Bordeaux 1
Year 2015-2016

Master ESSV
MtoCPL

TD n°5 - Subtyping polymorphism

Exercice 1: Constrained polymorphism in OCaml

The OCaml language possesses constructs to define objects and classes[1]. For instance, the following code defines a standalone object :

```
let cpt = object
    val mutable value = 0
    method get_value = value
    method incr n = value ← value + n
  end;;
cpt#incr 123;;
```

1. How is the type of an object defined ?
2. Write another object with the same type, but different internal state, and check that they have the same type, for example by putting them into a list.

OCaml objects may be defined with classes. In OCaml, a class is simply a constructor function for the object, using the `class` keyword instead of `let` :

```
class counter init_value = object
  val mutable value = init_value
  method get_value = value
  method incr n = value ← value + n
end;;
let cpt = new counter 456;;
```

3. Check the type of the class and compare it to the type of the object it constructs.

Class inheritance is done via the `inherit` keyword and the coercion operator is written `:>` :

```
class resettable_counter init_value = object
   inherit counter init_value
   method reset = value ← 0
end;;
let cqt = ((new resettable_counter 789) :> counter);;
```

[1] The code given here is sufficient for the purpose of the exercise. Additional information can be found at http://caml.inria.fr/pub/docs/u3-ocaml/ocaml-objects.html.

4. What is the OCaml policy with regard to upcasts and downcasts ?
   How would you qualify such a policy ?

Imagine that you want to write a generic functions on counters that tests whether two counters possess the same value. At first, this function should not be a method.

5. What is the type inferred by OCaml for such a function ?
6. Is it possible to write this function as a method for the `counter` class ? What kind of problem appears ?

▷ C# is a programming language developped by Microsoft (http://msdn.microsoft.com/en-us/vcsharp/aa336809.aspx), that possesses an open-source compiler and runtime environment within project Mono (http://www.mono-project.com). In order to programm in C# with Mono, one must use the following tools :

- Compilation : use the `mcs` compiler in the following manner to obtain an `.exe` file from a `.cs` source code file :

  `gmcs "file.cs"`

- Execution : use the runtime environment in the following manner :

  `mono "file.exe"`

Here is an example of a very simple C# program :

```
using System;
class Program {
  public static void Main() {
    Console.WriteLine("Caramba !");
} }
```

Exercice 2: Subtyping polymorphism in Java and C#

Let us compare the capabilities of Java and C# with regard to the subtyping polymorphism. A small hierarchy of classes is available in the sources for both languages, and is used for the following examples.

Observe the following snippet of code in Java :

```java
class P implements Comparable<P> {
    public Integer x;
    P(Integer x) { this.x = x; }
    public int compareTo(P other) {
        return (x.compareTo(other.x));
    }
}
class SubP extends P {
    public Double y;
    SubP(Integer x, Double d) { super(x); this.y = d; }
    public int compareTo(SubP other) {
        if (x.compareTo(other.x) != 0)
            return (x.compareTo(other.x));
        else
            return (y.compareTo(other.y));
    }
}
```

Both classes `P` and `SubP` implement the `Comparable` interface.

1. What kind of relations are expected from an equality or comparison function ?
2. Are these relations respected in this situation ? More concretely, is it possible to create instances of `P` and `SubP` that are different but equal under the `compareTo` method ? Try without upcasts and then with upcasts.
3. What happens when trying to upcast or downcast a variable in an ill-typed manner ? What is the error message, if any, and when does it occur ?

   What does it imply in terms of type safety ?

Covariance and *contravariance* are properties that characterize the type compatibilities for values occurring in a "type position" : the type of the parameter of a function, the type of its return value, the type parameter of a generic function …

4. Recall the rules with regard to covariance and contravariance for function parameters and function results.
5. How do Java or C# behave with regard to these rules ?
   (replacing the term "function" by "method")
6. Write an example function for each case of covariance and contravariance.

▷ Java and C# propose two different ways to define constraints on the variables :

- Java/ Use-site variance : the variance indication on the type variable appears in the method using the variable

```
class Collections<T> { .. };
public void copy(List <? super T> dst, List <? extends T> src) {
.. };
```

- C#/ Definition-site variance : the variance indication on the type variable appears at the definition of the class

```
interface Collections<in T> {
  T getElem()
.. };
```

In both cases, the code must satisfy the *get-put* principle for a generic class `C<T>` :

- if the type variable `T` is in a *covariant* position, one can call every method in `C<T>` except the generic methods having an *argument* of type `T`.
- if the type variable `T` is in a *contravariant* position, one can call every method in `C<T>` except the generic methods having a *result* of type `T`.

Exercice 3: Type variance
In languages with generics, the `Collections` and their containers usually provide nice examples of parameterized types.

1. What is in general (that is to say without taking a specific language into account) the rule of variance for a parameterized type such as `Container<Parameter>` ?
2. Verify your proposition by writing examples in Java and C#. What is the error message, if any, and when does it occur ?

According to the previous questions, if ever an `Orange` inherits from a class `Fruit`, then it implies that there is no relation whatsoever between `List<Orange>` and `List<Fruit>`, which is very restrictive.

3. Give an example that shows how restrictive this choice is.

Now there is a possibility to alleviate this behavior with variance indications. Type compatibility is explicitly stated, either at the definition of the variables (*definition-site variance*, for example in C# and Scala) or at the place they are used, usually the arguments of functions (*use-site variance*, for example in Java via the use of *wildcards*).

4. Consult the Java 1.7 documentation page for the `Collections` class (`http://docs.oracle.com/javase/7/docs/api/java/util/Collections.html`), and discuss the types of the functions `fill`, `copy` and `max`.

5. Consult the C# documentation for the `IEnumerable` interface `http://msdn.microsoft.com/en-us/library/vstudio/9eekhta0%28v=vs.100%29.aspx` and discuss the differences with the Java case, in particular for the `max` functions.

For these specific types, there is a general principle explaining what kind of methods can or cannot be applied to a wildcard type. For each of the following citations[2], construct a Java example that does not type-check, and explain why it doesn't.

6. About covariance :

   > ▷ For example the type `List<? extends Number>` is often used to indicate read-only lists of `Number`s. This is because one can get elements of the list and statically know they are `Number`s, but one cannot add `Number`s to the list since the list may actually represent a `List<Integer>` which does not accept arbitrary `Number`s.

7. About contravariance :

   > ▷ Similarly, `List<? super Number>` is often used to indicate write-only lists. This time, one cannot get `Number`s from the list since it may actually be a `List<Object>`, but one can add `Number`s to the list.

Let us write a generic function in to sort elements of a list.

8. Write the prototype of a generic sorting function in C# that takes a list and a comparison function, that is covariant in the list parameter and contravariant in the other.

9. Write the prototype of a sorting algorithm in Java that is equivalent to the previous one. The functional comparison can be replaced by the `Comparator` interface.

   What is the difference with the C# solution ?

10. Suppose that we only require that the type `T` implements the `Comparable` interface. What is the most generic prototype you can write ?

[2]Taken from `http://www.cs.cornell.edu/~ross/publications/tamewild`