Module et Généricité : Exemple de la pile

Objectifs

- Comprendre la notion de module (paquetage en Ada).
- Comprendre la généricité.
- Savoir gérer deux instances d'un même module générique.

Rappel : Comme pour tous les TP, il faut commencer par faire un « `git pull` » depuis votre dossier « pim/tp » pour récupérer les fichiers fournis pour cette séance.

Exercice 1 : Module générique en Ada : exemple de la pile

L'objectif de cet exercice est de comprendre les modules en Ada. On s'appuiera en particulier sur le module `Piles` (fichiers `piles.ads` et `piles.adb`) et le programme `utiliser_piles.adb`.

1. Pourquoi les commentaires de spécification n'apparaissent que dans la spécification du module `Piles` et pas dans son implantation ?

2. Où sont formalisés les contrats ?

3. Ce module `Piles` est générique. Comment le sait-on ? Quels sont ses paramètres de généricité ? Que faut-il faire pour pouvoir utiliser ce module `Piles` ?

4. Où faudrait-il définir un sous-programme spécifique au module `Piles` ? Et sa documentation (commentaire de spécification) ?

5. Qu'est ce que la surcharge ? La lecture de la procédure `Illustrer_Surcharge` aidera à répondre à cette question. On suivra en particulier les consignes données en commentaire et on décommentera la ligne indiquée.

6. La procédure `Afficher_Element` n'est pas un paramètre de généricité du module `Piles` mais de sa procédure Afficher. Pourquoi ? La lecture de `Illustrer_Plusieurs_Afficher_Pour_Meme_Pile` aidera à répondre à cette question.

7. Si dans un même contexte, par exemple un même sous-programme, on a besoin de deux piles avec des caractéristiques différentes (capacités différentes ou types des éléments différents), comment gérer le fait que les deux instances du module `Piles` fournissent les mêmes noms (`T_Pile`, `Empiler`, `Est_Vide`...) ? Ce cas de figure est présent dans `Illustrer_Plusieurs_Piles`.

Exercice 2 : Expression bien appairée

On s'intéresse au fichier `parenthesage.adb`.

1. Compléter la fonction `Index` puis la procédure `Verifier_Parenthesage`.

Pour la procédure `Verifier_Parenthesage`, on utilisera deux piles (même si une seule pile pourrait suffire) : la première `Pile_Ouvrants` stockera les symboles ouvrants rencontrés lors de l'analyse de la chaîne et la seconde, `Pile_Indices`, stockera leur indice.

Un programme de test est fourni pour chacun de ces sous-programmes.

Remarque : Pour un paramètre de type `Chaine : in String`, on peut avoir des indices différents pour la chaîne. Aussi, il est conseillé d'utiliser `Chaine'First`, `Chaine'Last` ou `Chaine'Range`.
2. Est-ce que l'on peut utiliser `Verifier_Parenthesage` dans d'autres programmes ? Que faudrait-il faire pour y arriver ?