Université Paris Cité

Compléments en Programmation Orientée Objet
TP n° 6 (Noté) - Info 4 + Info-Jap
Modéliser les formules propositionnelles

À savoir

- Ce TP sera noté, le code source contenant vos réponses est à **rendre** via Moodle **5 minutes avant la fin de la séance**. Seulement, les fichiers archives `zip` and `tar` sont acceptés (merci d'éviter les archives `rar`).
- À part de classes demandées dans le sujet, on attend une où plusieurs classes avec un `main` pour effectuer les tests.
- Si vous avez de remarques particulières, mettez les dans commentaires dans votre code. Il est possible d'ajouter un ficher texte `README` avec des explications supplémentaire (bien que on ne voit pas d'utilité).
- Le cours, vos réponses et notes des TPs précédents ainsi que la documentation et les ressources Java sur internet sont autorisés.
- **Toute communication entre deux personnes dans la salle ou entre une personne dans la salle et quelqu'un à l'extérieur sera considérée comme une fraude.**

1 Formules propositionnelles

Le but de ce TP est d'implémenter une petite bibliothèque pour la manipulation des formules propositionnelles, une telle formule est construite à partir de **connecteurs logiques**, des **variables propositionnelles** et des **valeurs booléennes** :

- Une **valeur booléenne** $\in \{true, false\}$;
- Une **variable propositionnelle** (telle que p, q...) est reconnue par son nom ;
- Les **connecteurs logiques** (`Not` ($\neg$), `And` ($\wedge$) et `Or` ($\vee$)) permettent de construire des formules composées à partir des valeurs booléennes, variables propositionnelles et/ou d'autres formules composées.

Exemples de formules : $true$, p, $p \wedge true$, $p \vee (a \wedge c)$...

2 Programmation

2.1 Usage

On veut que la bibliothèque soit utilisable de la façon suivante

```
Context.Formule tt = Context.Value(true); // true
Context.Formule ff = Context.Value(false); // false
Context.Formule p = Context.Variable("p"); // p
Context.Formule q = Context.Variable("q"); // q
Context.Formule ex1 = Context.And(List.of(p, q, tt));
Context.Formule ex2 = Context.Neg(q);
Context.Formule ex3 = Context.Or(List.of(ex1, ex2, ff));
```

Dans l'exemple au-dessus, la variable `ex3` doit contenir la représentation de la formule

$$(p \wedge q \wedge true) \vee \neg q \vee false$$

Sur Moodle, vous trouvez le squelette de la bibliothèque que vous implémenteriez avec un ensemble de tests pour vérifier son fonctionnement à chaque niveau.
Compléter le fichier `Context.java` jusqu'à que le fichier `Test1.java` (ignorer les autres fichiers Test pour le moment) soit compilable et exécutable en implémentant les fabriques statiques `Value`, `Variable`, `And`, `Neg` et `Or` et les différentes classes nécessaires pour représenter les formules comme décrites dans la section 1.

2.2 Affichage et `toString`

Ajouter dans toutes les classes la méthode `toString` appropriée. On va suivre le style des S-expression [1] pour l'affichage des formules et comme le tableau suivant le résume :

Formule	Son Affichage
$true$	`true`
$false$	`false`
p	`p`
$\neg p$	`(not p)`
$\neg\neg p$	`(not (not p))`
$p \wedge q$	`(and p q)`
$(p \wedge q) \vee \neg p$	`(or (and p q) (not q))`

Les tests dans `Test2.java` et les commentaires qu'il contient doivent vous aider à implémenter cette phase.

2.3 Implémentation de `substitute`

Définissez pour objets de type `Context.Formule`, la méthode `Formule substitute(Map<String, Formule> substitution)` qui retourne une nouvelle formule identique à la formule au quelle elle est appelée avec les variables données dans le paramètre `substitution` remplacés par la formule associée. Exemple :

```
Context.Formule tt = Context.Value(true); // true
Context.Formule ff = Context.Value(false); // false
Context.Formule p = Context.Variable("p"); // p
Context.Formule q = Context.Variable("q"); // q
Context.Formule ex1 = Context.And(List.of(p, q));
System.out.println(ex1); // (and p q)
System.out.println(ex1.substitute(Map.of(
    "q", p
))); // (and p p)
System.out.println(ex1.substitute(Map.of(
    "p", q,
    "q", p
))); // (and q p)
```

Les tests dans `Test3.java` et les commentaires qu'il contient doivent vous aider à implémenter cette phase.

2.4 Simplification de formule

Modifiez les fabriques statiques `And`, `Neg` et `Or` de `Context` tel qu'elles retournent des simplifications de formule quand c'est possible. Exemples :

Formule	Simplification
$\neg true$	$false$
$\neg\neg p$	p
$p_1 \wedge ... \wedge p_3 \wedge false$	$false$
$p_1 \wedge ... \wedge p_3 \wedge true$	$p_1 \wedge ... \wedge p_3$
$p_1 \vee ... \vee p_3 \vee false$	$p_1 \vee ... \vee p_3$
$p_1 \vee ... \vee p_3 \vee true$	$true$

Les tests dans `Test4.java` et les commentaires qu'il contient doivent vous aider à implémenter cette phase.

1. Pour plus d'information `https://fr.wikipedia.org/wiki/S-expression`

Critères d'évaluation

- Sécurité du sous-typage. (Impossibilité de créer d'autres cas/sous-types sans modification de votre code source.)
- Immuabilité. (Toute instance de votre hiérarchie doit un être un objet non modifiable.)
- Encapsulation maximale. (Inaccessiblité, depuis l'extérieur, de tout autre membre que les opérations demandées.)
- Utilisation (raisonnée) des briques de construction vues en cours (`abstract` / `final` / `sealed` / `enum` / `record`, ...).