Duck typing issues

Alexey V. Kanatov Apr 21st 2022

“If it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck.”

The key word here is **PROBABLY** and I will try to explain why. Any object has the type it belongs to. Type describes attributes (fields, data elements) and operations (routines – procedures and functions) every object of this type has and …type invariant as well as operations’ preconditions and postconditions. Now let’s see

```
type A
        foo (p1: T1)
                require pre1
                ensure post1
        invariant inv1
end
```

```
type B
        foo (p1: T2)
                require pre2
                ensure post2
        invariant inv2
end
```

So, can we say that type A and B define the same class objects? So, if pre1 is identical to pre2 and post1 is identical to post 2 and the same holds for inv1 and inv2 then we may state that these are ‘ducks’. Or if there is a relation between these predicates (one implies another) then we may agree that assignment of object of type A into object of type B be Ok (type safe) or vice versa.

```
a: A = new B // case 1
```

```
b: B = new A // case 2
```

Another aspect is conformance of signatures of all operations exported to clients

```
a.foo (expression1) //case 1
```

```
b.foo (expression2) // case 2
```

So, type T1 should conform to T2 for case 1 and vice versa for case 2.

It effectively means that both cases 1 and 2 require a lot of very complicated tests to be performed to allow such kind of polymorphic assignments.

Another application of duck typing is dynamic type test and then accessing the member of the object

```
if a is type B then
        a.foo (expression_of_type_T2) // We know that type of a can be treated as B
end
```

Again we need to check at runtime that type of a has proper invariant and all call to operations are being done in accordance with their pre and post conditions

It implies the fact that duck typing in any form (assignment or dynamic type test) does not work together with software verification elements based on systematic assertions usage. So, inheritance works well with assertions and having its drawback of coupling between parent and child types it sets the solid foundation for the design and development of the reliable software