Чернышев Георгий

# Объекты (версия 19.11.2011)

# ООП

- Результат эволюционного развития идей процедурного программирования
- Попытка программировать с помощью высокоуровневых абстракций
- Первый язык – Симула, 1967 год

# Три кита ООП

- Инкапсуляция
  - Объединение записей с процедурами и функциями работающими с этими записями. Это – объект.
- Наследование
  - Определение свойств объекта и использование их в дальнейшем для построения иерархии порожденных объектов
- Полиморфизм
  - Присваивание определенному действию имени, которое потом используется по всей иерархии объектов сверху до низу, причем каждый объект выполняет это действие характерным ему способом

# Пример: Строка в языке Си

- Конкатенация

```
char* str1 = new char[10] ;
str1 = “abc\0”;
char* str2 = new char[10] ;
str2 = “def\0”;
char* str3 = new char[strlen(str1) + strlen(str2)];
strncpy(str1, str3, strlen(str1));
strncpy(str2, str3+strlen(str1), strlen(str2))
str3[strlen(str1) + strlen(str2)] = ‘\0’;
```

- И такой код писать каждый раз при конкатенации!
  - Что будет с существующем проектом если механизм работы с этой строкой поменяется?

# Реализация наследования в Паскале

- RECORD не имеет возможности наследования

```
TYPE
   TPerson = OBJECT
        Name: STRING[30];
        Date:STRING[10];
        Rate:REAL;
END;

   TStudent =  OBJECT(TPerson)
        Score:REAL;
END;
```

# Экземпляры объектов

- Так же как и обычные статические или динамические переменные

```
TYPE
   PStudent = ^TStudent;
VAR
   Static_student: TStudent;
   Dynamic_student: PStudent;
```

# Поля объектов

```
Student.Score := 4.5;


WITH Student DO BEGIN
   Name := 'Иванов Иван Иванович';
   Date := '19-11-2010';
END;
```

# Инициализация и проба методов

```
TYPE
    TPerson = OBJECT
        Name: STRING[30];
        Date:STRING[10];
        Rate:REAL;
    PROCEDURE Init (Nm, Dt: STRING, Rt:Real);
END;

PROCEDURE TPerson.Init(Nm, Dt:String; Rt:REAL)
BEGIN
    Name := Nm;
    Date := Dt;
    Rate := Rt;
END;
```

# Определение методов

- Функции и процедуры
- Поля данных объявляются перед методами

```
TYPE
   TPerson = OBJECT
        Name: STRING[30];
        Date:STRING[10];
        Rate:REAL;
   PROCEDURE Init (Nm, Dt: STRING, Rt:Real);
   FUNCTION GetName : STRING;
   FUNCTION GetDate : STRING;
   FUNCTION GetRate : REAL;
END;
```

# Определение методов

```
PROCEDURE TPerson.Init(Nm, Dt:String; Rt:REAL)
BEGIN
   Name := Nm;
   Date := Dt;
   Rate := Rt;
END;

FUNCTION TPerson.GetName: String;
BEGIN
   GetName := Name;
END;
```

# Некоторые конструкции

- WITH <объект> DO …
  - Возможность использование полей, вызова метода без прописывания идентификатора объекта
- Self – неявная ссылка на себя
  - GetName := Name;
  - GetName := Self.Name;

# Скрытие методов объекта

- Модификаторы доступа:
  - PUBLIC – видны всем и всегда
  - PRIVATE – ограничены радиусом модуля
  - Действуют и на данные и методы
- Модули:
  - Подключаются с помощью USES

# Инкапсуляция

- Фактически была описана выше
  - Объединение в объекте кода и данных
- Ключевые аспекты:
  - Нет необходимости обращаться к полям непосредственно

# Зачем закрывать поля?

- Объекты будут использоваться другими программистами
  - Защита от ошибок
    - Ваша задача – предоставить набор объектов совокупность которых решает задачу. Самое важное – сделать так, чтобы этим набором правильно пользовались.
  - Защита от недобросовестного использования
  - Ясный контракт помогает быстрее разобраться

# Пример: объект строка

```
TYPE
    TString = OBJECT
        contents: array of [1..255] char;
        length: integer;
        PROCEDURE PRINT;
        PROCEDURE SETSTR (s: array of char);
END;

TString.PRINT;
VAR
        i: integer;
BEGIN
        i := 0;
        while (i < length) do writeln(contents[i]);
END;
```

- Что будет если пользователь вдруг изменит только длину строки?

Вывод: длина строки должна быть закрыта от пользователя модификатором private

# Переопределение методов

```
TYPE
    TPerson = OBJECT
        Name: STRING[30];
        Date:STRING[10];
        Rate:REAL;
    PROCEDURE Init (Nm, Dt: STRING, Rt:Real);
END;

TYPE
    TStudent = OBJECT(TPerson)
        Score: REAL;
        FUNCTION GetScore:REAL;
        PROCEDURE Init (Nm, Dt: STRING, Rt, Sc:Real);
        PROCEDURE ShowScore;
END;

PROCEDURE TStudent.Init(Nm, Dt:String; Rt, Sc:REAL)
BEGIN
    TPerson.Init(Nm, Dt, Rt);
    Score := Sc;
END;
```

Вызов наследуемого метода <Предок>.<Метод>.

# Наследование статических методов

- Все что было написано ранее – касалось наследования статических методов
- Если методы A и B заданы в объектах типа TPerson и TStudent, то при вызове из метода A, метода B в объекте TStudent будет вызван метода B объекта TPerson
  - Иначе говоря: будет вызываться метод родителя

# Решение: виртуальные методы

- Это суть идеи полиморфизма
- Раннее и позднее связывание

# Совместимость типов

- Потомок совместим со своими родителями
- Проявляется:
  - Между экземплярами объектов
  - Между указателями на экземпляры объектов
  - Между формальными и фактическими параметрами
- Реализована с помощью приведения типов

# Приведение типов

- X = (ТИП) Y
- Теперь тип объекта X будет Y
- Сталкивались и без объектов (неявное приведение типов, первый семестр):
  - char -> integer -> real
  - Но не наоборот (нужно явное)

# Виртуальные методы

- Задаются ключевым словом VIRTUAL
  - Задаются как в родительском типе
  - Должны быть так же объявлены в потомках

```
FUNCTION GetSum:REAL; VIRTUAL;
PROCEDURE ShowSum; VIRTUAL;
```

# Пример

```
TPerson.Print()
Begin
   writeln('человек');
End;
TStudent.Print()
Begin
   writeln('студент');
End;
// виртуальные
```

```
Var
   a: array [1..3] of
   TPerson;
   i : integer;
Begin
   a[1] := TPerson.Init(..);
   a[2] := TStudent.Init(..);
   a[3] := TPerson.Init(..);
   for i := 1 to 3 do
   a[i]^.Print();
End.
```

Выведет: человек студент человек

# Конструкторы и Деструкторы

- Каждый тип объекта имеющий виртуальные методы обязан иметь к-р
- Конструктор – специальный тип процедуры выполняющий инициализацию
- Вызов любого виртуального метода происходит только после вызова конструктора

```
CONSTRUCTOR Init(Nm, Dt: STRING, Rt, Sc:
   REAL);
DESTRUCTOR Done; VIRTUAL;
```

# Динамические объекты

- Все что было раньше – статические объекты, на стеке
- Можно и нужно держать в динамической памяти. Для этого использовать указатели и NEW:
  - VAR P:^TPerson;
  - NEW(P, <конструктор>);
- P^.Init(…) – в начале нужен вызов конструктора, всегда

# Деструкторы

- Удаление такого объекта из динамической памяти
- Dispose(P) может быть мало:
  - Сложная структура которая держит память
  - Бинарное дерево, реализуем операцию удалить все элементы большие X

# Это совсем не конец

- Особенности устройства таблицы виртуальных методов
- Динамические методы
- TypeOf, SizeOf
- …

# Что почитать

- Википедия про ООП, все ссылки (много)
- Турбо Паскаль 7.0, BHV
  - Любая другая хорошая книжка по нему
- Встроенный Help Turbo Pascal, учебника это, однако, не заменит

# Задачи

- Для зачета за эту тему надо набрать 3 балла
  - Задачи без * считаются за 1 балл
  - Задачи со * и ** считаются соответственно за 2 и 3 балла
- Некоторые задачи имеют дополнения в условиях, при которых их ценность повышается. Для зачета этого типа задач необходимо предъявить оба варианта (и простой вариант тоже)

# О задачах

- Эти задания – на объекты, а не на алгоритмическую реализацию. Тут важно:
  - Понимание основ ООП (наследование, полиморфизм)
  - Сложные задачи показывают как применять паттерны проектирования
- Задачи очень творческие, имеют много решений

# Задача – "Комплексные числа"

- Реализовать комплексные числа как объект
- Должен поддерживать методы:
  - CompareTo(Complex)
  - Add
  - Multiply
  - Print

# Задача – “Стек”

- Реализовать стек как объект
- Должен поддерживать методы:
  - Push
  - Pop
  - Clone
  - CompareTo(Stack)

# Задача – “Очередь”

- Реализовать очередь с приоритетами как объект
- Должен поддерживать методы:
  - Enqueue
  - Dequeue
  - Clone
  - CompareTo(Queue)

# Задача – “Список”

- Реализовать объект “двусвязный список”
- Должен поддерживать методы
  - Поиск
  - Конкатенация
  - toArray
  - Поворот

# Задача – “Компьютерная сеть”

- Есть компьютеры, N типов операционных систем, Li уровень защиты от вируса
- Компьютеры соединены с другими компьютерами сетью
- Вирус живет на N типе операционной системы, имеет уровень атаки K.
  - Если компьютеры соединены сетью и уровень вируса выше уровня защиты то компьютер заражается
  - После заражения вирус может включать или выключать компьютеры по желанию
- Вычислить какие компьютеры вирус, стартуя из данного узла может выключить (выключаете их вы, заражение происходит строго до выключения и до конца)
- Вывести все “пограничные” компьютеры

# Задача – “Матрица”

- Реализовать библиотеку по работе с матрицами
  - Умножение на матрицу, на вектор
  - Печать
  - След и вычисление решения методом гаусса уравнения Ax=B, строки ЛНЗ

# Задача – “Множество”

- Реализовать структуру данных “множество”
- Обязательно должны быть операции
  - Пересечение
  - Объединение
  - Разность
  - Проверка принадлежности элемента множеству
- Если сделать так, чтобы элементом данного множества мог быть любой тип какой захотим, без переписывания всего кода, то тогда *
  - В данном случае нужен компаратор

# Задача – “Кинотеатр”

- Реализовать сервис по продаже билетов
  - Кинотеатр – поле N*M
  - Есть L зон, непересекающихся, со своей ценой
  - Зона – последовательный набор рядов
- Запросы:
  - Купить X мест, в линию или прямоугольник, бронь идет на группу с названием C
  - Снять бронь взятую группой с  названием C
  - Инициализировать зал
  - Подсчитать прибыль
  - Людям чем ближе тем лучше, но важно сесть вместе

# Задача – “Поезд”

- Есть поезд с набором вагонов C, у вагона есть грузоподъемность и объем
- Груз – типы A,B,C..Z
- Поезд проходит станции, на каждой из которых могут сделать следующее
  - Загрузить груз (попытаться, не влез - остался)
  - Выгрузить груз, с головного вагона N тонн
  - Разворовать, с хвостового вагона – объем тот же, вес уменьшается вдвое. Разворовывают определенный объем
  - Угоняют целый вагон (с конца)
  - Действия известны на каждой станции
- Посчитать с чем приедем

# Задача – “Просто строки”

- Реализовать объект “Строка”
- Методы:
  - Char charAt(int x)/setCharAt(int x, char c)
  - concat(string x)
  - Substr(int start, int end)
  - Print (string x)
  - Split(char), выдает массив строк на основе деления по символу: “abc*cc*ff*”->abc, cc, ff
  - toUpper()/toLower()
  - toUpper(int)/… - для конкретного слова N

# Задача* – “Кватернионы”

- Реализовать кватернионы на комплексных числах
- Операции:
  - Сложение/умножение
  - Сопряжение/модуль/обращение
  - Печать, ввод итд

- http://ru.wikipedia.org/wiki/%D0%9A%D0%B2%D0%B0%D1%82%D0%B5%D1%80%D0%BD%D0%B8%D0%BE%D0%BD

# Задача* – “Магазин”

- Реализовать симулятор магазина со следующей функциональностью
  - Обработка нескольких заказов
  - Несколько заказов от одного клиента получают скидку
- Заказ – набор продуктов, с ценами, от клиента. Операции
  - Подсчитать стоимость всего
  - Добавить/удалить товар
- Подсчитать выручку магазина за день

# Задача* – “Префиксное дерево”

- Реализовать префиксное дерево

- Операции
  - Добавление
  - Удаление
  - Поиск

# Задача* – “Scheduler”

- Реализовать “параллельное” исполнение нескольких работ. Работа - это исполнение в цикле каких-либо инструкций
- Планировщик последовательно  в случайном порядке исполняет работы (threads)
- Реализовать такую систему
  - Пригодятся паттерны стратегия и диспетчер

# Задача* – “Транзакции”

- Транзакция - это последовательность операций
  - Read(Xi), Write(Xi), Commit, Abort
  - Commit - успешное завершение, Abort – отмена всех операций
- Каждой транзакции идет в паре уровень изоляции – никакой и полный (замок налагается на все переменные в транзакции)
- Выполнить данный набор транзакций с их уровнями. Исполнитель транзакций должен в случайном порядке исполнить данный набор транзакций

# Задача* – “список++”

- Тоже что и простой список, но должен уметь работать с любым элементом

- Нужно создать класс элемент и от него отнаследовать различные типы

# Задача* – “Список с итератором”

- Реализовать тот же набор операций, что в задаче “Список”
  - Добавить объект “итератор”, зависящий от списка, и позволяющий: вставлять за настоящим элементом, удалять настоящий элемент
- Итератор:
  - http://en.wikipedia.org/wiki/Iterator

# Задача* – “Строки”

- Реализовать объект “Строка”
- Методы:
  - Char charAt(int x)/setCharAt(int x, char c)
  - concat(string x)
  - Substr(int start, int end)
  - Clone(string x)
  - New (string x)
  - Print (string x)

- Ленивость строки повышает до **
  - http://en.wikipedia.org/wiki/Lazy_evaluation
  - То есть, должен быть string pool, строки раскопируются тогда когда это нужно

# Задача* – “Электронная почта”

- Написать сервис электронной почты который позволяет
  - Читать и отправлять “сообщения”
  - Сервис локальный на одной машине (данные хранятся в файле)
  - Все это сделать с объектами
- Дополнительные баллы за
  - Сортировку, классификацию по тэгам
  - Типы пользователей: базовый, платный и администратор

# Задача* – “Файловый менеджер”

- Создать симулятор файлового менеджера с интерфейсом командной строки
  - Реализации работы с диском не требуется
  - Создание, удаление, перемещение файла/каталога
  - Поиск в файле, поиск по имени файла
  - Ссылки на файлы – как  в линуксе
- ** - сделать поиск по имени файла с */?

# Задача* – “Множество++”

- Реализовать объект – “множество”, должно быть произвольного типа
- Операции:
  - Стандартные: пересечение, объединение, разность
  - Дополнительные:
    - Отбросить все элементы > X, < X
    - Вырезка X<?<Y, результат – тоже множество
    - Вырезка <X, >X , результат – тоже множество
  - Реализовать его на основе дерева

# Задача* – “Органайзер”

- Создать электронный органайзер который
  - Имеет встроенные часы (в тестовых целях переводятся вручную)
  - Имеет два типа задач
    - Повторяющиеся (например еженедельные)
    - Разовые, установленные на определенное время
  - По этим задачам, обладает возможностью в любое время вычислить список задач на сегодня
  - Задачи могут добавляться и удаляться вручную

# Задача* – “Касса”

- Создать приложение обеспечивающее работу кассы железнодорожных билетов
  - Вагоны бывают трех различных типов
  - Каждый тип вагона имеет свое расположение мест
  - Стоимость зависит от типа вагона и места
- Должны быть реализованы
  - Бронирование на группу (с учетом пожеланий) по местам в вагонах
  - Снятие бронирования на группу
  - Подсчет выручки

# Задача** – “Бронирование”

- Создать систему продажи билетов для авиаперевозок с учетом пересадок
  - Есть 3 типа мест – эконом класс, первый класс, бизнес-класс, цена зависит от типа места
  - Рейс
    - Вылетает из A в B, во время X, прилетает во время Y
    - Считать что пересадки требуют времени не более 15 минут
  - Ваша задача предложить пользователю варианты того как долететь до места назначения, вывести список с ценами. По требованию пользователя упорядочивать по цене, по общему времени полета, по количеству пересадок, по времени ожидания в аэропортах

# Задача** – “Менеджер памяти”

- Для хранения структур данных вида запись будем использовать структуру хранения, которая состоит в виде единого куска памяти и указателей на нее

- Функциональность:
  - New
  - Delete
  - Clone
  - Подсчитать размер утекшей памяти в конце работы
- Для простоты можно считать что работаем с одной “страницей”, записей большего размера чем страница нет

# Задача** – “Список с умным итератором”

- Тоже самое что в задаче с итератором, однако, теперь, к каждому списку может подцепиться несколько итераторов. То есть, может случиться беда, если один из итераторов удалит элемент на который указывает другой элемент
- Сделать так, чтобы не было проблемы
  - Подсказка – нужен менеджер и замки. Хотя решений тут масса

# Задача** – “Рогалик”

- Создать Roguelike игру
  - http://en.wikipedia.org/wiki/Roguelike
- В ней должны быть объекты:
  - Внутри-игровые: игрок, монстры, предметы, абстракция команды
  - Объекты, связанные с визуализацией
- Задание довольно обширное, и поэтому творческое. Если что-то не будет реализовано, то ничего страшного. Однако, если задание будет слишком упрощено, будут сделаны некоторые Feature Requests для зачета

# Задача** – “Цивилизация”

- Создать игру для N игроков в режиме hot-seat
  - Поле M*N, на котором отряды и города
  - Города производят отряды
  - Отряды передвигаются по карте и могут захватывать города
- Все остальное по вашему усмотрению
- Задание довольно обширное, и поэтому творческое. Если что-то не будет реализовано, то ничего страшного. Однако, если задание будет слишком упрощено будут сделаны некоторые Feature Requests для зачета

# Задача** – “Оконный менеджер”

- Создать оконный менеджер, который умеет
  - Работать с несколькими окнами, накладываемыми друг на друга
  - Окно, имеющее фокус, рисуется поверх остальных (концепция Z-уровней)
  - Окна можно перемещать по экрану, сворачивать
- Окно имеет
  - Цвет рамки
  - Цвет фона

# Задача** – “Диалоговый менеджер”

- Создать диалоговый менеджер, который умеет
  - Работать с несколькими диалогами
  - Окно, имеющее фокус, рисуется поверх остальных (концепция Z-уровней)
- Диалог имеет
  - Цвет фона
  - Текст, не выходящий за рамки окна
  - Кнопки, по которым вызывается другие диалоговые окна

# Задача** – “SVG”

- Создать редактор векторной графики, который умеет
  - Работать с несколькими различными примитивами (линия, окружность, прямоугольник и.т.д.)
  - Запоминать их в виде объектов, именовать композицию
- Редактор позволяет
  - Изменять масштаб изображения
  - Удалять части из композиции, передвигать объекты и их композиции
  - Сохранять в файле

# Задача** – “mini СУБД”

- Реализовать СУБД:
  - Набор таблиц со значениями (INT, STR)
  - Реляционная алгебра:
    - SELECT
    - PROJECT
    - JOIN по “=”
- Объекты:
  - Таблица, запрос
  - Промежуточный результат
- Задача не самая простая, поэтому упрощения
  - Оптимизация не нужна
  - Запросы в удобной для вас форме
  - Условия на SELECT “= Value”, без предикатов
  - http://en.wikipedia.org/wiki/Sql

# Задача** – “Диспетчер”

- Реализуем оконный простенький менеджер, как из предыдущего задания, но теперь с другим фокусом
- Идея: обрабатывать клики мышкой, если возможно drag-n-drop наших окошек
- Главное: диспетчер
  - Хранит очередь событий
  - Хранит объекты
  - Маршрутизирует объекты событиям, нотифицируя нужных
- Проблемы:
  - Научиться работать с мышью
  - Хотелось бы что-то типа такого:
    - http://en.wikipedia.org/wiki/Mediator_pattern

# Задача** – “Компьютер”

- Есть узлы компьютера, с различными разъемами (A, B, C, …)
  - Каждый компонент определенного типа потребляет N вольт, всю оставшуюся передают узлам по порядку подключенным
  - Вставляется только  в разъем определенного типа
- Поддерживать данную структуру
  - Добавлять узлы (говорить нельзя или нет)
  - Удалять узлы
  - Подсчитать используемую мощность

# Ссылки

- http://en.wikipedia.org/wiki/Design_pattern_%28computer_science%29
  - По-хорошему, сильному программисту отсюда надо знать все
- Книги
  - Gamma, “Gang of four”, классика от звезд, очень глубокая, написана тяжелейшим языком, зато полно примеров по делу и везде есть
  - Портянкин И. *Swing*. Эффектные пользовательские интерфейсы – как оно работает, на примере работающей JAVA SWING и устройство GUI систем
  - Какие-то книги по concurrency паттернам
    - Java concurrency in practice, обложка с поездами, не читал но профессионалы все хвалят