

## Compléments pour le sujet de juin 2003

### Actions

L'action de base indispensable est celle qui permet de réaliser l'affectation d'une valeur à une variable du contexte, permettant ainsi de faire évoluer celui-ci. Appliquer une telle action (méthode `apply()`) reviendra à effectuer l'affectation. Celle-ci est donc définie par l'identificateur de la variable à affecter et la valeur à lui attribuer. On se limitera aux situations où cette valeur est soit l'une des constantes booléennes *vrai* ou *faux*, soit une constante numérique (un entier).

**Q 1.** Définissez une classe `Affectation` qui implémente `Action` et correspond à cette affectation.

**Q 2.** Définissez une classe `ActionFactory` qui appliquera le design pattern singleton et qui dispose d'une méthode :

```
public Action build(String s) throws MalformedActionException
```

qui à partir de la chaîne `s` construit une action. `s` vérifiera l'une des syntaxes suivantes :

- "`varBool`" correspond à affecter la valeur *vrai* à la variable d'identificateur `varBool`
- "`non varBool`" correspond à affecter la valeur *faux* à la variable d'identificateur `varBool`
- "`varNum = n`" correspond à affecter la valeur entière *n* à la variable d'identificateur `varNum`

### Règle

Une règle (instance de la classe `Rule`) est définie par une condition qui est de type `BooleanExpression` et une action qui est de type `Action`. Sur le principe des factories pour les expressions booléennes et actions, créez une classe `RuleFactory` qui disposera d'une méthode :

```
public Rule build(String s) throws MalformedBooleanExpression, MalformedActionException
```

la chaîne devra définir la syntaxe "*si condition alors action*" où *condition* sera conforme à la syntaxe des expressions booléennes et *action* à celle des actions.

### Base de règles

Une base de règles est constitué d'une collection de règles, pour la "remplir" vous coderez les deux méthodes suivantes :

```
public void addRule(Rule rule)
public void addRule(String rule)
```

la seconde ajoutant la règle qui correspond à la chaîne de caractères passée en paramètre (via la factory), si cette chaîne est mal formée, il ne se passe rien.

## Tests

Ci-dessous trois petits exemples (disponibles depuis le portail), que vous pouvez et devez librement et facilement adapter pour réaliser vos tests. Voici deux de ces exemples.

```
package ... ;

public class TestAnimaux {
    public static void main(String[] args) {
        RuleBase ruleBase = new RuleBase();
        Context ctxt = new Context();

        ruleBase.add("si a_poils alors mammifere");
        ruleBase.add("si donne_lait alors mammifere");
        ruleBase.add("si a_plumes alors oiseau");
        ruleBase.add("si vole et ponds_oeufs alors oiseau");
        ruleBase.add("si mange_viande alors carnivore");
        ruleBase.add("si a_dents_pointues et a_griffes et a_yeux_devant alors carnivore");
        ruleBase.add("si mammifere et a_sabots alors ongule");
        ruleBase.add("si mammifere et rumine alors ongule");
        ruleBase.add("si mammifere et carnivore et a_couleur_brune et a_taches_sombres alors guepard");
        ruleBase.add("si mammifere et carnivore et a_couleur_brune et a_raies_noires alors tigre");
        ruleBase.add("si ongule et a_long_cou et a_longues_pattes et a_taches_sombres alors girafe");
        ruleBase.add("si ongule et a_raies_noires alors zebre");
        ruleBase.add("si oiseau et non vole et a_long_cou et a_longues_pattes et noir_et_blanc alors autruche");
        ruleBase.add("si oiseau et non vole et nage et noir_et_blanc alors pingouin");
        ruleBase.add("si oiseau et vole_bien alors albatros");

        ctxt.addVariable("a_poils", new BooleanValue(BooleanConstant.VRAI));
        ctxt.addVariable("vole", new BooleanValue(BooleanConstant.FAUX));
        ctxt.addVariable("rumine", new BooleanValue(BooleanConstant.VRAI));
        ctxt.addVariable("a_long_cou", new BooleanValue(BooleanConstant.FAUX));
        ctxt.addVariable("a_raies_noires", new BooleanValue(BooleanConstant.VRAI));

        ruleBase.chainageAvant(ctxt);

        ctxt.afficheTout(); // affiche toutes les variables connues avec leur valeur
    }
}
```

```
package ... ;

public class TestRiche {
    public static void main(String[] args) {
        RuleBase ruleBase = new RuleBase();
        Context ctxt = new Context();

        ruleBase.add("si malhonnete et fort alors riche");
        ruleBase.add("si parents_riches et intelligent alors riche");
        ruleBase.add("si travailleur et intelligent alors riche");
        ruleBase.add("si fortune > 1000000 alors riche");
        ruleBase.add("si habite_chateau alors riche");
        ruleBase.add("si medecin alors riche");
        ruleBase.add("si informaticien alors riche");
        ruleBase.add("si fonctionnaire alors pauvre");
        ruleBase.add("si non habite_chateau alors pauvre");
        ruleBase.add("si pauvre alors non riche");
        ruleBase.add("si a_fait_prison alors malhonnete");
        ruleBase.add("si grand et lourd alors fort");
        ruleBase.add("si taille > 185 alors grand");
        ruleBase.add("si poids > 85 alors lourd");
        ruleBase.add("si fortune_parents > 1000000 alors parents_riches");
        ruleBase.add("si travail_par_jour > 8 alors travailleur");

        ctxt.addVariable("fortune_parents", new NumericValue(1234765));
        ctxt.addVariable("intelligent", new BooleanValue(BooleanConstant.VRAI));
        ctxt.addVariable("taille", new NumericValue(190));
        ctxt.addVariable("poids", new NumericValue(82));

        ruleBase.chainageAvant(ctxt);

        ctxt.afficheTout(); // affiche toutes les variables connues avec leur valeur
    }
}
```