Casting Types

Type casting in boo works the same as in C# (see this article). The only difference is that with explicit casts in boo you use the "cast" operator (see below).

You can try to cast one type to another type 3 ways: the "as" statement, an explicit cast, or an implicit cast. Here are examples:

```
//The "as" cast silently returns null on failure
yy = "123" as (int) //try to "as" cast a numeric string
//It cannot, thus yy is null:
if yy is null:
    print "yy is null"

//Explicit casts use the cast operator.
numb = 123.4
zz = numb cast int

//For implicit casts you declare the type of the left-hand variable,
// and then try to assign an object to that variable:
aa as int = 123.4
```

Don't use "as" cast for value types

One thing that might be misleading is that with the "as" cast that returns null if the cast fails, you cannot use that with value types (like ints, enums, etc.). Why? Because value types cannot be null (You can't have a null integer, for example. Integers have a default value of 0). The above example might be misleading because it uses the "as" cast for an *array* of ints. Arrays are passed by reference, not by value.

So in those cases when you have value types, you would use the explicit or implicit casts instead. This is how it works in C#, too.

Using "isa" for checking types before casting

Sometimes you may want to check an object's type first before attempting a cast. You can use the "isa" operator:

```
for item in [1, 2.0, "three"]:
 if item isa int:
  print item, "is an integer"
  print (item cast int + 3)
 elif item isa double:
  print item, "is a double"
  print (item cast double / 0.33)
 elif item isa string:
  print item, "is a string"
  //a string is not a value type so you can do
  //the "as" cast if you want
  print ((item as string).ToUpper())
```

Use -ducky option or duck types for quick prototyping

See Duck Typing. Recent versions of boo add a -ducky option to the compiler that implicitly casts an object to type "duck" when the type cannot be inferred. What this does is hold off type resolution until runtime. This might be useful when you are just testing out some code.

With the -ducky option set, the above example could be coded more simply without the casts:

```
for item in [1, 2.0, "three"]:
 if item isa int:
  print item + 3
 elif item isa double:
  print item / 0.33
 elif item isa string:
  print item.ToUpper()
```

It does make your code run slower though, so you'd want to later turn off the ducky option and fill in the appropriate casts as needed.