Universidade do Porto
Faculdade de Engenharia
FEUP

Mestrado Integrado em Engenharia Informática e Computação

Métodos Formais em Engenharia de Software 2020/21

TP4. Design by contract and verification of object-oriented programs in Dafny (Part I)

1. Stack

Assume the following initial implementation of a Stack with a bounded capacity in Dafny:

```
type T = int // to allow doing new T[capacity], but can be other type

class Stack {
    const elems: array<T>; // immutable (pointer)
    var size : nat; // used size

    constructor (capacity: nat) {
        elems := new T[capacity];
        size := 0;
    }

    predicate method {:verify false} isEmpty() {
        size == 0
    }

    predicate method {:verify false} isFull() {
        size == elems.Length
    }

    method {:verify false} push(x : T) {
        elems[size] := x;
        size := size + 1;
    }

    function method {:verify false} top(): T {
         elems[size-1]
    }

    method {:verify false} pop()  {
         size := size-1;
    }
}

// A simple test case.
method {:verify false} Main() {
    var s := new Stack(3);
    assert s.isEmpty();
    s.push(1);
    s.push(2);
    s.push(3);
    assert s.top() == 3;
    assert s.isFull();
    s.pop();
```

```
    assert s.top() == 2;
    print "top = ", s.top(), " \n";
}
```

a) Compile and run this program and check that the correct result is printed.
b) Add a class invariant (predicate Valid()) to check the validity of the fields values, i.e., that the size of the stack does not exceed the allocated capacity. Add the {:autocontracts} attribute after the "class" keyword, so that Dafny checks automatically the class invariant before & after all methods.
c) Add relevant pre-conditions to methods, constructors, functions, and predicates, and remove the "{:verify false}" attribute.
d) Add post-conditions to methods and constructors describing their intended effect. Notice that functions and predicates don't need post-conditions. At this point, the assertions in the test scenario should be checked successfully by Dafny.

2. Person

Assume the following initial definition of the class Person in Dafny:

```
datatype Sex = Masculine | Feminine
datatype CivilState = Single | Married | Divorced | Widow | Dead

class Person
{
    const name: string; // 'const' for immutable fields
    const sex: Sex;
    const mother: Person?; // '?' to allow null
    const father: Person?;
    var spouse: Person?;
    var civilState: CivilState;

    constructor (name: string, sex: Sex, mother: Person?, father: Person?)
    {
        this.name := name;
        this.sex := sex;
        this.mother := mother;
        this.father := father;
        this.spouse := null;
        this.civilState := Single;
    }

    method marry(spouse: Person)
    {
        spouse.spouse := this;
        spouse.civilState := Married;
        this.spouse := spouse;
        this.civilState := Married;
    }

    method divorce()
    {
        spouse.spouse := null;
        spouse.civilState := Divorced;
        this.spouse := null;
        this.civilState := Divorced;
```

```
    }

    method die()
    {
        if spouse != null
        {
            spouse.spouse := null;
            spouse.civilState := Widow;
        }
        this.spouse := null;
        this.civilState := Dead;
    }
}
```

a) Add a class invariant (predicate Valid()) to check the validity of the fields' values, namely that:
   - only married people have a spouse
   - the mother, if defined, must be a woman (of sex feminine)
   - the father, if defined, must be a man (of sex masculine)
   - a person can only marry another person of the opposite sex
   - the "spouse" relation is symmetric, i.e., "I am the spouse of my spouse"

   Note: In this exercise, don't use the {:autocontracts} attribute.

b) Add relevant pre-conditions and frame clauses (read/write) to all methods and constructors. Don't forget to include appropriate calls to the Valid() predicate. Besides pre-conditions needed to respect the class invariants, there are also restrictions on the civil state transitions allowed.

c) Add post-conditions to methods and constructors describing their intended effect. Don't forget to include appropriate calls to the Valid() predicate.

d) Write and run a valid test scenario, covering all possible state transitions. Use assertions to checks objects' state.

e) Make sure that a person does not marry an ancestor or brother/sister. Suggestion: you may add a ghost field to store all the ancestors of a person.

f) Give also examples of invalid test scenarios, violating operations' pre-conditions.