Perl Advanced

Объектно-ориентированное программирование в Perl

Пакеты

- Пакет — часть программы
- Пакеты предназначены для разделения глобального пространства имен.
- Пакет начинается с заголовка:
  package ИмяПакета;
- По умолчанию всё выполняется в пакете main.
- Доступ к именам, описанным в другом пакете:
  $ИмяПакета::ИмяПеременной

Пакеты

- Можно несколько раз объявлять один и тот же пакет:

```
package Class1;
$x = 1;
print $Class1::x, $x;                    #11

package Class2;
$x = 2;
print $x, $Class1::x;                    #21

package Class1;
print $x, $Class2::x, $Class1::x;        #121
```

Модули

- Модули представляют собой отдельные файлы, содержащие часть программы.
- Принятое расширение для главного модуля — pl, для загружаемого - pm.
- Подключаются с помощью директив use и require.

```
use Module;     # Загружает модуль Module.pm
```

- use загружает модуль на этапе компиляции, а require на этапе исполнения.

Модули

- Загрузка модуля из подкаталога:

```
# загружает модуль Dir1/Dir2/Module.pm
use Dir1::Dir2::Module;
```

Объекты и классы

- Класс — то же самое, что и пакет.
- Объект — ссылка на любой встроенный тип данных, приведённая к классу с помощью функции bless.

Функция bless

- Функция bless($h, $class) приводит ссылку $h к классу $class. Если второй аргумент отсутствует, то ссылка приводится к текущему пакету.
- Получить имя класса по ссылке можно при помощи функции ref.

```
bless($h, 'Class');      #приведёт ссылку $h к типу Class
$inst = bless($h, 'Class1');
bless($mass, ref($inst));     #приведёт $mass к Class1
```

Методы

- Методы — это все функции пакета.
- Конструктор и деструктор — обычные методы. Их может и не быть. Называться конструктор может как угодно, хотя принято его называть по имени класса или new. Деструктор должен называться DESTROY.

Конструктор

- Метод, возвращающий ссылку на объект.
- Типичный конструктор:

```
sub new{
    my $class = shift;          #ссылка на класс
    my $self = {};              #анонимный хэш
    bless($self, $class);       #приводим к классу
    return $self;
    }
```

Конструктор

- Вызовы конструктора:
  $object1 = new Class;
  $object2 = Class->new();
- Небольшое ухищрение — немного изменим конструктор:
  $invocant = shift;
  $class = ref($invocant) || $invocant;
  $self = {};
  bless($self, $class);
  return $self;

Конструктор

- Строчка $class = ref($invocant) || $invocant; позволяет вызвать конструктор не как статический метод класса, а как метод существующего объекта. Тогда можно создавать объект таким образом:
  my $object3 = $object1->new();

Методы

- Конструктор — метод, ничем по сути не отличающийся от остальных.
- Любые методы могут быть вызваны как статические (используя имя класса) и как методы объекта.

Методы

- Существуют две синтаксические формы записи вызова:

```
# классическая синтаксическая форма - взята из C++
$object->method($arg1, $arg2, $arg3); # метод объекта
Class1->method($arg1, $arg2, $arg3); # метод класса

# вторая синтаксическая форма
method $object $arg1, $arg2, $arg3; # метод объекта
method Class1 $arg1, $arg2, $arg3; # метод класса
```

Методы

- Для большей однозначности можно вызывать, используя ::
  method Class:: $arg1, $arg2, $arg3;
  Class::->method($arg1, $arg2, $arg3);
- Не путать с вызовом функции!
  Class::method($arg1, $arg2, $arg3);

Деструктор

- Perl для каждого субъекта ссылки поддерживает счётчик ссылок и освобождает память, когда он становится равным нулю.
- Если нужно совершить какие-либо дополнительные действия, можно написать собственный деструктор. Он должен называться DESTROY.

Деструктор

- Простой пример:

```
sub DESTROY{
    print «Good Bue!\n»;
    }
```

Наследование

- Каждый класс (пакет) в perl имеет переменную @ISA, в которой перечислены некоторые пакеты. Эти пакеты — базовые классы для текущего класса. Добавить класс в @ISA можно строкой в начале класса-наследника:
  push @ISA, qw(ИмяКласса);

Наследование

- Когда мы вызываем в наследнике метод базового класса, perl просматривает пакеты, перечисленные в @ISA, в поисках метода с таким именем и вызывает его.
- Так как @ISA — массив, мы можем реализовать множественное наследование.

Наследование

```
package Base;
sub new{...}

package Derived;
@ISA = qw(Base);

package main;
$object = Derived->new(); #при создании
```

объекта класса Derived вызовется конструктор класса Base, которого в Derived нет

Наследование: замечания

- При удалении объекта автоматически вызывается только деструктор текущего класса. Деструкторы базовых классов нужно вызывать вручную.
- В поисках метода базового класса perl просматривает пакеты в @ISA в том порядке, в котором они там встречаются и вызывает первый метод с подходящим именем. Если такого нет, в @ISA ищется метод AUTOLOAD.

Класс UNIVERSAL

- В @ISA всегда неявно добавляется класс UNIVERSAL.
- Он содержит 3 метода:
  - can

    ```
    $sub = $obj->can('print');
    ```
  - isa

    ```
    $has_class = $obj->isa("Class");
    ```
  - VERSION

    ```
    $his_vers = $obj->VERSION();
    Some_Module->VERSION(3.0);
    ```

Безымянные подпрограммы

- В предыдущих реализациях класса основой объекта служила ссылка на структуру данных (обычно — хэш), которую мы позже приводили к классу с помощью bless. В таком варианте все данные объекта остаются открытыми.
- Как сделать их закрытыми?

Безымянные подпрограммы

- Нужно из конструктора возвращать ссылку на безымянную подпрограмму, которая имеет доступ к данным объекта через локальную переменную.

Безымянные подпрограммы

```
sub new {
  my $invocant = shift;
  my $class = ref($invocant) || $invocant;
  my $self = { FIELD1 => undef, FIELD2 => undef};
  my $closure = sub {
    my $field = shift;
    if (@_) { $self->{$field} = shift }
    return $self->{$field};
  };
  bless($closure, $class);
}
```

Безымянные подпрограммы

- Getter и Setter в одном методе:

```
sub FIELD1  {
    &{$_[0]}("FIELD1", @_[1..$#_])
    }

sub FIELD2 {
    &{$_[0]}("FIELD2", @_[1..$#_])
    }
```

Связанные переменные

- Очень интересной возможностью языка PERL является связывание переменных с объектами. Суть этого явления состоит в том, чтобы скрыть реализацию объекта за переменной. Мы можем читать и изменять значение связанной переменной обычным образом, но при этом неявно будут вызываться соответствующие методы объекта, которые могут быть сколь угодно сложными.

Связанные переменные

- Для того, чтобы использовать связывание, мы должны оформить класс специальным образом. Это оформление зависит от типа связываемой переменной.
- Конструктор должен называться TIESCALAR для связываемого скаляра.
- Для чтения переменной определяем функцию FETCH. Передаётся один параметр — связанная переменная.
- Для записи — функцию STORE. Параметры — связанная переменная и новое значение.

Связанные переменные

```
sub TIESCALAR {
    my $class = shift;
    print "конструктор\n";
    bless $value, $class;
}
sub FETCH {
    my $self = shift;
    print "чтение\n";
    return $self->value;
}
```

```
sub STORE {
    print "запись\n";
    my $self = shift;
    my $value = shift;
    $self->value = $value;
}
```

Связанные переменные

- Чтобы связать переменную с классом, нужно вызвать функцию tie. В этот момент вызывается конструктор TIESCALAR.

```
my $var;
tie $var, 'Class', $arg;
```

- Получить ссылку на созданный объект можно использовав функцию tied $переменная. Если переменная ни с чем не связана, функция возвращает undef.
- Для разрыва связи используется untied. Вместе с ней неявно вызывается деструктор объекта.

```
untie $var;
```