INF8881 — Principes avancés des langages à objets
Examen final

Jean Privat

Mercredi 20 mai 2009

Consignes générales :
- Les documents sont autorisés mais pas nécessaires.
- Veuillez systématiquement justifier et argumenter vos réponses même lorsque ce n'est pas précisé dans l'énoncé des questions.
- En cas de doute sur la compréhension d'une question, n'hésitez-pas à demander des éclaircissements.

Classes, types statiques et généricité

L'objectif est de modéliser les concepts de *classes* et de *types statiques* nécessaires à la réalisation d'un analyseur de code source tel qu'on peut en trouver dans un compilateur ou d'autres outils connexes.

Le langage considéré par l'outil est un langage pur à objet, en héritage simple, à typage statique sûr et avec la généricité bornée simple.

1. Expliquez pourquoi un tel langage impose de distinguer le concept de type statique du concept de classe alors que cela ne serait pas nécessaire dans le même langage mais sans généricité.
2. Quel est le type statique du receveur dans une classe générique ? Justifiez votre réponse.
3. Soient deux types statiques génériques t_1 =`A<B>` et t_2 =`C<D>`, comment déterminer si t_1 est un sous-type de t_2 ? Justifiez votre réponse.

   Pour répondre à la question, vous pouvez entre autres considérez les cas suivants :
   - `A` = `C`.
   - `A` sous-classe directe de `C` (plusieurs sous-cas possibles).
   - `A` incomparable avec `C` (mais elle possèdent au moins un ancêtre commun, éventuellement `Object`).
   - `A` sous-classe directe de `E` (qui n'est pas générique) et `E` sous-classe directe de `C` (plusieurs sous-cas possibles).
4. Même question mais en considérant cette fois que la politique de typage est covariante (non sûre).

5. Proposez une modélisation (classes, méthodes sans leur corps, attributs) qui incluent les éléments suivants :
   - Les concepts de *type statique* et de *classe* (plusieurs classes de *types* et plusieurs classes de *classes* peuvent être nécessaires).
   - Une méthode `get_type`, sans paramètre, qui retourne le type statique associé au receveur dans une classe.
   - Une méthode `get_param(i)` qui retourne le i-ème paramètre formel d'une classe générique.
   - Une méthode `get_instantiate_type` qui retourne un type statique instancié d'une classe générique. `get_instantiate_type` prend en paramètre une liste de types.
   - Une méthode `is_subclass` qui indique si une classe en spécialise une autre (directement ou non).
   - Une méthode `is_subtype` qui indique si un type est un sous-type d'un autre. N'oubliez pas que la relation de sous-typage est réflexive.
   - Une méthode `get_class` qui indique à quelle classe correspond un type statique. Cette méthode sert à connaître les propriétés statiquement disponibles via un type statique.
   - Une méthode `get_bound` qui retourne le type qui sert de borne à un type formel.
   - Les constructeurs qui vont avec les classes.

   Attention ! Quelque soit vos choix de modélisation, il est important que vous les justifiez voire que vous les discutiez si plusieurs modélisations possibles vous viennent à l'esprit.
6. Implémentez le corps des méthodes que vous avez modélisé. Normalement, le code de chacune des méthodes est relativement simple ; la seule difficulté devrait être la méthode `is_subtype` pour les types génériques car elle devrait nécessiter que vous ajoutiez des éléments qui n'apparaissent pas dans la liste de la question précédente.

   Attention ! Arrangez-vous pour qu'un seul objet par type statique existe : ne créez pas plusieurs fois un même type statique.
7. Exposez les différentes raisons qui font qu'il est intéressant de ne pas dupliquer une même instance d'une classe *type statique* (cf. la mise en garde de la question précédente).
8. Quelles auraient été les difficultés supplémentaires dans les questions 5 et 6 si jamais le langage était en héritage multiple ? Justifiez votre réponse.
9. Expliquez pourquoi l'exercice que vous venez de faire s'apparente à de la méta-programmation ?