Mokhtar MIAL
Rana SHERIF

---

# TP4

## Question 1 - Lecture et test d’invariant

- **Propriété 1:** Cette propriété assure que l'on n'essaie pas de stocker plus d'éléments incompatibles que possible.

  **Propriété 2:** Cette propriété assure qu'il n'y aura pas plus de 30 éléments affectés aux bâtiments.

  **Propriété 3:** Cette propriété assure que toutes les paires d’éléments incompatibles commencent par la chaîne de caractères “Prod".

  **Propriété 4:** Cette propriété assure que dans toutes les paires de bâtiment produit, l'élément à l'index 0 commence par la chaîne de caractères "Bat" et l'élément à l'index 1 commence par la chaîne de caractères “Prod".

  **Propriété 5:** Cette propriété assure que l'on n'ajoute pas une incompatibilité entre un produit et lui-même.

  **Propriété 6:** Cette propriété assure que les incompatibilités sont correctement dupliquées pour obtenir un résultat tel que pour deux produits incompatibles A et B, [A, B] et [B, A] sont présents dans le tableau.

  **Propriété 7:** Cette propriété assure que si deux éléments se trouvent dans le même bâtiment, ils ne peuvent pas être une paire d'éléments incompatibles.

- On a écrit 7 tests différents, chacun ne respecte pas une des propriétés dessus. Chaque test fait référence à l’invariant que l’on cherche à invalider grâce au commentaire du message d’erreur. Par conséquent, les 7 tests du fichier **TestExplosivesJUnit4.java** échouent.

## Question 2 - Calcul de préconditions

Dans le fichier Explosives.java, on a rajouté les préconditions suivantes:

| Propriété | Explication de la précondition associée |
|---|---|
| 1 | Le nombre d'éléments incompatibles **nb_inc** doit être strictement inférieur à la taille du tableau **incomp**. |
| 2 | On vérifie que la variable **nb_assign** est strictement inférieure à 30. |
| 3 | On vérifie que les chaînes de caractères **incomp**[i][0] et **incomp**[i][1] commencent par prod, avec i supérieur ou égal à 0 et i strictement inférieur à 50. |
| 4 | On vérifie que la chaîne de caractère **assign**[i][0] commence par Bat et la chaîne de caractère **assign**[i][1] commence par Prod, avec i supérieur ou égal à 0 et i strictement inférieur à 30. |
| 5 | On vérifie que **incomp**[i][0] et **incomp**[i][1] sont différents. |
| 6 | On vérifie que pour tout **incomp**[i], il existe un j tel que **incomp**[j][0] = **incomp**[i][1] et **incomp**[j][1] = **incomp**[i][0]. Pour i et j supérieurs ou égal à 0 et i, j strictement inférieurs à 50. |
| 7 | On vérifie que pour chaque **assign**[i][0] et **assign**[j][0] qui sont égaux, **assign**[i][1] et **assign**[j][1] ne sont pas une paire d'éléments incompatibles, c'est à dire qu'il n'existe pas un k tel que **incomp**[k][**assign**[i][1]] et **incomp**[k][**assign**[j][1]]. Pour i, j supérieurs ou égal à 0 et i, j strictement inférieurs à 30 et k supérieur ou égal à 0 et k strictement inférieur à 50 |

Ensuite, on a rajouté des tests Inconclusifs qui invalident nos préconditions dans le fichier **TestExplosivesJUnit4.java**. Par conséquent, ces tests échouent.

## Question 3 - Ajout d'assertions

Dans la classe **Explosives.java**, on a rajouté deux invariants pour les propriétés 8 et 9 et une contrainte d'histoire pour la propriété 10.

## Question 4 - Recherche d'un bâtiment

La fonction findBat a les spécifications suivantes:

- Le String passé en paramètre doit commencer par "Prod".
- Soit A le produit passé en paramètre. La fonction retourne un des bâtiments "valides", où c'est possible de stocker A. Pour cela, on commence par créer un tableau **Incompatibilities** où on stocke les éléments incompatibles avec A.

Ensuite, on crée une map **batiments** qui pour chaque bâtiment, associe les éléments qui y sont présents. Après, on parcours cette map et on renvoie le premier bâtiment dont tous les produits ne se trouvent pas dans **Incompatibilities** et qui ne contient pas A.

- La fonction renvoie le premier bâtiment “Valide” qui ne contient pas déjà A.