Compléments en Programmation Orientée Objet
TP n° 6 : Modéliser l'algèbre de rectangles

Il est possible de consulter le cours, vos notes de TP et la documentation et les ressources java sur internet.

Une communication quelconque entre deux personnes dans la salle ou entre une personne dans la salle et quelqu'un à l'extérieur sera considérée comme une fraude.
Si vous avez des questions posez le au chargé de TP.

Rendu. Le code source est à rendre sur moodle avant la fin de séance, l'heure exacte indiquée par le chargé de tp.
Les fichiers archive `zip` ou `tar` sont acceptés, merci d'éviter les archives `rar`.
A part de classes demandées dans le sujet on attend une où plusieurs classes une avec `main` pour effectuer les tests.
Si vous avez de remarques particulières mettez les dans commentaires dans votre code. Il est possible d'ajouter le ficher texte `README` avec des explications supplémentaire (bien que je ne vois pas d'utilité).
Sur moodle vous trouverez (le squelette de) l'interface `ARS` pour le projet à implémenter en TP et un programme (très léger) de tests. Surtout il ne faut pas se contenter de tests fournis sur moodle qui concerne essentiellement une méthode. Vous devez ajouter d'autres tests.

1 Rectangles sur le plan

Pour un point a sur le plan nous allons écrire (a_x, a_y) pour désigner les coordonnées de a.
On considère les rectangles sur les plan avec les côté parallèles aux axes. Pour un tel rectangle avec les sommets a, b, c, d, les côtés ab et cd parallèles à l'axe des abscisses et les côtés bc et da parallèles à l'axe des ordonnés.
On suppose aussi les sommets a, b, c, d sont disposés de cette façon que

$$a_x <= c_x \quad \text{et} \quad a_y <= c_y \tag{1}$$

comme dans ce dessin
Nous allons dire que le rectangle avec les côté parallèles aux axes et qui satisfait (1) est un rectangle simple.
Évidemment si on connaît les sommets a et c d'un rectangle simple il est facile de calculer les coordonnées de sommets b et d, $b = (c_x, a_y), d = (a_x, c_y)$.

2 Algèbre de rectangles simples

Une algèbre engendrée par les rectangles simples c'est la plus petite famille $\mathcal{F}$ d'ensembles sur le plan telle que :

(1) chaque rectangle simple appartient à $\mathcal{F}$,

(2) si les ensembles T, S appartiennent à $\mathcal{F}$ alors l'union $T \cup S$, l'intersection $T \cap S$ et le complément $\overline{T} = \mathbb{R}^2 \setminus T$ appartiennent à $\mathcal{F}$.

Notez que la définition de l'algèbre engendrée par les rectangles simples est récursive.
Il est inutile d'essayer visualiser les éléments de l'algèbre engendrée par les rectangles simples, nous allons travailler uniquement avec les expressions algébriques.
Exemples. Dans les exemples on notera les opération sur les ensembles en utilisant la notation préfixe :

- l'union $R \cup T$ sera notée comme $\mathsf{U}(R, T)$,

- l'intersection $R \cap T$ sera notée comme $\mathsf{I}(R,T)$,
- le complément d'un ensemble R est souvent noté en mathématique comme $\overline{R}$ mais la notation préfixe $\mathsf{C}(R)$ est aussi parfois utilisée et elle sera reprise dans les exemples.

Soit P, Q, R, S, T des rectangles. Alors

- $\mathsf{C}(P)$: en notation mathématique $\overline{P}$,
- $\mathsf{C}(\mathsf{I}(\mathsf{U}(R,S),\mathsf{C}(\mathsf{U}(\mathsf{U}(P,R),Q))))$: en notation mathématiques $\overline{((R \cup S) \cap \overline{((P \cup R) \cup Q)})}$,
- $\mathsf{U}(\mathsf{I}(\mathsf{C}(R),\mathsf{C}(\mathsf{I}(S,R))),\mathsf{U}(\mathsf{C}(T),S))$: en notation mathématiques $((\overline{R} \cap \overline{(S \cap R)}) \cup (\overline{T} \cup S))$.

Bien sûr des expressions algébriques différentes peuvent désigner le même ensemble sur le plan, par exemple pour n'importe quels ensembles T, S, W nous avons $T \cap (S \cup W) = (T \cap S) \cup (T \cap W)$ (où avec notre notation préfixe $\mathsf{I}(T,\mathsf{U}(S,W)) = \mathsf{U}(\mathsf{I}(T,S),\mathsf{I}(T,W))$). Trouver si deux expressions représentent le même ensemble de point est intéressant mais hors sujet pour nos exercices.

3 Programmation

Votre tâche consiste à d'implémenter les rectangles simples et les opérations union, intersection, complément comme des classes implémentant le même interface `ARS` [1]. (Une autre solution est de définir `ARS` comme une classe abstraite avec les classes à définir comme les classes dérivées de `ARS`. Vous êtes libre à choisir et implémenter cette solution.)
Le squelette de l'interface `ARS` est disponible sur moodle.
L'interface `ARS` contient une seule méthode non-statique :

```
1    boolean contains( Point2D.Double p);
2
```

qui est sensée retourner `true` si le point `p` appartient à l'objet `this`.

L'interface `ARS` contient aussi les méthodes statiques (fabriques statiques) `getUnion`, `getIntersect`, `getComplement` qui retournent respectivement les instances de l'union, l'intersection et le compléments des ARS passés en paramètres.
Et finalement la fabrique statique `getRectangle` retournera un nouveau rectangle simple. Les paramètres de la fabrique `getRectangle` doivent satisfaire la condition (1), sinon on lancera `IllegalArgumentException`.
Les fabriques statiques sont définies dans l'interface `ARS` sur moodle mais leurs corps est vide, à vous de combler ce vide [2].

3.1 Implémentation de `contains`

Tout le mode doit être capable de fabriquer la formule qui vérifie si un point `p` appartient à un rectangle simple. Pour d'autre opération on suit la définition habituelle :

- `p` appartient à $S \cup T$ si `p` appartient à S ou `p` appartient à S,
- `p` appartient à $S \cap T$ si `p` appartient à S et `p` appartient à T,
- `p` appartient à $\mathsf{C}(S)$ si `p` n'appartient pas à S.

Notez que `contains` est la première méthode qui effectue des calculs.

4 toString

Ajouter dans toutes les classes la méthode `toString` appropriée. Les rectangles peuvent être représentés par un String `"??[(a_x,a_y),(c_x,c_y)]"` où `??` le nom de votre classe, à vous de choisir une représentation lisible pour l'union, l'intersection et le complément.

1. `ARS` comme abréviation de « Algèbre de Rectangles Simples «
2. ce qui devait être immédiat, le corps chaque fois c'est juste un `return new Constructeur( paramètres )` où `paramètres` sont les paramètres de la fabrique correspondante

5 Simplifier les expressions de l'algèbre

L'opération de complément satisfait les identités suivantes :

$$\complement(T \cup S) = \complement(T) \cap \complement(S) \quad \text{et} \quad \complement(T \cap S) = \complement(T) \cup \complement(S) \quad \text{et} \quad \complement(\complement(S)) = S \tag{2}$$

On appliquant récursivement ces formules il est possible d'obtenir pour chaque expression une expression équivalentes mais avec le complément qui est appliqué uniquement sur les rectangles. Par exemple

$$\complement((T \cup S) \cap (\complement(W) \cup T)) = (\complement(T) \cap \complement(S)) \cap (W \cap \complement(T)) \tag{3}$$

Ajouter dans `ARS` la méthode `static`

```
1  static ARS simplifyComplement( ARS f )
```

qui pour l'ARS `f` retourne une expression équivalente avec l'opération de complément appliquée uniquement aux rectangles simples. Bien sûr vous devez implémenter cette méthode.

6 Critères d'évaluation

La note finales prend en compte plusieurs critères :

- Sécurité du sous-typage : vous devez faire en sorte qu'il soit impossible de créer d'autres cas/sous-types de `ARS` sans modification de votre code source.
- Immuabilité : les instances de vos classes doivent être immuables (non modifiables) depuis l'extérieur.
- Encapsulation maximale : les membres de classes doivent être inaccessibles depuis l'extérieur, sauf ceux demandés explicitement dans le sujet.
- Utilisation (raisonnée) des briques de construction vues en cours (abstract / final / sealed / enum / record, ...).
- Par défaut, redéfinitions des méthodes par sous-type, ou alors justification (bref commentaire) en cas d'autre choix de structuration du code.
- Programme qui passe les tests fournis. La qualité et l'exhaustivité de vos tests.