'accident
- `nivImportance` : le niveau d'importance de l'accident, un nombre entier 1 et 3

Dans le cas d'un `GazEvent` on ajoutera l'attribut suivant :

- `typeGaz` : le nom du gaz qui fuit (hélium, CO2, ...)

Enfin pour un `RadiationEvent` on ajoute :

- `niveau` : le niveau de radiation détecté. Il est affecté grâce à la méthode `setNiveau()`. C'est ici que la `ValeurIncorrecteException` est déclenchée si on essaye de créer un `RadiationEvent` dont le niveau de radiation n'est pas compris entre 1 et 100.

Interface AnomalieListener dérive de EventListener (mère de IncendieListener, GazListener, RadiationListener)

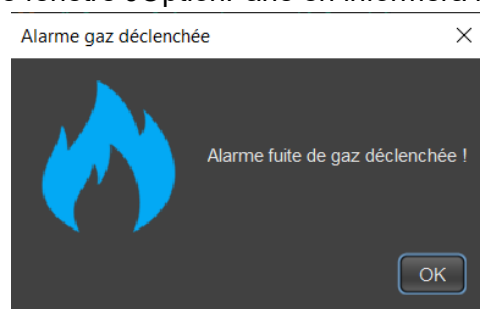
Ensuite, on indique à tous les `GazListener` associés au `GazCapteur` qu'un `GazEvent` vient de se produire en appelant leur méthode `declenchementAlarmeGaz(GazEvent)`. Tous les `AnomalieListeners` d'un `Capteur` sont sauvegardés dans une liste d'`EventListener` initialisée à chaque création d'un `Capteur` et actualisée avec la méthode `setListeners()` de la classe `Fenetre`.

Classe Moniteur (implémente IncendieListener, IncendieListener et RadiationListener)

Dans cet exemple de modèle à événements, les `AnomalieListener` sont des interfaces puisqu'il y a encore un niveau d'hérédité. En effet, les objets `Moniteur` jouent le rôle d'écouteurs et implémentent donc les listeners.

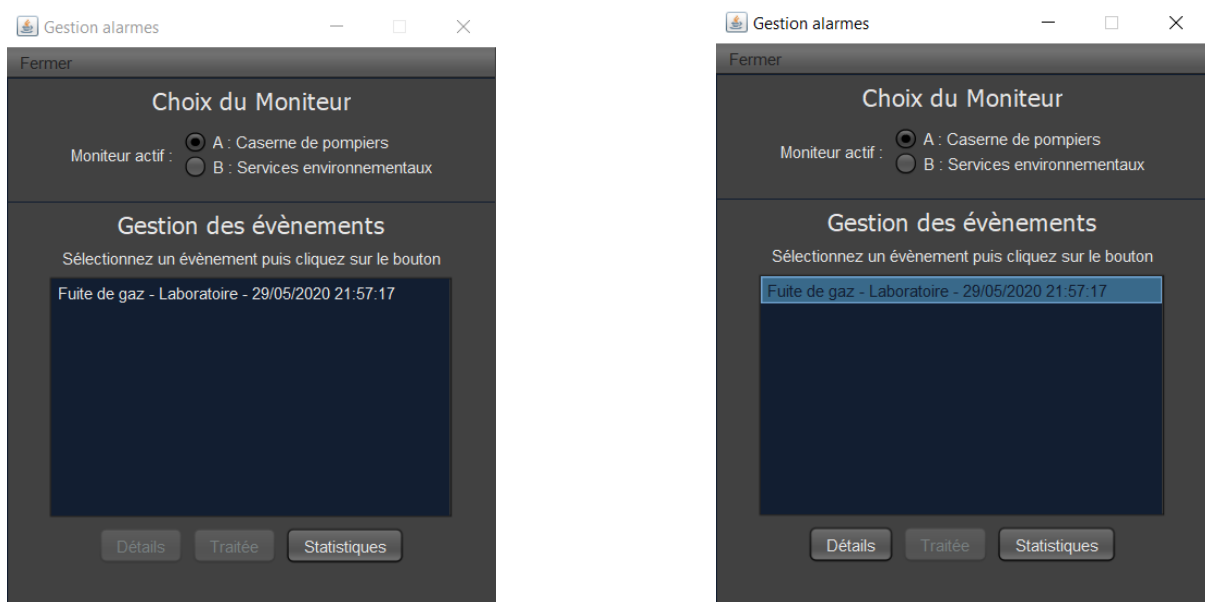
Le parcours de notre `GazEvent` créé depuis que l'utilisateur a appuyé sur le bouton `Simuler` se termine dans la liste des listeners, attribut de chaque objet `Moniteur`.

Si tout s'est bien passé, une fenêtre `JOptionPane` en informera l'utilisateur :

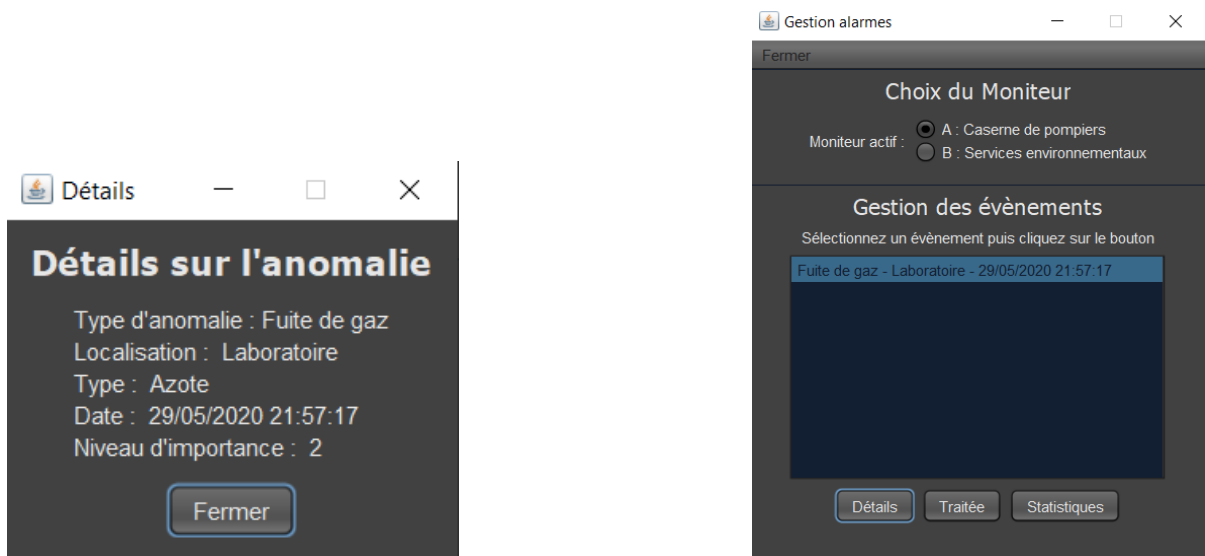


Classe FenetreDroite : gestion des alarmes simulées

De plus, dès que l'utilisateur aura appuyé sur le bouton "OK", la JList sur la fenêtre de droite va s'actualiser et le GazEvent apparaîtra dans la liste :



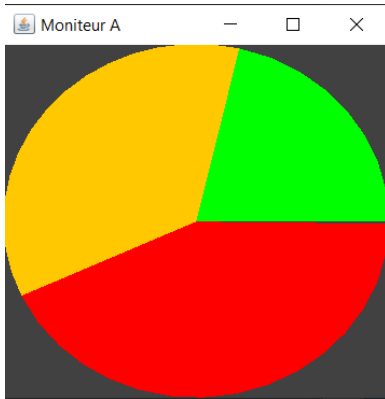
La fenêtre de droite est dédiée à la gestion des alarmes. En sélectionnant l'un des éléments de la liste (comme sur la figure ci-dessus) le bouton Détails s'active. Lorsqu'on clique dessus, une fenêtre affiche les détails de l'évènement sélectionné (figure ci-dessous) et le bouton Traitée s'active. En cliquant sur ce bouton, la fenêtre de détails disparaît et l'évènement sera retiré de la liste.



Le haut de la fenêtre de droite permet de modifier le moniteur actif. Dès que l'on sélectionne "A : Caserne de pompiers" la liste s'actualise et affiche tous les incendie et fuite de gaz simulées et non traitées. L'autre bouton radio affichera les fuites de gaz et les accidents nucléaires.

II. Ajouts par rapport à la date de la démonstration

Ajout du bouton Statistiques

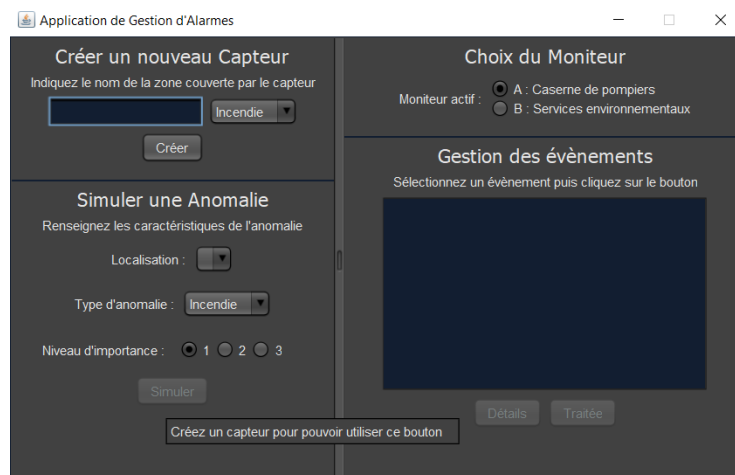


Ce bouton est activé dès qu'au moins un événement est simulé et entendu par un moniteur. Quand on appuie dessus, un diagramme circulaire apparaît et montre la proportion des alarmes entendues par un moniteur en fonction de leur niveau d'importance (1, 2 ou 3) :

Les classes PieChart et Section sont utilisées pour créer ce graphique, où les sections servent à délimiter les 3 parties en couleur du graphique. Le niveau d'importance 1 est représenté en vert, le niveau 2 en orange et le niveau 3 en rouge.

Séparation des fenêtres

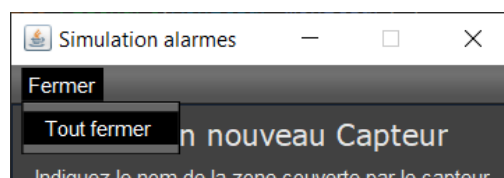
Au moment de la démonstration, il n'y avait qu'une seule fenêtre qui servait à la fois à simuler et à gérer les alarmes. Elles étaient simplement séparées par une barre verticale :



Maintenant, les fenêtres sont séparées ce qui permet à deux services qui sont potentiellement situés à des endroits différents de pouvoir simuler et gérer les alarmes de manière indépendante.

Ajout de la barre de menu

Enfin, le dernier ajout qui a été fait est la création d'une barre de menu en haut de chaque fenêtre. Cette barre ne contient que le menu Fermer puis le sous-menu Tout fermer qui permet la fermeture simultanée de toutes les fenêtres visibles à l'écran.



Conclusion - Ce qu'il reste à faire

Pour conclure, le modèle à événement et les différentes manipulations que l'on peut faire avec l'interface graphique ont l'air de fonctionner correctement.

Pour ce qu'il reste à faire, le graphique circulaire n'est vraiment pas très beau. Il pourrait largement être amélioré en utilisant une librairie plus spécialisée dans les statistiques et les graphiques. Et surtout, je n'ai pas réussi à intégrer une légende sur la même fenêtre que le graphique : pour l'instant, la fenêtre décrivant la légende est séparée et l'utilisateur peut s'y perdre facilement puisqu'en tout 4 fenêtres différentes sont visibles à l'écran quand il clique sur le bouton Statistiques.

On pourrait aussi améliorer la barre de menu qui ne permet pour l'instant que de fermer toutes les fenêtres et rien d'autre.

Enfin, d'un point de vue plus personnel, la partie de ce projet qui m'a posé le plus de difficultés est le modèle à événements. J'ai eu beaucoup de mal à me faire à l'idée que le rôle des capteurs n'était pas de "capter" et donc d'écouter les événements mais plutôt de les créer et les déclencher. Le rôle d'écouteur était réservé aux moniteurs. Je me suis aussi beaucoup emmêlé les pinceaux avec les nombreuses relations entre les classes mères, classes filles, interfaces, classes abstraites, ... Mais après avoir fini ce projet j'ai l'impression d'avoir mieux compris tout ce système et j'ai aussi appris beaucoup de choses au niveau des interfaces graphiques.