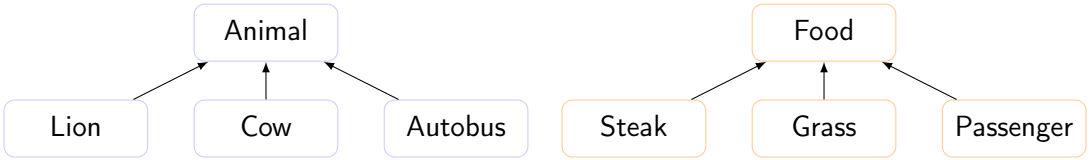
ENSEIRB-Matmeca
Year 2024-2025

IF Course
IT234

Tutorial #6 - Subtyping and Variance

Exercice 1: Who eats whom ?
Consider the following programming problem : two hierarchies of classes, that interact with one another via a method conveniently called `eats` :




```
class Food {
  public String toString() { return "some generic food, maybe tofu"; }
}

class Animal {
  void eats(Food v) { System.out.println("A common animal, say a platypus, eating " + v); }
}
```

More than certainly, some animals are not made to eat every possible kind of food. Consider the problem of relating an Animal to its Food in a one-to-one correspondence, inside a program (in this exercise, the Autobus's diet is considered to be restricted to its one and only favorite Food).

1. Pick a concrete Animal and write the necessary code for him to eat a value of type Food on the one hand, and a value whose type corresponds to its dedicated Food on the other hand.
2. What kinds of polymorphism are at play here ?
3. What would it take for a call to `Animal.eats(Food)` to call the function on the dedicated food instead of the generic one ?
   At which time of the execution of a program would a type error be caught ?

▷ Scala is a programming language designed by Martin Odersky developed at the EPFL (`http://www.scala-lang.org`), that can be compiled to Java bytecode and run on the Java virtual machine. In order to program in Scala, one follows the these steps :

- Compilation : use the `scalac` compiler to transform a `.scala` file into a series of `.class` files :

  `scalac file.scala`

- Execution : use the virtual machine in the following way :

  `scala file`

  The virtual machine `scala`, when launched alone, can be used as a REPL for the Scala language.

In order to add the executables into your path, it suffices to apply the following command : `export PATH=$PATH:/net/ens/renault/scala/bin`. Here follows a very simple example of Scala program :

```
object HelloWorld {
  def main(args: Array[String]) {
    println("Chapito les tepos")
  } // This is a comment
}
```

Exercice 2: Definition-site variance in Scala

In Scala, parametric classes may contain a variance annotation, at the point of definition of the class. One speaks of *definition-site* variance. The following examples are taken from the Scala standard library :

```
abstract class List[+A]    // Covariant in A
trait Function1[-T1, +R]   // Contravariant in T1
```

Further examples may be found by browsing the Scala standard library documentation available at `http://www.scala-lang.org/api/current/index.html`.

1. What is the biggest difference in terms of variance between the classes in the `scala.collection.immutable` hierarchy and the `scala.collection.mutable` ones ?
2. Knowing that `List[+A]<:Seq[+A]`, and that `Integer <: Number`, what are the subtype relations between the four following types :

   `Seq[Integer]`, `Seq[Number]`, `List[Integer]`, and `List[Number]` ?

In the Scala REPL, the following lines allow the creation of an `Array` and a `List` :

```scala
var a = Array(1,2,3)    // scala.collection.mutable
a(0)                    // → 1 (index accessor)
var l = List(4,5,6)     // scala.collection.immutable
a(0)                    // → 4 (index accessor)
```

In order to cast a value in Scala, it suffices to use the method `asInstanceOf` in the following manner : `l.asInstanceOf[List[Any]]`

3. What collection allows the modification of its contents ? What collection allows a cast changing the type of its contents ? Why is this exclusive ?
4. How do you add an element (for example a `String`) into each collection ?
   How does this modify the type of the collection ?
   Is this coherent with the previous question ?

In Scala, it is possible to apply a `map` to a collection using the following syntax :

```scala
l.map((x) ⇒ if (x >= 2) 10 else 0)  // → Array(0, 10, 10)
```

5. What is the type of the `map` function ? What is its « full signature » ?
6. What are the results of the following calls ?
   Are they coherent with the type of `map` found previously ?

```scala
import scala.collection.immutable.WrappedString
val s1 : WrappedString = "abc" // s1 is a Java String in Scala
s1.map ((x) ⇒ (x.toInt+1))
val s2 : Set[Int] = Set(1,2,3) // s2 is a set without repetition
s2.map((x) ⇒ x % 2) // % is modulo
```

Exercice 3: Use-site variance in Java

To shed some more light on the problem of having multiple references with different types on the same object (a form of *aliasing*), compile the following Java example :

```java
class CovariantArray {
    public static void main(String[] args) {
        String[] strings = new String[1];
        Object[] objects = strings;       // Arrays are covariant
        objects[0] = new Integer(1);      // Runtime failure
    }}
```

1. What operation is possible in Java, that was impossible in Scala ?

In Java, the variance annotations are different from the Scala and C# systems. They do not appear at the point of definition of the parameter, but where it is used. One speaks of *use-site* variance. The following examples are taken from the Java standard

library, more particularly the `Collections` class (`https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html`) :

```
class Collections<T> { .. };
public <T> void copy(List<? super T> dst, List<? extends T> src);
public <T> void fill(List<? super T> list, T obj);
public <T> T    max(Collection<? extends T> coll, Comparator<? super T> comp);
```

The « `?` » character is called a *wildcard*, and depending on the case, it allows different sorts of instanciations :

— An annotation `? extends T` at the place of a type variable means that this variable can be instantiated by any *subclass* of `T`.

— An annotation `? super T` at the place of a type variable means that this variable can be instantiated by any *superclass* of `T`.

2. In what sense are these annotations related to the usual definition of variance ?
3. Give a simplification of the following types :

```
public <T> void concat(Collection<? extends T> l1, Collection <? extends T> l2);
public <T> void iter(Collection<? extends T> l, Consumer<? super T> f);
```

For these « wildcard » 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 [1], construct a Java example that does not type-check, and note the compiler error message.

4. 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.

5. 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.

Consider the `sort` functions of the framework :

1. Citations taken from *Taming Wildcards in Java's Type System* from Tate et. al., available at `http://www.cs.cornell.edu/~ross/publications/tamewild`

```
public <T extends Comparable<? super T>> void sort(List<T> list)
public <T>                               void sort(List<T> list, Comparator<? super T> c)
```

6. In a language with definition-site variance, possessing appropriately typed List[·] and Comparator[·] type functions, write the prototype of a sort function that is as expressive as in Java.

Exercice 4: Binary methods

In this exercise, let us consider binary methods possessing two arguments of the same type. In object-oriented programming, where the first arguments of methods is implicitly the type of `self` : T, these methods are those which take (at least) one parameter of type T.

1. Give an example of binary method appearing in the class `Object` in Java.
2. Write the type of such a method as a logic formula.

These methods become problematic when the implementation must be overloaded depending on the type of the parameters. This is naturally impossible using a purely universal polymorphism. In their article *On Binary Methods* (available at `http://lucacardelli.name/Papers/Binary.pdf`), Cardelli et. al. analyze the problem and discuss possible solutions. They consider (an equivalent of) the following Java class :

```
public class Point {
    private final int x;
    private final int y;

    public Point(int x, int y) {
     this.x = x;
     this.y = y;
    }

    public int getX() { return x; }
    public int getY() { return y; }

    public boolean equals(Object o) {
     if (!(o instanceof Point))
         return false;
     Point p = (Point) o;
     return p.x == x && p.y == y;
    }
}
```

3. Propose an extension of the `Point` class as a `ColoredPoint`, that provides an equality function for all pairs of `Point`s.

   In what measure does your extension achieve the following objectives :

   (i) The extension is independent of the `Point` class.

   (ii) The extension reuses the code of the `Point` class whenever possible.

(iii) The classical axioms for equality hold for the `equal` function.

In their article, Cardelli et. al. propose section 4.2 a solution with multi-methods.

5. Explain what are multi-methods, and how they provide a solution for the problem. What are its downsides ?