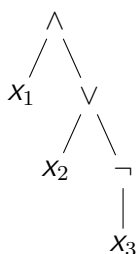


## TP n°1

### Une classe pour les formules propositionnelles

Dans ce TP on va définir une représentation abstraite d'une formule, qu'on utilisera pour l'évaluation.

**Syntaxe abstraite** La syntaxe des formules propositionnelles peut se représenter sous forme d'un arbre qu'on appelle l'*arbre syntaxique*. Par exemple considérons la formule  $(x_1 \wedge (x_2 \vee \neg x_3))$ . On peut représenter cette formule comme suit :



L'arbre a une *racine* (l'opérateur  $\wedge$ ) et deux *sous-arbres*. Le premier sous-arbre (celui de gauche) est la variable  $x$  (une *feuille*). Le deuxième sous-arbre (celui de droite) a l'opérateur  $\vee$  comme racine et ainsi de suite.

Les arbres qui ont  $\wedge$  ou  $\vee$  comme racine ont toujours deux sous-arbre. Les arbres qui ont  $\neg$  comme racine ont un sous-arbre. Les variables sont toujours des feuilles.

On construira une classe `Formule` qui représente l'arbre syntaxique d'une formule. Un objet de la classe aura un champ qui dit s'il s'agit d'une variable, d'une négation, d'une conjonction ou d'une disjonction.

**Exercice 1** Créez une classe `Formule`, avec un champ entier `opPrinc`. Ce champ vaudra 0 pour une variable, 1 pour une négation, 2 pour une conjonction, 3 pour une disjonction. Pour ne pas avoir à se rappeler cette convention, définissez les variables statiques suivantes :

```
public static final int VAR = 0;
public static final int NEG = 1;
public static final int CONJ = 2;
public static final int DISJ = 3;
```

**Exercice 2** Écrivez les méthodes suivantes :

- { `isNegation()` qui renvoie `true` si la formule est une négation.
- { `isConjonction()` qui renvoie `true` si la formule est une conjonction.
- { `isDisjonction()` qui renvoie `true` si la formule est une disjonction.
- { `isVariable()` qui renvoie `true` si la formule est une variable.

Une formule peut avoir une ou deux sous-formules.

**Exercice 3** Ajoutez a la classe `Formule` deux champs prives `sousFormule1` et `sousFormule2` initialement avec valeur `null`. Ecrivez les methodes `getSousFormule1()` et `getSousFormule2()` correspondantes.

On suppose que les variable sont de la forme  $x_n$ . Donc pour connaître une variable, il faut et il su t d'en connaître le numero :

**Exercice 4** Ajoutez a la classe `Formule` un champ prive `numeroDeVariable` initialement avec valeur -1. Ecrivez la methodes `getNumeroDeVariable()`.

On va maintenant ecrire les constructeurs. Pour cela il vaut mieux pouvoir lancer des exceptions. La facon la plus simple de declarer une exception est d'ajouter a notre classe les lignes suivantes

```
class MonException extends RuntimeException {
    public MonException (String s) {super(s);}
}
```

Pour la lancer il faut ecrire

```
throw new MonException("Message");
```

Vous pouvez creer plusieurs exceptions, ou lancer la même avec des messages di erents.

**Exercice 5** { Ecrivez le constructeur `Formule(int op, int numeroDeVariable)` qui cree une variable, si `op` est 0 et si le numero de variable n'est pas negatif, et qui lance une exception sinon.

{ Ecrivez le constructeur `Formule(int op, Formule f1)` qui cree la negation de la formule `f1`, si `op` est 1 et si `f1` n'est pas `null`, et qui lance une exception sinon.

{ Ecrivez le constructeur `Formule(int op, Formule f1, Formule f2)` qui cree la conjonction des formules `f1` et `f2`, si `op` est 2 et si `f1, f2` ne sont pas `null`, qui cree la disjonction des formules `f1` et `f2`, si `op` est 3 et si `f1, f2` ne sont pas `null` , et qui lance une exception sinon.

**Exercice 6** Pour vous familiariser avec cette classe, utilisez les constructeurs de la classe pour creer un objet de la classe representant la formule propositionnelle  $((x_1 \wedge x_1) \vee \neg x_3)$ .

**Exercice 7** Dans la classe `Formule`, implementez une methode `String toString()` qui permette de convertir une formule en une cha ne de caracteres. Si `f` est une formule, `f.toString()` retournera la cha ne correspondante. Au lieu des symboles  $\neg; \wedge; \vee$  on utilisera `non`, `et`, `ou` et au lieu de  $x_1; x_2; \dots; x_1, x_2, \dots$

Par exemple, la formule creee a la question precedente s'a chera sous la forme suivante :  $((x_1 \text{ et } x_1) \text{ ou } \text{non } x_3)$

N'oubliez pas de tester votre methode.

**Exercice 8** (bonus) Ecrire une nouveau constructeur pour la classe `Formule` : `Formule(String op, int[] t)` `op` devra être "conjonction" ou "disjonction", et la formule construite devra être la conjonction ou disjonction des variables indiquees. Par exemple, un appel a

`Formule("conjonction", {1, 2, 3, 5})` creera un objet `Formule` associee a

$((x_1 \text{ et } x_2) \text{ et } x_3) \text{ et } x_5$ ). Le constructeur devra a cher une message d'erreur en cas d'utilisation erronee. A cher des formules creees de cette facon pour tester votre constructeur.