

## Exercise Sheet 6

### Assignment 6.1 Dispatching in Java I

Given this Program:

```
class A {
    public static void p (Object o) { System.out.println(o); }
    public void m1 (A a) { p("m1(A) in A"); }
    public void m1 () { m1(new B()); }
    public void m2 (A a) { p("m2(A) in A"); }
    public void m2 () { m2(this); }
    public void m3 (B b) { m2(this); }
}
class B extends A {
    public void m1 (B b) { p("m1(B) in B"); }
    public void m2 (A a) { p("m2(A) in B"); }
    public void m2 (B b) { p("m2(B) in B"); }
    public void m3 (B b) { super.m1(b); }
}
```

What is the output of the following statements?

1. A a = new B(); a.m1((B)a);

.....

2. A a = new B(); ((B)a).m1(a);

.....

3. A a = new B(); ((B)a).m1((B)a);

.....

4. B b = new B(); b.m1();

.....

5. B b = new B(); b.m2();

.....

6. A b = new B(); b.m2();

.....

7. A a = new B(); a.m3((B)a);

.....

## Assignment 6.2 Dispatching in Java II

Given this Java program:

```
interface I {
    default void hoo (B b) { A.p("hoo(B) in I"); }
}
class A {
    public static void p (Object o) { System.out.println(o); }
    public void foo (A a, B b) { p("foo(A,B) in A"); }
    public void goo (B b) { p("goo(B) in A"); }
    public void hoo (B b) { p("hoo(B) in A"); }
}
class B extends A {
    public void foo (B b, A a) { p("foo(B,A) in B"); }
    public void goo (A a) { p("goo(A) in B"); }
    public void hoo (B b) { p("hoo(B) in B"); }
}
class C extends A implements I {
    public void hoo (B b) { p("hoo(B) in C"); }
}
```

Can the following statements be compiled to code? What is their output at runtime?

1. B b = new B(); b.hoo(b);

.....

2. B b = new B(){ public void hoo(B b){ super.hoo(b);} }; b.hoo(b);

.....

3. B b = new B(); ((A)b).hoo(b);

.....

4. B b = new B(); ((A)b).goo((A)b);

.....

5. B b = new B(); b.goo(b);

.....

6. B b = new B(); b.foo(b,b);

.....

7. B b = new B(); ((A)b).foo(b,b);

.....

8. I i = new C(); i.hoo(new B());

.....

### Assignment 6.3 Visitor Pattern: Single vs. Multiple Dispatching

1. Familiarize yourself with the *Visitor Pattern* which is used in `ExpJava.java` by the expression evaluation visitor `EvalVisitor` (comprising constants, sum and products) written in Java.
2. Implement such a expression evaluation in Perl6 (<https://perl6.org/>) taking advantage of the multiple dispatch functionality (<https://docs.perl6.org/syntax/multi>). The class structure should stay the same. Thus, (`Const` and `BinExp` are subclasses of `Exp` (no abstract class); and `Sum` and `Prod` are subclasses of `BinExp`. Implement evaluation functions `eval` that return the result of a given expression `Eval`.