Exercices INF3140 : série #9

1. Type immuable `String`

Dans ce qui suit, nous allons supposer que le type `String` n'existe pas en OCL et nous allons le spécifier en utilisant les mêmes méthodes que celles disponibles en Java.

Le module OCL 1 (p. 2) définit partiellement (signatures des méthodes seulement) un tel type `String` pour des chaines semblables à celles en Java. Comme en Java, il s'agit d'un **type immuable**. Dans notre cas, il s'agit aussi d'une spécification *partielle* puisque le vrai type `String` de Java définit 13 constructeurs et plus de 60 méthodes! Nous allons aussi supposer qu'il existe en OCL, comme en Java, un type `Char` — que nous ne définirons pas.

Spécifiez des contrats avec pré/postconditions, dans la notation OCL, pour les opérations indiquées ci-bas — utilisez `true` comme précondition si aucune condition particulière n'est requise.

Rappel : En Java, et donc dans les arguments des méthodes du type `String`, les index débutent à la position 0. Par contre, en OCL, les index d'une valeur de type `Sequence` débutent à 1. Il faut donc faire les calculs d'index appropriés.

a. `charAt( index: Integer ): Char`

b. `concat( str: JString): String`

c. `replace( oldChar: Char, newChar: Char ): String`

d. `startsWith( prefix: String ): Boolean`

2. Type mutable `Map`

Un *dictionnaire* permet d'associer des définition à des clés — comme une fonction. Plus précisément, une clé donnée ne possède, à un instant donné, qu'une seule et unique définition.

On veut spécifier, en OCL, des dictionnaires comme ceux disponibles en Java.

Soit les classes OCL définies dans le module OCL 2, qui décrivent des classes auxiliaires ainsi que la signature des méthodes de la classe `Map`.

a. Quel(s) attribut(s) faudrait-il utiliser pour spécifier l'état d'un `Map`?

b. Quel devrait alors être l'invariant requis?

c. Spécifiez le contrat OCL pour l'opération `get`.

   La documentation Java indique ce qui suit pour cette opération : «*Returns the value to which this map maps the specified key. Returns null if the map contains no mapping for this key.*

   Note : En OCL, on peut «simuler» la valeur `null` de Java avec la valeur `Undefined` ou avec le prédicat `isUndefined`.

```
class String

attributes
  cars: Sequence(Char)

operations
  length(): Integer

  charAt( index: Integer ): Char
  // Returns the char value at the specified index.


  startsWith( prefix: String ): Boolean
  // Tests if this string starts with the specified prefix.


  substring( beginIndex: Integer ): String
  // Returns a new string that is a substring of this string.


  concat( str: String): String
  // Concatenates the specified string to the end of this string.


  replace( oldChar: Char, newChar: Char ): String
  // Returns a new string resulting from replacing all occurrences of
  // oldChar in this string with newChar.
end


class FabriqueString
operations
   String( value: Sequence(Char) ): String
     post:
       result.cars = value
end
```

```
@Test public void exemples() {
    assertEquals( 'c', "abc".charAt( 2 ) );

    assertEquals( "def", "abcdef".substring(3) );

    assertEquals( "abcdef", "abc".concat( "def" ) );

    assertEquals( "aaaac", "abbac".replace( 'b', 'a' ) );
}
```


Module OCL 1: Un type immuable définissant un type `String` semblable à celui en Java, avec une méthode de fabrication pour simuler l'un des constructeurs, et un petit cas de test pour illustrer le comportement des méthodes.

```
class K end  -- Type, generique, pour les cles.
class V end  -- Type, generique, pour les definitions (Value).


class Paire
attributes
  cle: K
  valeur: V
end


class Map

attributes
  ?

operations
  size(): Integer

  containsKey( key: K ): Boolean

  containsValue( value: V ): Boolean

  keySet(): Set(K)

  get( key: K ): V

  put( key: K, value: V ): V

  remove( key: K ): V

  values(): Bag(V)

  clear()
constraints
  ?
end
```

Module OCL 2: Un type mutable OCL spécifiant (partiellement) le type `Map` de Java.