**АКАДЕМИЯ УПРАВЛЕНИЯ «ТИСБИ»**

**А.Н. Козин**

***Объектно-ориентированное программирование***

***УЧЕБНО-МЕТОДИ******ЧЕСКОЕ пособие***

Практические задания

КАЗАНЬ – 2006

**1.5. Практические задания**

Задание 1. Создание и использование отдельных классов для графических примитивов.

**Постановка задачи.** Требуется создать небольшой набор не связанных друг с другом классов, описывающих основные графические примитивы (окружность, отрезок, прямоугольник и т.д.). Каждый класс должен содержать необходимые параметры-свойства (координаты базовой точки, размеры, признак видимости), не менее двух конструкторов, метод отображения примитива **Show,** метод для перемещения объекта **MoveTo.** Кроме того, классы могут содержать уникальные методы, например – изменение радиуса окружности, изменение линейных размеров прямоугольника, поворот отрезка и т.д. Каждый класс оформляется в виде отдельного модуля, который подключается к основной программе для демонстрации возможностей.

**Порядок выполнения работы** (для инструментального пакета Delphi)**:**

1. Создать новый проект и сохранить его в собственном каталоге.
2. Разместить на главной форме основное меню для последующей реализации команд демонстрации возможностей объектов-примитивов. На начальном этапе достаточно в меню поместить две команды – **Создать** и **Переместить,** подменю которых конкретизируют используемые объекты.
3. Разместить на форме компонент **Image** для отображения примитивов и выровнять его на всю клиентскую область.
4. Создать обработчик события формы **OnCreate** для инициализации датчика псевдослучайных чисел и установки режима рисования на поверхности компонента **Image**:

**Image1.Canvas.Pen.Mode := pmNotXOR;**

этот режим позволяет стирать существующее изображение за счет повторного вывода на то же самое место с автоматическим восстановлением фона.

1. Добавить в проект новый программный модуль, **НЕ СВЯЗАННЫЙ** со своей формой: команда меню **File / New / Unit** и **обязательно сохранить его вместе с проектом.** Поскольку модуль является чисто программным, то его заготовка содержит лишь заголовок, начало интерфейсной секции (директива **interface**) и начало секции реализации (директива **implementation**).
2. В интерфейсном разделе нового модуля ввести описание **(ТОЛЬКО ОПИСАНИЕ!)** одного из создаваемых классов (в разделе описания типов). Для простоты сначала можно взять класс окружностей, подробно описанный в лекционном материале. Остальные классы можно добавить после проверки работы класса окружностей. В классе сделать поля данных закрытыми, а методы – открытыми. Каждый примитив должен иметь базовую точку, изменение координат которой приводит к перемещению примитива без изменения его размеров.
3. Оформить раздел реализации модуля, включив в него программную реализацию каждого объявленного метода. Например:

**procedure TCircle.MoveTo(dx, dy : integer);**

**begin**

**Show; //** повторный вывод для стирания изображения примитива

**x := x + dx; y := y + dy; //** изменение координат базовой точки

**Show; //** рисование в новом месте

**end;**

В методе **MoveTo** для удобства дальнейшего использования задаются не новые координаты базовой точки, а **СМЕЩЕНИЕ** относительно нее, которое может быть и отрицательным.

1. Программная реализация методов показа примитивов может использовать стандартные графические примитивы системы **Windows.** Например, вывод окружности выполняется следующим образом:

**Form1.Image1.Canvas.Ellipse(x-r, y-r, x+r, y+r);**

Поскольку этот код использует компонент **Image** основной формы, необходимо подключить модуль главной формы к модулю с классами, записав сразу после директивы **interface** строку

**uses** ИмяГлавногоМодуля;

1. После полного описания всех методов сохранить модуль и подключить его к основному модулю.
2. В разделе реализации основного модуля объявить одну-две переменные объектного типа, а также – массив объектных указателей для проверки возможностей групповой обработки объектов-примитивов.
3. Написать обработчики команд создания примитивов, выполняющие следующие действия:

* если соответствующий объект-примитив уже был ранее создан (указатель на него не равен **nil**), то стереть старое изображение повторным вызовом метода **Show** и уничтожить старый объект методом **Free;**
* создать новый объект вызовом конструктора со случайными параметрами примитива;
* показать примитив методом **Show.**

1. Написать обработчики команд перемещения примитивов в новую точку со случайными координатами, используя генерацию значений смещения координат в пределах [-50, 50].
2. Написать обработчики команд изменения геометрических свойств объектов (радиус окружности, размеры прямоугольника, положение конечной точки отрезка и т.д.).
3. Написать обработчики команд создания массивов объектов и группового изменения их свойств.
4. Сохранить проект и проверить его работу.

**Задание 2. Самостоятельно придумать и описать на разных языках 2-3 класса для объектов разной природы.**

**2.4. Практические задания**

Задание 1. Реализация взаимодействия классов на основе агрегации

**Постановка задачи:**

Разработанные в первом задании отдельные классы для графических примитивов необходимо модифицировать с целью реализации возможности их агрегационного взаимодействия. Для этого требуется ввести класс точек и заменить в классах для графических примитивов непосредственное использование свойств-координат объектами-точками.

**Порядок выполнения работы;**

1. Открыть ранее созданный проект и сохранить его в новом каталоге для внесения необходимых изменений.
2. Добавить новый класс для примитивов-точек, содержащий:

* два свойства для хранения координат точки;
* конструктор для инициализации свойств-координат и вывода сообщения о создании объекта (это будет использоваться в дальнейшем при проверке правильности вложенных вызовов конструкторов взаимодействующих классов);
* Set/Get-методы для доступа к координатам точки.

1. Внести следующие изменения в ранее созданный класс Окружность:

* заменить свойства-координаты одним свойством-точкой;
* реализовать два конструктора, которые, кроме стандартных операций, выполняют дополнительно:
  + - проверку значения радиуса и при необходимости - изменение его так, чтобы окружность не выходила за пределы области рисования;
    - вывод сообщения о создании объекта-окружности;
  + в методе прорисовки окружности вместо прямого использования координат центра применить вызовы соответствующих методов класса точек;
  + в методе перемещения окружности заменить прямую установку новых значений координат вызовом соответствующего метода класса точек.

1. Проверить работу приложения, обратив внимание на порядок создания взаимодействующих объектов.
2. Создать новый класс Кольцо, содержащий два свойства-окружности, два конструктора с выводом сообщения, а также методы прорисовки и перемещения.
3. Проверить работоспособность всех методов созданных классов.
4. Внести аналогичные изменения в классы Отрезок и Прямоугольник.
5. Самостоятельно придумать и реализовать 1-2 класса для манипуляций со сложными объектами, включающими более простые объекты Точка, Отрезок, Прямоугольник, Окружность; проверить работоспособность этих классов.

Задание 2. Создание и использование библиотеки классов для графических примитивов на основе принципа наследования.

**Постановка задачи:**

Требуется создать небольшую иерархию классов, описывающих основные графические примитивы (окружность, треугольник, отрезок, прямоугольник и т.д.). Корнем всей иерархии является класс **TFigure,** определяющий общие свойства и поведение всех объектов-примитивов: координаты базовой точки примитива, конструктор, методы доступа, **абстрактные** методы прорисовки **Show** и перемещения **MoveTo.** В каждом классе необходимо реализовать свой конструктор и методы прорисовки и перемещения. Кроме того, классы могут содержать методы, уникальные только для соответствующего поддерева, например – изменение радиуса окружности, изменение линейных размеров прямоугольника, поворот треугольника и т.д.

Вся библиотека оформляется в виде отдельного модуля, который подключается к основной программе для демонстрации возможностей этой библиотеки.

**Порядок выполнения работы:**

1. Открыть один из ранее созданных проектов и сохранить его в собственном каталоге.
2. В интерфейсный раздел модуля-библиотеки ввести описание класса **TFigure**, используя для задания абстрактных методов директиву **abstract** в заголовке метода.
3. Привести программную реализацию всех неабстрактных методов класса **ТFigure** (в конструкторе предусмотреть вывод сообщения об инициализации свойств-координат).
4. Создать класс окружностей **TCircle** как потомка класса **TFigure** с добавлением нового свойства – радиуса окружности, собственным конструктором, методом **Show,** методом **MoveTo** и новым методом изменения радиуса.
5. Дать программную реализацию конструктора дочернего класса окружностей, ОБЯЗАТЕЛЬНО начинающуюся с вызова конструктора родителя:
   * **inherited Create**(параметры);
   * в конце предусмотреть вывод сообщения о создании объекта
6. Дать программную реализацию остальных методов класса окружностей
7. В основном (демонстрационном) модуле предусмотреть создание объектов-окружностей, их перемещение и изменение радиуса.
8. Проверить работоспособность созданных классов.
9. На базе класса **TCircle** создать дочерний подкласс эллипсов добавлением свойства-полуоси. Реализовать свой конструктор и методы прорисовки и перемещения. Для поворота на 90 градусов ввести новый метод.
10. Проверить работоспособность созданных классов.
11. Аналогично создать класс четырехугольников и подклассы для ромбов, трапеций, прямоугольников и проверить их работоспособность.

**3.4. Практические задания**

Задание 1. Создание библиотеки классов для графических примитивов с использованием виртуальных методов.

**Постановка задачи:**

Требуется изменить созданную ранее иерархию классов для основных графических примитивов с целью использования виртуальных методов. Для этого в иерархию классов необходимо внести некоторые изменения.

**Порядок выполнения работы:**

1. Открыть ранее созданный проект и сохранить его в собственном каталоге.
2. Внести в базовый класс фигур следующие изменения:
   * абстрактный метод прорисовки объявить виртуальным;
   * метод перемещения сделать реальным, с вызовом виртуального метода прорисовки.
3. Во все дочерние классы внести следующие изменения:

* объявить метод прорисовки переопределяемым, оставив без изменения его программную реализацию;
* убрать из класса метод перемещения, который теперь должен наследоваться и динамически настраиваться.

1. Проверить работоспособность созданных классов.

Задание 2. Использование полиморфных объектных указателей.

**Постановка задачи:**

На основе созданной в предыдущем задании библиотеки графических фигур необходимо реализовать приложение для демонстрации основных возможностей полиморфных указателей. Для этого требуется создать простейший контейнер с возможностью занесения в него **любых** объектов-фигур. Простейший контейнер (иногда называемый коллекцией) можно создать на основе массива полиморфных указателей. Должны быть предусмотрены следующие операции:

* заполнение массива указателями на случайно выбранные фигуры;
* показ всех фигур в массиве;
* стирание всех фигур в массиве;
* перемещение сразу всех фигур в новую базовую точку;
* выборочная обработка только фигур из некоторого поддерева (например, перемещение только окружностей и эллипсов) за счет динамического определения типа объектного указателя при прохождении по массиву;
* уничтожение массива фигур с освобождением памяти.

**Порядок выполнения работы:**

1. Открыть ранее созданный проект с библиотекой графических фигур и сохранить его в новом каталоге.

2. При необходимости внести в базовую библиотеку дополнения: библиотека должна содержать не менее трех уровней (например, ветку наследования классов **TFigure – TCircle – TEllipse**) и 5-6 классов.

3. В разделе реализации основного модуля объявить массив из 20-30 полиморфных указателей, т.е. элементов базового типа **TFigure.**

4. В основное меню ввести команду **Массив** с подкомандами **Создать, Уничтожить, Стереть, Показать** и **Переместить.**

5. Написать обработчик команды **Создать,** который должен в цикле создать случайный набор из 20-30 фигур со случайными параметрами и сохранить указатель на каждый из них как элемент массива, например:

**mas[ i ] := TCircle.Create(параметры);**

для случайного выбора типа фигуры можно каждому типу присвоить порядковый номер, генерировать этот номер случайно и использовать номер как селектор в инструкции выбора

6. Написать обработчик команды **Показать** для прорисовки всего изображения за счет прохода по массиву с вызовом для каждого элемента массива виртуального метода **Show.**

7. Написать обработчик команды **Стереть** только для стирания изображения всех фигур без уничтожения объектов, что дает возможность повторно нарисовать изображение.

8. Написать обработчик команды **Уничтожить** для стирания изображения всех фигур и уничтожения соответствующих объектов; для этого в цикле для каждого указателя вызывается метод **Show** и затем метод **Free.**

9. Написать код, который перемещает все изображение влево-вправо-вверх-вниз по нажатию соответствующей клавиши. Для простоты шаг перемещения можно взять равным 10 пикселам. Для реализации необходимо:

* + установить свойство формы **KeyPreview=true** для перехвата формой клавиатурного ввода;
  + создать обработчик события формы **OnKeyDown;**
  + в этом обработчике проверить по **Case-Of** значение параметра **Key,** используя стандартные виртуальные коды клавиш **vk\_Left, vk\_Right, vk\_Up, vk\_Down;**
  + для каждого случая в цикле вызвать общий метод перемещения **MoveTo** с соответствующими значениями смещений.

10. Сохранить проект и проверить его работу.

11. Добавить в меню команду для выборочного перемещения только объектов-фигур некоторой подиерархии. Обработчик этой команды должен как обычно пройти по массиву объектных указателей, но при этом выполнить проверку динамического типа каждого указателя на принадлежность к заданной подиерархии с помощью операции **is.**

12. Проверить работу приложения.

13. Добавить возможность вызова при проходе по массиву уникального метода для заданной подиерархии, например – изменения радиуса окружности (он же горизонтальная полуось эллипса). Для этого кроме проверки динамического типа указателя, надо выполнить его приведение к базовому подтипу, например – типу **TCircle**; для приведения используется операция **as**, после применения которой появляется возможность вызова с помощью полиморфного указателя методов подкласса (например, методов класса **TCircle).**

14. Проверить работу приложения.

Задание 3. Создание и использование контейнерных классов.

## Постановка задачи:

Требуется создать класс-контейнер для хранения любых графических примитивов из ранее созданной библиотеки. Контейнер должен быть реализован в двух вариантах – на базе динамического массива и на базе линейного динамического списка. Каждый контейнерный класс реализуется в отдельном модуле. Демонстрационная программа должна показывать использование созданных контейнеров.

## Порядок выполнения работы:

1. Добавить в ранее созданный проект еще один программный модуль, предназначенный для реализации контейнерного класса на базе массива или линейного списка.
2. В разделе интерфейса этого модуля:

* подключить модуль с графической библиотекой;
* описать либо базовый класс контейнера-массива, либо два класса – класс элементов списка и класс самого контейнера-списка.

1. В разделе реализации записать код всех методов описанных классов. Особое место среди методов контейнера занимает метод-итератор, который организует проход по массиву или списку и с каждым объектом-примитивом выполняет заданное действие. В простейшем случае итератор можно оформить как метод, принимающий входной параметр-признак выполняемого действия. Внутри метода с помощью инструкции **Case** можно выбрать требуемое действие и внутри цикла применить его к каждому объекту. Можно предусмотреть выполнение следующих действий: перемещение примитива на заданное смещение, показ/стирание примитива, уничтожение примитива со стиранием.
2. В разделе реализации главного модуля подключить контейнерный модуль, ввести переменную-указатель на объект-контейнер и переменную-указатель на любой примитив (типа **TFigure).**
3. Добавить в основное меню команду **Контейнер** с элементами **Создать, Добавить, Переместить, Уничтожить.**
4. Написать обработчик команды **Создать,** в котором с помощью конструктора создается пустой контейнер и выводится информационное сообщение.
5. Написать обработчик команды **Добавить,** в котором создать случайный примитив со случайными параметрами, добавить его в контейнер и показать методом **Show.**
6. Написать обработчик команды **Переместить,** который лишь вызывает итератор с соответствующими параметрами.
7. Написать обработчик команды **Уничтожить,** который с помощью итератора уничтожает все объекты-примитивы, затем уничтожает методом **Free** сам контейнер и выводит об этом сообщение.
8. Сохранить проект и проверить его работу.

**7. Рекомендуемая литература**

1. Иванова Г.С. Объектно-ориентированное программирование. – М.: МГТУ им. Баумана, 2004.
2. Кьоу Дж., Джеанини М. Объектно-ориентированное программирование. Учебный курс. – СПб.: Питер, 2005.
3. Вайсфельд М. Объектно-ориентированный подход: Java, .NET, C++. – М.:КУДИЦ-ОБРАЗ, 2005
4. Бадд Т. Объектно-ориентированное программирование в действии. – СПб.: Питер, 1997.
5. Буч Г. Объектно-ориентированное проектирование с примерами применения. – М.: Конкорд, 1992.
6. Грэхем И. Объектно-ориентированные методы. Принципы и практика. – М.: Вильямс, 2004.
7. Элиенс А. Принципы объектно-ориентированной разработки программ. – М.: Вильямс, 2002.
8. Бобровский С. Технологии Delphi 2006. – СПб.: Питер, 2006.
9. Пачеко К. Delphi for .NET. Руководство разработчика. – М.: Вильямс, 2005
10. Павловская Т.А. С++. Объектно-ориентированное программирование. - СПб.: Питер, 2004.
11. Павловская Т.А., Щупак Ю.А. С++. Объектно-ориентированное программирование: Практикум. – СПб.: Питер, 2004.
12. Топп У., Форд У. Структуры данных в С++. – М.: БИНОМ, 2000.
13. Эккель Б. Философия Java. – СПб.: Питер, 2001.
14. Ноутон П., Шилдт Г. Java 2. – СПб.: БХВ-Петербург, 2001.
15. Байдачный С.С. .NET Framework 2.0. Секреты создания Windows-приложений. – М.: СОЛОН-Пресс, 2006
16. Темплман Д., Виттер Д..NET Framework: Библиотека классов. – М.: КУДИЦ-ОБРАЗ, 2003.
17. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб.: Питер, 2001.
18. Буч Г., Рамбо Д., Джекобсон А. Язык UML. Руководство пользователя. – М.: ДМК, 2000.