

# Exercise 7

## Parametric polymorphism

1) Implement a list in Java or C# with two methods:

- `public void add(int i, Object el)`
- `public Object get(int i)`

Implement it:

- using only one class without generics
- using one abstract class/interface and then (some) subclasses that implement it for different types
- using generic types

Discuss the advantages and the limits of the three different approaches.

2) You have the following Scala class:

```
class Matrix[T] (val m : Array[Array[T]]) {  
  def get(i : int, j : int) : T = {return m(i)(j);}   
  def set(i : int, j : int, elem : T) : Unit = {  
    m(i)(j)=elem;  
  }  
}
```

- What happens if we try to use covariant type arguments on T (that is, `Matrix[+T]`)? Why?
- And what about contravariance (`Matrix[-T]`)?
- We can add two generics (one with covariant and the other one with contravariant annotation) to class `Matrix` (one for positive and the other one for negative positions).  

```
class Matrix[T, +T1 >: T, -T2 <: T] (val m :  
Array[Array[T]]) {  
  def get(i : int, j : int) : T1 = {return m(i)(j);}   
  def set(i : int, j : int, elem : T2) : Unit = {  
    m(i)(j)=elem;  
  }  
}
```

In practice, is this more flexible than the initial solution?

3) Consider the following Scala code:

```
class A[-T]  
class B[... T] {  
  def m(in : A[T]) : int = {...}  
}
```

We want to annotate the generic type of B. Must we use a covariant or a contravariant annotation of the generic type of B? Why? Justify your answer with an example.

## Concepts of Object-Oriented Programming

4) Consider the following Scala classes:

```
class A
class B extends A
class P1[+T]
class P2[T <: A]
```

- What are the possible instantiations of P1 and P2?
- What is the difference between P1[A] and P2[A] from the perspective of a client? Provide an example to show which class is more restrictive.

5) Consider the following Java method:

```
public void add(Object value, List<?> list) {
    list.add(value);
}
```

The Java compiler rejects this program, with the following message:

The method add(capture#1-of ?) in the type List<capture#1-of ?> is not applicable for the arguments (Object) List.java

- Explain why we obtain such an error.
- Fix the program by using a generic type for the parameter of method add and constraining the wildcard appropriately.
- We can use the following alternative signature for add:  
public <V> void add(V value, List<V> list)  
Is this solution more restricted than the one obtained using the wildcard?
- Consider the following methods:

```
public <V> void addAll(List<V> v, List<? super V> l) {
    for(V el : v)
        l.add(el);
}
public <V> void addAll1(List<V> v, List<V> l) {
    for(V el : v)
        l.add(el);
}
```

Method addAll is less restrictive than addAll1. Provide an example to prove this claim.

6) Consider the following Java method:

```
String concatenate(List<Object> list) {
    String result="";
    String separator="";
    if(list instanceof List<String>) {
        result="String:";
        separator=" ";
    }
    else if(list instanceof List<Integer>) {
```

## Concepts of Object-Oriented Programming

```
        result="Integers:";
        separator="+";
    }
    for(Object el : list)
        result=result+separator+el.toString();
    return result;
}
```

- This program is rejected by Java compiler. Why?
- Using the advice given by the Java compiler, rewrite and compile the program. What are the results of the executing the method passing each of the following:
  - A list of strings containing only one element "word"?
  - A list of Integers containing only one element Integer(1)?
  - A list of Objects containing only one element (initialized by new Object())?
- Is this behavior consistent with what you would expect from the initial program? If not, how can you fix it?
- What happens if you compile and execute the initial program in C#? Why?

7) Consider the following class relations and the definition of method `foo`

```
class A
class B extends A
class C extends B

B foo(List<? super B> list1, List<? extends B> list2) {
    list1.add(0, list2.get(0));
    return list2.get(0);
}
```

in which the signatures of the methods of `List<T>` are

```
public void add(int index, T value) {...}
public T get(int index) {...}
```

Can the method body be typechecked with respect to the method signature?

- 8) Java does not allow lower bounds for type parameters. Because of erasure the runtime environment would not be able to check if an object respects a lower bound. On the other hand, Scala allows lower bounds for type parameters, even though it has exactly the same runtime limits of Java, since it is compiled to Java bytecode. Discuss why this approach is inherently unsafe and provide an example to support your claims.