Object Oriented Programming

A Perl class is just a package, the terms are interchangeable

In OOP terms a class usually provides a method for creating multiple objects of the class as well as methods for defining the behavior of each object created

- A “new” subroutine to create a new object
- Accessors to get the values from the object
- Mutators to set the values in the object

Creating the “class”

Create a new package

Remember that all variables are global unless declared otherwise regardless of where they are physically declared

Create a subroutine named **new**. This subroutine will create the new object. The subroutine can be called anything but is usually called new to stay consistent with Java and C++ where it is mandatory

- This will be the constructor method for the class

New subroutine

The new subroutine will require at least one argument. The first, and potentially only argument, must be the package name

The new subroutine creates a reference to an anonymous hash and then fills the hash with the parameters

```
$className = $_[0];  #or $className = shift @_
my $ref = {} #a reference to an empty anonymous hash
```

Additional arguments can be passed to set values in the object

```
$ref->{‘hashKey’} = $_[1];  #set the hash key to the
  second argument
$ref->{‘hashKey2’} = $_[2];  #set another hash key
  to the third argument
```

The defined function can be used to determine the existence of arguments

bless function

The reference to the anonymous hash needs to be turned into an object. This is done with the bless function. The bless function takes two arguments, the reference and the class name. If $class is omitted the current class is used

```
bless ($ref, $className);
```

The reference is returned

```
return $ref;
```

Calling the new subroutine

The new subroutine will return a reference to an anonymous hash that stores the data for the object

```
$newObject = PackageName::new(PackageName, any
  arguments);
```

Alternative formats to call the new subroutine

```
$newObject = new PackageName (arguments);
```

Or

```
$newObject = PackageName->new(arguments);
```

Both of these formats pass the package name as the first argument to the new subroutine

Writing the accessors and mutators

Any well written class should provide subroutines to retrieve data from an object. These methods are called accessor methods. A well written class should also provide subroutines to set data in an object. These methods are called mutator methods.

The first argument and required argument to any subroutine will be a reference to the anonymous hash. Create a variable to store the reference

```
$ref = shift @_;
```

Set or access any keys in the hash using the reference to the hash using the referencing notation

```
$ref->{‘key’} = $someValue # sets a value in the
  hash

$someValue = $ref->{‘key’} #retrieves a value from
  the hash
```

Calling the accessors and mutators

To invoke an accessor subroutine

```
$someVal = $newObject->accessorSub();
Or
$someVal = accessorSub ($newObject, args);
Or
$someVal = PackageName::accessorSub ($newObject, args);
```

To invoke a mutator subroutine

```
$newObject->mutatorSub (args);
Or
mutatorSub ($newObject, args);
Or
PackageName::mutatorSub ($newObject, args);
```

Tying a scalar

Perl allows the program to associate a variable with a class using the built-in tie function. The association continues until the program exits or until the untie function is called.

```
tie $scalarVariable, className (, optional arguments)
```

This calls the subroutine TIESCALAR in the ClassName package.

Once a variable is tied every retrieval or modification of its value results in a subroutine being called.

The class that implements the code for the tied variable needs two more subroutines, FETCH and STORE

- Every time the variable has it’s value set the STORE subroutine is called
- Every time the variable has it’s value accessed the FETCH subroutine is called

tie and TIESCALAR

- tie function
  - Receives at least two arguments
    - The first argument is the scalar variable
    - The second argument is the class to which the variable is being tied
    - Any remaining arguments can be used to initialize the variable
  - When the tie function receives a scalar as its first argument it calls the TIESCALAR subroutine and passes the remaining arguments that were originally passed to tie
- TIESCALAR
  - Must be named TIESCALAR within the class/package
  - Initializes the variable (object) for use
  - The TIESCALAR subroutine receives the class from the first argument, creates an object to store the information, blesses a reference to the object with the data and returns a reference to the object

FETCH and STORE

FETCH

- Must be named FETCH within the class/package
- Retrieves the value for the object
- The FETCH subroutine receives a reference to the object as the first argument. The subroutine needs to return a value.

STORE

- Must be named STORE within the class/package
- Sets the value for the object
- The STORE subroutine receives a reference to the object as the first argument. The second argument is the value to be assigned to the object.