

# Multimethods in boo

These examples demonstrate the use of duck typing to implement [multimethods](#) (also called [multiple dispatch](#)). With multimethods, the decision about which method to call is deferred until runtime based on the actual runtime types of the arguments to the method. The Dylan and Nice programming languages implement it by default, boo lets the programmer decide when to use it via duck typing.

Multimethods are useful when the correct method to call cannot be determined at compile time because the runtime argument types are not known. Two examples are shown - multiple dispatch of static methods and multiple dispatch of instance methods.

The inspiration for the first example was taken from [Visitor Pattern Versus Multimethods](#) and demonstrates multiple dispatch of a static class method.

```
abstract class Expression:
    pass

class IntExp(Expression):
    public value as int

class AddExp(Expression):
    public e1 as Expression
    public e2 as Expression

[Module]
class PP:
    static def prettyPrint(e as IntExp):
        print e.value

    static def prettyPrint(e as AddExp):
        // Use duck typing to delay
determination of which method to call until
runtime
        // because at compile time this call
is ambiguous
        (PP as duck).prettyPrint(e.e1)
        print " + "
        (PP as duck).prettyPrint(e.e2)

a = IntExp(value: 3)
b = IntExp(value: 4)
e = AddExp(e1: a, e2: b)
PP.prettyPrint(e)
```

The next example shows how a class which collects integers (`IntCollector`) can be extended with minimal effort to sift through lists too (`MyListIntCollector`) and demonstrates multiple dispatch of an instance method.

```

class IntCollector:
    stuff = []
    def collect(o as object):
        pass
    def collect(i as int):
        stuff.AddUnique(i)
    def collect(i as double):
        stuff.AddUnique(cast(int,
System.Math.Round(i)))
    def dump():
        for s in stuff:
            print s

class MyListIntCollector(IntCollector):
    def collect(a as List):
        for o in a:
            // without multiple dispatch
this will call collect(o as object)
            // with multiple dispatch the
correct method will be called based on the
type of o
            (self as duck).collect(o)

s = MyListIntCollector()
s.collect(10)
s.collect(12.1)
s.collect("hi")
s.collect([1,2,3,"hello",3.1416,3.8])
s.dump()

```

The output of s.dump() is:

**10**

**12**

**1**

**2**

**3**

**4**