# In-Class Exercises

ETH Zurich

## 1 Contracts

The purpose of this exercise is to practice writing more complicated contracts that can be used for software specification. Your job is to write contracts for the class *SET*[*G*] which is shown in listing 1. The intended function results are given in the comments in mathematical terms.

For example, the comment of function *is_equal* stipulates that **Result** should be *True* if and only if **Current** and *other* represent the same set, and the comment of function *add* specifies the set of **Result** should be equal to the union of the sets of **Current** and the set containing only *element*.

Read through the code, then complete the postconditions so that they reflect the function comments and add a class invariant.

Please note:

- The number of dotted lines is not indicative of the number of missing contract clauses.
- The following features from class *LINKED_LIST*[*E*] may be useful:

```
class interface LINKED_LIST[E]

feature
    count: INTEGER
        -- Number of elements actually stored in the collection .

    empty: BOOLEAN
        -- Is collection empty?

    has (x: like item): BOOLEAN
        -- Look for x using equal for comparison.

    nb_occurrences (element: like item): INTEGER
        -- Number of occurrences of element using equal for comparison.

    -- Other features omitted.
end
```

Listing 1: Class *SET*[*G*]

```
class SET[G]

inherit
    ANY
        redefine is_equal end

create make

feature

    make
        do
            create element_list.make
        end

feature

    is_empty: BOOLEAN
        -- Result = (S_Current = ∅)
        do
        ensure

            .....

            .....

        end

    is_equal (other: like Current): BOOLEAN
        -- Result = (S_Current = S_other)
        require
            other /= Void
        do
        ensure

            .....

            .....

            .....

        end

    has (element: G): BOOLEAN
        -- Result = (element ∈ S_Current)
        do
        ensure

            .....

            .....
```

```
            ..........

        end

    is_subset_of (other: like Current): BOOLEAN
        -- Result = (S_Current ⊂ S_other)
        require
            other /= Void
        do
        ensure

            ..........

            ..........

            ..........

        end

    is_superset_of (other: like Current): BOOLEAN
        -- Result = (S_other ⊂ S_Current)
        require
            other /= Void
        do
        ensure

            ..........

            ..........

            ..........

        end

    count: INTEGER
        -- Result = |S_Current|
        do
        ensure

            ..........

            ..........

        end

feature

    union (other: like Current): like Current
        -- Result = (S_Current ∪ S_other)
        require
            other /= Void
```

```
        do
        ensure
            ..........
            ..........
            ..........
            ..........
        end

    intersection (other: like Current): like Current
        -- Result = (S_Current ∩ S_other)
        require
            other /= Void
        do
        ensure
            ..........
            ..........
            ..........
            ..........
        end

    set_minus (other: like Current): like Current
        -- Result = (S_Current \ S_other)
        require
            other /= Void
        do
        ensure
            ..........
            ..........
            ..........
            ..........
        end

feature

    add (element: G)
        -- S_Current ← (S_Current ∪ {element})
        do
```

```
        ensure
            ..........
            ..........
            ..........
            ..........
        end

    remove (element: G)
        -- S_Current ← (S_Current \ {element})
        do
        ensure
            ..........
            ..........
            ..........
            ..........
        end
feature
    element_list : LINKED_LIST[G]
invariant
            ..........
            ..........
            ..........
            ..........
end
```