



## Programming Parrot: Using Objects

Jonathan Scott Duff  
duff@pobox.com

Yes, you've read correctly. Parrot can create and manipulate objects (aka, object oriented programming). While it may seem strange for a low-level language like PIR to have the facility for object-oriented programming, it makes perfect sense in this particular case.

Parrot's number one goal is to be the underlying implementation for Perl6, which has object oriented features. Parrot's secondary goal is to provide a good platform for other dynamic languages such as Python, Ruby, PHP, Javascript, etc. and those languages also have the ability (if not the requirement) to be object-oriented. Thus Parrot has facilities for a manipulating objects so that language implementors can easily express the appropriate object semantics for their language of interest.

### Namespaces

Before I begin talking about how to create classes and instantiate objects, I first need to talk about an intimately related subject: namespaces. Namespaces serve a two-fold purpose; they allow me to group related routines together and they allow me to give several subroutines the same name but different, domain-specific implementations. These characteristics are, oddly enough, similar to the basic requirements for a class.

For instance, I can put all of my subroutines dealing with people in a `Person` namespace and all of my subroutines dealing with computer programs in

the `Process` namespace. Both namespaces may have a subroutine named `run()` but with radically different implementations. This code to illustrate this example:

```
.namespace [ "Person" ]

.sub run
    print "Run Forrest, Run!\n"
.end

.namespace [ "Process" ]

.sub run
    print "Running process #53\n"
.end
```

As you may have guessed, the `.namespace` directive tells Parrot which namespace to group subroutines under. A namespace ends when another `.namespace` directive changes the namespace or when it reaches the end of the file. A bare `.namespace` directive (*i.e.*, with no name following it) changes back to the default namespace.

Perl programmers will recognize that Parrot `.namespace` declarations are just like Perl `package` declarations. But there are a few differences. I'll talk more about how Parrot uses namespaces and classes together in just a minute.

### PIR with class

Creating classes in Parrot is relatively easy since there are opcodes for it. The easiest to start with is `newclass`; I just say

```
$P0 = newclass 'Foo'
```

where `$P0` can be any PMC thing (a register or a local variable) and `'Foo'` is the name of the class I want to create.

When you wish to instantiate objects that belong to the class you've created, it's equally simple. I just say:

*Jonathan also wrote "Parrot Magic Cookies" for TPR 3.0 (Fall 2006), following up on Alberto Simões's "Programming Parrot" in TPR 2.3 (Summer 2006). When TPR was only a digital edition, Dan Sugalski wrote "Parrot Bits" for TPR 0.1 (March 2002) and TPR 0.3 (May 2002). Clinton Pierce wrote about implementing BASIC on Parrot for TPR 0.4 (July 2002).*