HYPE 3. Библиотека поддержки парадигмы обьектно-ориентированного программирования в sp-forth 4. Руководство пользователя.
rev 1.2
Дмитрий Якимов, 2006. support@activekitten.com

Обьектно-ориентированного программирование далее ООП.

ООП парадигма включает в себя инкапсуляцию данных, наследование поведения и полиморфизм. Рассмотрим применение этих концепций.

1. Инкапсуляция.

Означает изоляцию данных от пользователя и то что каждый экземпляр содержит свои данные.

Эта концепция реализована с помощью форт словарей. Слово **CLASS** начинает объявление класса, слово **;CLASS** заканчивает. Все что между этими словами компилируется в именованный форт словарь класса.

методы
Методы класса являются самыми обычными форт словами. Вызов метода класса из другого метода этого же класса не отличается от вызова форт слова, т.е. обычный CALL.

данные
Для определения данных экземпляра служат слова **DEFS** (n "name" --), **PROPERTY** (n "name" --), **OBJ** (class "name").

Слово DEFS создает переменную размера n байт с именем "name". При обращении к переменной в методе экземпляре класса на стек кладется адрес начала переменной. N может быть любым 32 разрядым числом.

Слово PROPERTY дополнительно к переменной компилирует слова доступа name@ и name!, и производит выравнивание переменной так, чтобы на целевой платформе не возникло ошибки невыровненных данных (ARM, MIPS). N может быть равно 1, 2, 4, 8. Это слово сослужило хорошую службу на pocket pc.

Слово OBJ компилирует объектную переменную, которая будет автоматически инициализирована экземпляром обьекта при создании данного экземпляра-агрегата (владельца). Все обьектные переменные IMMEDIATE форт слова, при вызове они выбирают имя метода из входного потока и компилируют или выполняют его.

Например:

```
CLASS CTest
   CAnyClass OBJ obj

: testMethod obj name TYPE ;

;CLASS
```

2. Наследование

Наследование классов осуществляется словом **SUBCLASS** (class "name").

Как уже упоминалось - класс по сути дела является форт словарем, таким образом чтобы обеспечить решение задачи ООП наследования достаточно связать форт словари.

При компиляции метода обьектной или локальной переменной или через синтаксис статической компиляции **::** - имя ищется в словаре текущего класса и классов предков.

Если же необходимо вызвать метод базового класса из метода класса наследника - используется синтаксис **SUPER nameOfMethod**

Например:

```
CLASS CParent

: test ." test" ;

;CLASS

CParent SUBCLASS CChild

: calltest SUPER test ;

;CLASS

\ динамический вызов
CChild ^ calltest
```

3. Полиморфизм

Это важнейшее свойство ООП, оно означает возможность динамически в момент выполнение программы найти и выполнить метод экземпляра.

В hype реализуется словом ^ (obj "name"). Например конструкция CClass ^ method найдет и выполнит метод method класса CClass. Или выдаст сообщение об ошибке если такого метода у класса нет.

Это позволяет создавать абстрактные интерфейсы, за которыми стоит совершенно разный код и использовать эти интерфейсы в одном контексте.

4. Вызов методов.

Метод экземпляра или класса можно вызвать статически либо динамически. Статически означает обычный call, переход по адресу. При динамическом вызове метод ищется в форт словаре во время выполнения программы.

Обьектные переменные, локальные переменные, статические экземпляры знают свои классы, поэтому методы в этих случаях вызываются статически.

А) Статический вызов метода не текущего класса. Применяется когда класс известен на момент компиляции и экземпляр у нас будет находиться на стеке в момент выполнения.
Синтаксис:

(obj) :: Class.method

Б) Динамический вызов метода класса или экземпляра:

(classOrObject) ^ method

Чтобы избежать конфликта с локальными переменными spf4 используется также синоним - слово =>

В) Вызов метода объектной или локальной переменной или метода статического экземпляра:

VariableName method

5. Классы тоже экземпляры

Классам можно посылать сообщения – динамически через ^ синтаксис.
Но – класс не имеет данных, поэтому обращение к SELF смысла имеет мало. Класс MetaClass имеет методы: получения имени **name** (-- addr u), печати всех методов класса и классов родителей **methods.** (--), получения размера экземпляра класса **size** (-- n), получения адреса экземпляра на стеке **this** (-- obj)

Так как все объекты неявно наследуются от MetaClass, то эти сообщения можно посылать и любым экземплярам.

6. Создание/удаление обьектов

Объекты можно создавать в

- В хипе
- В пространстве данных форта
- На стеке возвратов
- Где захочет пользователь

Библиотека предоставляет слова создания/уничтожения объектов в хипе: NewObj (class – addr) и FreeObj (obj).

В методе init автоматически создаются переменные класса – экземпляры других классов, определенные через слово OBJ, они будут созданы в том же пространстве что и класс владелец.

Пример кода создания объектов в хипе:

```
CTest NewObj VALUE ourObject

ourObject ^ name TYPE

ourObject FreeObj
```

Для создания статических объектов в пространстве данных форта используется слово
NEW (class “name”).

Все вложенные переменные-объекты также создаются в этом случае в пространстве данных форта. Имя этого статического объекта имеет IMMEDIATE значение (подобно переменным класса – объектам) – при вызове **name method** будет или скомпилирован код метода или метод будет выполнен.

Пример:

```
CTest NEW testObj

testObj name TYPE

testObj dispose
```

Слово FreeObj не подходит для статического объекта, поэтому для освобождение экземпляром всех ресурсов (клавиатура, экран, сокет) мы просто вызываем метод dispose.

Есть возможность создавать экземпляры указав XT слова выделения памяти под объект. Реализуется эта возможность словом **HYPE::NewObjWith** (class xt). Все вложенные экземпляры будут созданы с помощью этого же XT. Аналогично уничтожение словом **HYPE::FreeObjWith** (obj xt)

7. Инициализация/деинициализация

После выделения памяти NewObj или NEW вызовут метод **init** текущего класса и всех классов родителей по очереди, начиная с самого дальнего класса родителя и заканчивая init текущего класса.

При удалении последовательность обратная – сначала вызывается dispose текущего класса, затем dispose класса родителя, затем класса-деда и т.п.

В методе init автоматически создаются переменные класса – экземпляры других классов, определенные через слово OBJ, в dispose они удалятся (после того как отработает код dispose).

Чтобы обеспечить автоматический вызов конструкторов/деструкторов классов-родителей, а также чтобы автоматически уничтожить объектные переменные класса – используйте для объявления конструкторов и деструкторов синтаксис:

init: your_code_here **;**

dispose: your_code_here **;**

8. Локальные обьектные переменные

Часто необходимо иметь объект в пределах одного форт слова, весьма удобно при этом иметь автоматическое создание/уничтожение. Эту задачу решает библиотека локальных объектных переменных.

Синтаксис: || класс имя класс имя ||

где класс это имя класса, имя это имя переменной. При входе в форт слово экземпляры этих классов автоматически будут созданы на стеке возвратов. Внимание – вложенные переменные этих классов будут созданы в хипе.

Были применены оптимизации:

1. Расположение переменной на стеке возвратов обеспечивает очень быстрое выделение и удаление памяти экземпляра
2. Если класс не содержит dispose (вернее всего один dispose в ProtoObj классе) , и не содержит вложенных переменных то dispose вызываться не будет. Это весьма удобно для небольших классов которые часто используются, типа CRect.
3. В SPF4 взятие объекта со стека по смещению осуществляется одной командой процессора

Локальные объектные переменные в одном слове нельзя использовать с локальными переменными SPF4 (синтаксис { }). Поэтому были определены локальные переменные-значения:

R: - локальная переменная, берет свое значение со стека данных

D: локальная переменная, инициализируется нулем.

S: локальная переменная строка, используются ~ac\lib\str.5

Эти переменные VARIABLE типа, то есть оставляются адрес значения на стеке данных.

R: D: и S: на самом деле классы, в R: например в методе init происходит инициализация значения переменной класса величиной со стека данных.

А слова доступа @ и ! это методы.

Пример:

```
: test ( addr )
  || CTest obj R: addr D: addr2 ||
  addr @ addr2 !
  addr2 @ .
;
```

Таким же образом можно сделать F: - локальную переменную float число.

9. Обработка ошибок

Возможна ситуация, когда посланное в runtime сообщение методу с помощью оператора ^ не найдет адресата – экземпляр не содержит метода с таким названием. В этом случае будет вызван метод unknown (addr u), который определен в предке всех классов MetaClass. По умолчанию этот метод выводит сообщение о ненайденном методе. Если перегрузить метод unknown, возможно будет динамически формировать методы класса.

Также MetaClass содержит метод abort (f addr u), который сможет, если f != 0, вывести сообщение об ошибке и распечатать стек вызовов, включая имена классов, что серьезно упрощает локализацию ошибки.

10. FAQ

1. **Как получить указатель на объект в методах самого объекта?**

   Для этого есть USER-VALUE переменная SELF, или метод this.

2. **Как получить указатель на объект локальной переменной или объектной переменной класса?**

   Метод this.

3. **Как вызвать метод класса-родителя, имя которого (метода) формируется автоматически?**

   В WFL, например, имя метода – обработчика Windows сообщения формируется автоматически из числа, например W: WM_PAINT, имя метода в данном случае будет W0000001F, слово SUPER здесь помочь никак не может, используйте синтаксис безымянного наследования метода: **INHERIT** – это слово скомпилирует вызов метода класса-родителя одноименный текущему компилируемому:

   ```
   W: WM_PAINT
    INHERIT
   ;
   ```

4. **Как сократить количество вызовов SUPER?**

   Используйте синтаксис **with{ class_name }}**
   Пример:

   ```
   : test
      with{ MetaClass name TYPE size . }}
   ;
   ```