UQÀM 2023

Rapport TP3

MetaProgramming en Python

Équipe: Cypress

Alireza Nezami

nezami.alireza@courrier.uqam.ca

MGL7460 Réalisation et maintenance du logiciel
Professeure : Monsieur Hafedh Mili

Choix de conception

Ici, vous décrivez ce qui, selon vous, mérite d'être mentionné par rapport aux choix de conception de bas niveau/idiomes de programmation utilisé, comme cela a été fait avec les deux premiers TPs. Vous pouvez aussi marquer vos hypothèses.

Dans ce projet, nous avons réalisé une implémentation en Python, ce qui facilite le travail sur les fichiers JSON malgré le fait que cette technologie soit dotée de nombreuses bibliothèques.

Réflexions sur le travail

1. Le projet fait certaines hypothèses concernant le mapping entre structure de données JSON et structures de classes. Veuillez : a) les récapituler, b) discuter si elles sont correctes, et s'il y a d'autres interprétations alternatives, et c) réfléchir à (spéculer sur) une stratégie potentielle pour rendre le « mapping » unique.

   a) - Chaque clé de la structure JSON est mappée à un attribut de la classe.
   - Les valeurs de chaque clé en JSON sont traitées comme la valeur de la propriété correspondante dans la classe.
   - Les structures JSON internes (tableaux, autres objets JSON) sont représentées sous forme d'objets imbriqués ou de listes dans des classes.

   b) - Ces hypothèses peuvent s'avérer insuffisantes dans les cas où une correspondance hétérogène entre la structure et la classe JSON est requise. Par exemple, comment gérez-vous les cas comportant des champs variables ?

   Si une clé en JSON correspond à plusieurs propriétés dans une classe, un mécanisme tel qu'un identifiant unique pour chaque classe peut être utilisé pour résoudre ce problème.

   c) - Utilisez un fichier de configuration distinct qui fournit le mappage entre la structure JSON et les classes. Ce fichier de configuration peut déterminer comment mapper les éléments problématiques.
   - La possibilité de fournir une interface visuelle permettant aux utilisateurs de déterminer la carte. Cette interface permet aux utilisateurs de réaliser et de personnaliser eux-mêmes la cartographie.

2. Votre programme génère et exécute du code. Réfléchissez (discuter) les enjeux de tests pour les générateurs de code, et spéculer (ou documentez-vous et faites référence aux) stratégies de test possibles[1].

   Défis de test pour les générateurs de code :
   - Unicité du code généré : Garantir que le code généré sur la base de l'entrée liée à JSON est généré de manière unique et sans interférence avec d'autres parties du programme.
   - Qualité du code produit : Assurer la production d'un code lisible, maintenable et fiable.
   - Conformité aux standards : S'assurer que le code généré est compatible avec les standards de programmation souhaités (tels que PEP 8 pour Python).

Exceptions d'erreur : Tests pour garantir que le code généré gère correctement les erreurs et les exceptions et les gère de manière optimale.

- Intégration réussie : S'assurer que le code généré s'intègre correctement au reste de l'application et s'exécute correctement.
- Couverture complète des tests : Assurer une couverture complète de tous les types d'états d'entrée et des différentes conditions dans les tests.

Stratégies possibles pour les tests :

- Tests unitaires : Test de chaque partie du code généré indépendamment.
- Tests intégraux (*Integration Tests*) : Assurer l'intégration