Министерство общего и профессионального образования

Российской Федерации

## Казанский государственный технический университет

имени. А.Н. Туполева

----------------------------------------------------------------------------------

Кафедра АСОИУ

В.И. Медведев

# Методические указания к курсовой работе

# по дисциплине

# **«Объектно-ориентированное программирование»**

по направлению 230200 – Информационные системы

факультет ТКиИ

курс 2, сем. 3

### Казань 2008

**Медведев В.И.**

# M 42 Методические указания к курсовой работе по дисциплине

# «Объектно-ориентированное программирование» – Казань:

Казан. гос. гехн. ун-т, 2009. –107 c.: ил

Излагаются основные требования к выполнению курсовых работ по дисциплине объектно-ориентированного программирования. Приводится краткое изложение ключевых понятий дисциплины и описание нотации языка UML, достаточной для изображения диаграмм классов, иллюстрирующих этапы разработки разрабатываемой объектно-ориентированной программы.

Излагаются цели курсовой работы и требования к оформлению пояснительной записки.

Приводится задания для курсовых работ и пример оформления пояснительной записки курсовой работы.

Используется среда Visual Studio .NET с применением библиотеки классов .NET Framework и пакетов Java.

Для студентов вузов по направлению 230200 – информационные системы.

УДК 681.3.06

ББК 32.973.26 - 018

© Медведев В. И., 2009

**ВВЕДЕНИЕ К КУРСОВОЙ РАБОТЕ**

**Основные термины, используемые в дисциплине**

**“Объектно-ориентированное программирование”**

Напомним об основных понятиях объектно-ориентированного программирования.

***Объект (object)*** – это ограниченная сущность, характеризующаяся своим состоянием и поведением. Объект является *экземпляром класса*, который описывает всё множество объектов данного типа, то есть объектов с таким же состоянием и поведением. Объектно-ориентированная программа реализует объектно-ориентированную модель, состоящую из множества взаимодействующих между собой объектов, каждый из которых создан из соответствующего класса.

Тип ***class*** (класс) описывает множество однотипных объектов с одними и теми же данными и функциями, то есть обладающих *общим состоянием и поведением*. Посредством *class* программист описывает совместно данные и действия (функции) над этими данными. Из класса с помощью оператора new создаются *объекты* (переменные типа *class*). При этом данные, принадлежащие объекту некоторого класса, обусловливают ***состояние (state)*** этого объекта, а набор функций – его ***поведение (behavior)***.

Используя метки *private*, *public* и *protected* можно по-разному инкапсулировать (скрывать) данные и функции в классе, запрещая или разрешая доступ к ним извне объекта.

Метка ***private (закрытый)*** объявляет данные и функции, описанные с нею, закрытыми. Эти данные и функции могут использоваться только функциями данного класса.

Метка ***public (открытый)*** объявляет данные и функции класса, описанные с нею, доступными (открытыми) вне объекта внутри блока, в котором объявлен или создан объект этого класса. Эти данные и функции могут использоваться применительно к объекту извне объекта, а также функциями данного класса.

Метка ***protected (защищённый)*** объявляет данные и функции класса, описанные с нею, защищенными. Эти данные не доступны вне объекта и передаются при наследовании из класса в порождённый класс.

***Конструктор (constructor)*** – это функция класса, основное назначение которой состоит в инициализации значений данных вновь создаваемого объекта. Имя конструктора всегда совпадает с именем класса. Конструктор вызывается всякий раз при создании объекта с помощью оператора *new*. Класс может иметь несколько перегруженных конструкторов. При создании объекта применяется тот конструктор, параметры которого обеспечивают установку требуемых начальных значений данных этого объекта. Конструктор не имеет возвращаемого значения.

Открытые функции класса образуют ***интерфейс (interface) объектов****,* создаваемых из этого класса. В дальнейшем объекты взаимодействуют только через их интерфейсы, поскольку только интерфейсные функции доступны вне объекта. Используя ключевое слово ***interface*** можно отдельно описать интерфейс, включающий требуемые функции, свойства и события. В дальнейшем этот интерфейс можно использовать при разработке классов, наследуя их из одного интерфейса или нескольких интерфейсов.

***Порождение (наследование- inheritance) классов*** – это средство получения новых классов на базе существующих. При этом новый класс наследует данные и функции из одного ***базового (base)*** класса и, возможно из нескольких интерфейсов. В C# и Java запрещено множественное наследование классов. Наследуемые интерфейсы обязывают порождённый класс реализовать все функции, свойства и события, объявленные в этих интерфейсах

**В языке C#** объявление порождённого класса начинается с ключевого слова *class,* за которым указывается имя порождённого класса. Затем после двоеточия следует *список порождения*, состоящий из имени одного базового класса и отделённых запятыми имен интерфейсов. После этого *заголовка* следует *тело* порождённого класса, содержащее объявления данных и функций.

*сlass* имя-порожд-класса **:** **[{**базовый класс **[**,имя-интерф**] . . .]}**

**|{** имя-интерф **[**,имя-интерф**] . . .}]**

{

объявление данных и описание функций, свойств и событий

};

**В языке Java** объявление порождённого класса начинается с ключевого слова *class,* за которым указывается имя порождённого класса. Затем после ключевого слова ***extends*** можно указать имя одного базового класса, а после последующего слова ***implements*** отделённые запятыми имена интерфейсов. После этого *заголовка* следует *тело* порождённого класса, содержащее объявления данных и функций.

*class* имя-порожд-класса

**[{extends** базовый класс **[**, **implements** имя-интерф**] . . .]}**

**|{ implements** имя-интерф **[**,имя-интерф**] . . .}]**

{

объявление данных и функций

}

**В языке C#** ***свойство (property)*,** описанное в классе, позволяет просто использовать связанную с ним *закрытую обычную* или *ссылочную переменную*, устанавливая или получая её значение. Свойство приписывает этой переменной *имя свойства*. В дальнейшем ссылка к переменной осуществляется через имя свойства. Создав объект некоторого класса, можно использовать его свойства для получения или изменения соответствующих ему данных. При изменении значения свойства присваемые ему значения можно контролировать и модифицировать при реализации свойства. Возможно только чтение, только запись или как чтение, так и запись свойства. Свойства можно использовать в выражениях.

Для включения свойства в класс в классе объявляется закрытая переменная свойства или ссылка, связанная со свойством. Затем описывается свойство:

тип имя-свойства

{

*set*

{

код реализации *set*, использующий параметр *value* и *переменную*

или *ссылку свойства*

}

*get*

{

код реализации g*et*, возвращающий значение *переменной* или

*ссылки свойства*

}

}

**В языке Java *свойство*** описывается посредством функций, имя которых начинается с *set* или *get* с последующим именем свойства. Функция *set* устанавливает значение переменной свойства, а функция *get* получает (возвращает) значениие соответствующей переменной свойства.

***Делегат (delegate)* языка C#** – это объект, предназначенный для хранения указателя функции, подлежащей выполнению. Являясь необычным объектом, делегат может объединять в себе с помощью операторов “+” и “-“ несколько других делегатов, превратившись при этом в так называемый *множественный делегат*. Таким образом, делегат способен инкапсулировать указатели многих функций. Выполнение делегата приводит к выполнению всех инкапсулированных в нём функций. Объект делегата можно передать в качестве аргумента в какую-либо функцию и выполнить в её теле. Также объект делегата можно передать и в качестве аргумента события.

Делегат описывается так:

*delegate* тип-возвр-значения тип-делегата (список-парам);

А объект делегата создаётся следующим образом:

тип-делегата ссылка-на-делегат = *new* тип-делегата

(ссылка-на-объект**.**имя-обычной-функции);

тип-делегата ссылка-на-делегат = *new* тип-делегата

(имя-класса**.**имя-статической-функции);

При выполнении делегата необходимо указать список аргументов, тип и количество которых должно соответствовать списку параметров делегата:

ссылка-на-делегат (список-аргументов);

Выполнение делегата приведёт к выполнению инкапсулированной (-ных) в нём функции (-й) с указанными аргументами.

Используя оператор “+=“ можно к делегату добавить новый объект делегату с его инкапсулированной функцией, превратив тем самым его в множественный делегат. Выполняясь, множественный делегат выполнит последовательно функции всех добавленных делегатов в порядке их подсоединения. С помощью оператора “+=“ можно из множественного делегата удалить объект указанного делегата.

***Событие (event)* в языке C#** – специальный делегат, который привязан к объекту. Чрезвычайно важно знать, что ***в отличие от делегата, событие нельзя выполнять вне объекта !*** Важно также отметить, что как событие связано с определённым объектом, также и обработчики других объектов связаны с этим конкретным событием и соответственно – с его объектом. То есть имеет место жесткая фиксация связи между событием и теми обработчиками, которые должны выполниться при возникновении этого события. Выполнение события вызывает выполнение связанных с ним обработчиков.

Создание и применение события требует выполнения следующих шагов:

- объявить делегат события, определяющий формат функций-обработчиков этого события;

- объявить в *классе-источнике события*, используя ключевое слово ***event***, *ссылку на событие*:

*event*  тип-делегата-события ссылка-на-событие;

- объявить в классе-источнике события функцию, генерирующую это событие, то есть выполняющую объект события, или выполнить это событие в требуемых местах иных функций класса-источника;

- при выполнении события необходимо выполнить проверку на его пустоту, то есть на включение в событие хотя бы одного объекта делегата события с инкапсулированной в нём функции-обработчика события:

*if* ( *ссылка-на-событие* != *null* )

*ссылка-на-событие* (список-аргументов );

- объявить в *классах-приёмниках события* функции-обработчики этого события;

- создать *объект-источник события* и *объекты-приёмники события*;

- связать функции-обработчики каждого приёмника события с событием объекта-источника (подписаться на событие):

ссылка-на-объект-источник**,**ссылка-на-событие += *new*

тип-делегата-события(

ссылка-на-объект-приемник**.** имя-обычной-функции);

ссылка-на-объект-источник**,**ссылка-на-событие += *new*

тип-делегата-события(

ссылка-на-объект-приемник**.** имя-статической-функции);

- выполнить надлежащим образом событие в объекте-источнике.

Придерживаясь рекомендации фирмы Microsoft желательно с каждым событием связать *объект данных события*, создав его из класса, наследуемого класс EventArgs. В делегате события в качестве первого параметра рекомендуется использовать ссылку типа object на объект-источник события, а в качестве второго – ссылку на объект данных события. Функция-обработчик, воспользовавшись аналогичными параметрами, выявит источник события и воспользуется всеми данными, включёнными в объект события.

Библиотека .NET Framework, содержащая более 4000 классов, имеет многие из них с предопределёнными событиями, например, события MouseDown, MouseUp, Click генерируются при нажатии на клавишу мыши. Предопределенным событиям соответствуют предопределённые делегаты событий, например, для мыши - MouseEventHandler.

***Событие (event)* в языке Java** можно представить явным делегированием обработчика из объекта-приемника в объект-источник. При этом удобно воспользоваться интерфейсом, поскольку использование интерфейсной ссылки позволяет ссылаться к разнотипным объектам (приемникам), классы которых наследуют один и тот же интерфейс, связанный с этим событием. Класс источника должен иметь конструктор с параметром - интерфейсной ссылкой. Класс каждого приёмника этого события обязан наследовать этот интерфейс, включающий делегированный обработчик события. Генерирование события источником сводится к вызову в объекте-источнике обработчика приёмника, ссылка на который передаётся в источник через интерфейсную ссылку в конструкторе.

Широко применяются **в языке Java** особые события – ***уведомления***. Класс Observable и интерфейс Observer пакета java.util позволяют воспользоваться механизмом уведомления языка Java. *Наблюдаемые объекты*, являющиеся объектами-источниками, классы которых наследуют класс Observable, уведомляют *обозревателей*, являющимися объектами-приёмниками, классы которых наследуют интерфейс Observer. Класс наблюдателя Observable переопределяет функции setChanged(), сигнализирующую об изменении состояния наблюдателя, и функцию notifyObservers(), уведомляющую обозреватель и передающую ему объект с данными. Интерфейс Observer обязывает обозреватель переопределить функцию update(), реагирующую на уведомление и получающую объект данных.

Под ***потоком* (*thread*)** понимают выполняющийся программный код. Выполнение процесса начинается с выполнения ***основного* *потока* (*primary* *thread*)**, представленного главной функцией *main* или *WinMain*, или *DllMain*. Обычно основной поток реализует диалог с пользователем. Основной поток инициирует, выполняет и завершает работу процесса. Если основной поток создаёт окно, то его называют *основным* *оконным* *потоком*. Обычно с окном связан цикл обработки сообщений операционной системы.

Основной поток может создать и запустить так называемые ***рабочие* *потоки* (*work* *threads*)**, которые будут выполняться параллельно с основным и другими потоками. Если выполнение основного потока сводится к выполнению *главной* *функции* приложения, то выполнение каждого рабочего потока связано с выполнением так называемой ***потоковой* *функции***. **В языке Java** применяется потоковая функция с именем ***run***. **В языке же C#** в качестве потоковой функции можно использовать функции с любыми именами, но каждая из них инкапсулируется в поток посредсвом специального потокового делегата ThreadStart.

Итак, **в языке C#** класс ***Thread*** используется для создания и управления потоками, а тип ***ThreadStart*** - для создания делегата потока, инкапсулирующего потоковую функцию. При создании экземпляра потока конструктору класса *Thread* передаётся в качестве аргумента объект делегата потока, при этом возможны два варианта перегруженного конструктора делегата: 1) указывается статическая потоковая функция некоторого класса или 2) потоковая функция конкретного объекта определённого класса. Функции *Start*, *Suspend*, *Resume* и *Abort* класса *Thread* обеспечивают запуск, приостановку и возобновление функционирования или уничтожение потокового объекта. Следует сказать, что необходима осторожность при применении функций *Suspend* и *Resume* , поскольку они способствуют дедлоку – тупиковой ситуации, вызывающей взаимную остановку выполнения потоков из-за не освобождения некоторого разделяемого ресурса. В этом случае выполняющаяся программа “зависает”.

**В языке же Java** для создания и управления потоками используется класс ***Thread*** и интерфейс **Runnable.** Интерфейс **Runnable** позволяет передавать функцию run() в конструктор ***Thread*** при создании потоковых объектов

***Windows-приложение* (*a Windows application*)** – программа, выполняющаяся под управлением операционной системы *Windows*.

***Приложение* (*an application*)** – программа для выполнения определенного вида работ, например: редактирования текста, выдачи текущего времени и так далее. Каждое приложение является процессом и обычно представлено на экране своим прикладным окном.

***Прикладное*** или ***главное окно* (*an application*** или ***main window*)** – прямоугольная область на экране, выделяемая одному приложению для ввода и вывода данных и управления приложением. Операционная система *Windows* допускает параллельное одновременное выполнение нескольких приложений с возможной синхронизацией их работы, при этом каждое из них может быть представлено своим прикладным окном. Обычно наприкладном окне размещаются интерфейсные элементы.

***Интерфейсный элемент*** – это то, что представляет объект на экране, обеспечивая интерактивную связь пользователя с приложением. Обычно элемент представляется в виде специализированных дочерних окон – кнопок, панелей редактирования, ползунков и других.

***Графический интерфейс (graphic interface)*** использует для рисования такие объекты как цвет (*color*), перья (*pens*), кисти (*brushes*), изображения (*images*) и шрифты (*fonts*).

Прежде чем рисовать фигуру или линию, надо создать ***графический объект*** класса *Graphics*. Графический объект связан с конкретным окном (формой) и его функции выполняют рисование в этом окне (форме), включая при необходимости в качестве аргументов перья, кисти, шрифты и т.д. Таким образом, вспомогательные объекты типа *Color*, *Pen*, *SolidBrush*, *Image* и *Font* обеспечивают функционирование объектов класса *Graphics,* занимающих центральное место в создании графического интерфейса. Объекты вспомогательных типов хранят необходимую для рисования информацию, а графический объект её использует.

Наиболее часто употребляемый тип ***Point (точка)***, используется для создания объекта – одиночной точки, обычно в области клиента окна.

Также часто употребляется тип ***Rectangle (прямоугольник)*,** используемый для создания объекта-прямоугольника с горизонтальными и вертикальными сторонами.

Объект типа ***Color (цвет)*** представляет цвет.

Класс ***Pen (перо)*** используется для создания объектов-перьев (*pens*),

Класс ***SolidBrush (сплошная кисть)*,** порождённый из базового класса *Brush*, используется для создания объектов-кистей, применяемых при закрашивании фона окон или внутри фигур.

Объектами класса ***Font*** представляются шрифты с указанными названиями, размерами и стилями.

Объект класса ***Graphics*** является контекстом рисования, связанным с конкретным контекстом устройства и чаще всего с окном (формой). Класс *Graphics* предоставляет множество функций для рисования в этом окне строк (*strings*), прямоугольников (*rectangles*), эллипсов (*ellipses*) и т.д.

При всяком разрушении окна *Windows* посылает приложению событие ***Paint (нарисовать)*** и приложение реагирует на нем предопределённой функцией-обработчиком ***OnPaint***, пустой по умолчанию, то есть ничего не отображающей в окно. Обладая переопределённой (*override*) функцией *OnPaint*, приложение будет рисовать в области клиента окна что-то в соответствии с её реализацией.

Управляющие элементы составляютоснову при формировании пользовательского интерфейса приложения. Элементы управления, такие как нажимаемые или селективные кнопки, панели списков, панели редактирования и другие, широко используются разработчиками *Windows*–приложений. Они значительно облегчают интерфейс между пользователем и *Windows* -приложением.

Из множества классов управляющих элементов наиболее часто применяются два класса: *Button* и *TextBox*.

Из класса ***Button(кнопка)*** создаётся так называемая *нажимаемая кнопка*.

Из класса ***TextBox (текстовый редактор)*** **языка C#** и класса ***TextField*** **языка Java** создаётся *панель редактирования*, которая представляет собой простейший редактор с возможностью ввода и редактирования текста.

**Объекты, классы и язык UML**

Приступая к созданию какой-либо программной системы, разработчикам желательно воспользоваться языком более высокого уровня, чем языки программирования, чтобы обсудить структуру как всей системы в целом, так и её частей и взаимосвязь этих частей в пространстве (статически) и во времени (динамически) безотносительно к языку программирования. Желательно, чтобы этот язык оперировал такими понятиями объектно-ориентированных систем как класс, объект, устанавливал бы связи между ними статически и динамически.

При обсуждении разрабатываемой системы в нотациях этого языка хотелось бы уточнить данные и функции каждого класса, при необходимости разделить сложные классы на более простые или объединить некоторые простенькие классы, проанализировать динамику создания и уничтожения объектов во времени, их взаимосвязь, и так далее. И только после активного обсуждения проекта системы выбрать языки программирования для реализации её частей и приступить к созданию программного кода, придерживаясь принятых совместных решений.

Для графического представления объектно-ориентированных систем в настоящее время используется язык *UML*.

*Унифицированный язык моделирования* ***UML*** (*Unified Modeling Language*) предназначен для описания объектно-ориентированных систем в виде совокупности диаграмм, раскрывая статическую и динамическую суть системы как модели, состоящей из взаимодействующих объектов.

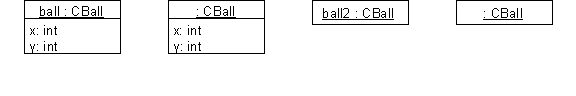
Перед применением языка *UML* необходимо проанализировать проектируемую систему, выявить основные сущности и понятия, образовать набор классов, из которых будет создаваться всё множество объектов системы. Затем приступают к разработке диаграмм языка *UML*: *диаграмм вариантов использования*, представляющих систему с точки зрения пользователя, *диаграмм классов*, представляющих классы и их статическую взаимосвязь, *диаграмм последовательности*, изображающих динамическое функционирование объектов во времени, и др.

На рис. 2.1 изображен класс на языке *UML*. Класс представляется в виде прямоугольника, разделённого на три части, в которых размещается имя класса, перечень атрибутов и, наконец, перечень функций класса. Перед именами данных и функций ставятся символы “– “ и “+”, указывающие на доступ *private* и *public* соответственно.



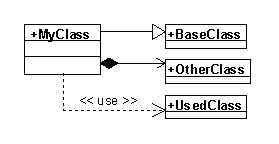
**Рис. 2.1.** Класс CBall

Из рис. 2.1 видно, что на языке *UML* представлен класс, имеющий имя *CBall*, закрытые данные *х* и *у* и открытые функции *Set*, *Move* и *Show*. Важно, что это представление класса не зависит от языков программирования и, в частности, от *С++*, *С++.NET*,  *Java*,  *J#*  и *С#*.



**Рис. 2.2.** Объекты класса CBall

На рис. 2.2 изображены объекты класса *CBall* на языке *UML*. В прямоугольнике помещается имя объекта, отделённое от класса этого объекта двоеточием. Если объект безымянный, то ставится двоеточие перед именем класса. Имя объекта и класса подчёркиваются.



**Рис. 2.3.** Класс MyClass наследует класс BaseClass, содержит объект класса OtherClass и использует класс UsedClass

Нотация на рисунке 2.3 показывает наследование одного класса из другого класса, обладание объектом другого класса и использование в одном классе другого класса.

Язык *UML* довольно-таки богат, и при необходимости можно познакомиться с ним и освоить его по многочисленной изданной литературе.

О многогранности описания разрабатываемой системы на этом языке можно судить по перечню описывающих её диаграмм языка UML: диаграмма взаимодействия (interaction diagram), диаграмма деятельности (activity diagram), диаграмма использования (use case diagram), диаграмма классов (class diagram), диаграмма компонентов (component diagram), диаграмма кооперации (collaboration diagram), диаграмма объектов (object diagram), диаграмма последовательности (sequence diagram), диаграмма развёртывания (deployment diagram), диаграмма состояний (statechart diagram).

В задачи книги не входит подробное изложение языка *UML*. Познакомиться с нотацией этого языка можно по книге Рамбо Дж., Якобсона А. и Буча Г. “UML: специальный справочник”.

Диаграммы класса языка *UML* должны использоваться при описании поэтапной разработки программ.

1. **ЦЕЛИ И ЗАДАЧИ КУРСОВОЙ РАБОТЫ**

**Целью курсовой работы** является приобретение практических навыков поэтапной разработки нераспределённых объектно-ориентированных программ, разработки собственных классов и применения классов библиотеки .NET Framework и пакетов языка Java, создания из них объектов и обеспечении требуемого совместного функционирования этих объектов как объектно-ориентированной модели, реализующей поставленную задачу, а также описания разработанного программного продукта как с точки зрения его пользователя, так и с точки зрения разработчика.

В результате выполнения курсовой работы студенты должны:

1. изучить и применить принципы поэтапной разработки и отладки программ средней сложности.
2. уметь разрабатывать собственные классы, создавать из них разнообразные объекты и применять их совместно для реализации требуемых алгоритмов функционирования программы.
3. уметь применять необходимые классы библиотеки NET Framework, создавать из них требуемые объекты и употреблять их в программе.
4. уметь применять необходимые классы пакетов Java, создавать из них требуемые объекты и употреблять их в программе.
5. уметь разрабатывать собственный интерфейс программы, применив необходимые интерфейсные элементы на базе классов библиотеки NET Framework и пакетов Java.
6. уметь создавать необходимые события, делегаты и потоки и применять их для эффективного параллельного функционирования объектов и их взаимодействия.
7. уметь описывать разработанный программный продукт как с точки зрения его пользователя, отражая его внешнюю сторону, так и с точки зрения программиста, отражая особенности его реализации.
8. **ТРЕБОВАНИЯ ПО ВЫПОЛНЕНИЮ КУРСОВОЙ РАБОТЫ**

При выполнении курсовой работы разработчик прежде всего должен проанализировать предметную область, связанную с разрабатываемой программной системой, выделить основные объекты и разработать для них соответствующие классы или воспользоваться готовыми классами библиотеки NET Framework.

Необходимо выделить особые объекты – источники событий и определить объекты – приёмники этих событий, разработать необходимые делегаты событий, обеспечить связь источников с их приёмниками.

Тщательно необходимо продумать вопросы, связанные с применением потоков и использования ими событий.

После анализа совокупности предполагаемых объектов и их взаимодействия и разработки соответствующих им классов надлежит перейти к поэтапной разработке создаваемой программной системы. Перед переходом к созданию программы на компьютере желательно на бумаге прикинуть поэтапную разработку программы, при которой программа из основного ядра будет обрастать объектами добавляемых классов, приближая своё функционирование к требуемому по заданию.

Желательно на каждом этапе добавлять по одному классу и каждый этап тщательно отлаживать. Ввиду отлаженных предыдущих этапов ошибки на данном этапе будут вызваны некорректностью добавляемых классов или неправильным применением их объектов.

Очень полезно для просмотра деталей функционирования программы любого этапа и облегчения поиска ошибок в Windows приложении кроме прикладного окна приложения иметь и консольное окно, выдавая на него изменяющиеся состояния объектов. По мере отладки целесообразно закомментировать в программе некоторые операторы выдачи на консоль, дабы сократить объём выводимой информации.

Результаты разработки каждого этапа и возникшие при этом проблемы должны быть отражены в пояснительной записке по курсовой работе.

Тематика курсовых работ должна быть связана с разработкой программ, реализующих модель, состоящую из разнотипных объектов с потоками, взаимодействующих между собой с помощью событий. Программный интерфейс должен включать интерфейсные элементы типа кнопок, текстовых редакторов, списков и т.д., объекты которых создаются из соответствующих классов библиотеки NET Framework.

Задание на курсовую работу описывает поведение некоторого множества объектов, подлежащих реализации в виде программной системы.

Это задание должно быть уточнено разработчиком путём уточнения поведения объектов, предложением новых событий, потоков и новых взаимосвязей между объектами. Разработчик должен предложить программный интерфейс, позволяющий не только управлять функционированием системы объектов, но и отображать информацию об их функционировании дополнительных окнах или текстовых редакторах.

Уже разработанные на этапах программы необходимо сохранять, чтобы облегчить поиск допущенных на предшествующих этапах логических ошибок, которые проявились при отладке текущего этапа. Каждый этап разрабатываемой программы должен тщательно тестироваться.

Разработанная программа должна быть реализована на указанном языке .NET платформы на последней версии Visual Studio .NET.

1. **СТРУКТУРА КУРСОВЙ РАБОТЫ**

Текстовая часть курсовой работы оформляется в виде пояснительной записки (ПЗ) в соответствии с требованиями стандарта [13,14].

Функционирование программы иллюстрируется приведёнными прикладными окнами, отображающими особые взаимодействия объектов, представленными в окне в виде некоторых фигур (например, круги, эллипсы или прямоугольники) или изображениями (самолёты, птицы и др.).

Тексты программ помещаются в приложении, в конце пояснительной записки. Тексты программ обязательно снабжаются блочными и построчными комментариями.

Пояснительная записка должна содержать:

Титульный лист;

Лист оглавления;

1 Задание;

2. Уточнение задания

3. Описание разрабатываемой программной системы с точки зрения пользователя.

4. Описание разрабатываемой программной системы с точки зрения программиста. Данный раздел включает следующие подразделы:

4.1.Объектное представление системы.

4.2.События.

4.3.Потоки.

5. Поэтапная разработка программной системы.

7. Описание проблем, возникших при разработке программной системы.

8. Список используемой литературы.

9. Приложение 1. Диаграмма классов разработанной программы.

10. Приложение 2. Текст программы.

**4. СОДЕРЖАНИЕ РАЗДЕЛОВ ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ**

*Задание*. Текст задания должен соответствовать заданию, выданному преподавателем.

*Уточнение задания*. Уточняются пункты задания разработчиком вместе с преподавателем, отражая личные пожелания разработчика и особенности используемой разработчиком среды разработки.

*Описание разрабатываемой программной системы с точки зренияпользователя*. Подробно описывается пользовательский интерфейс разрабатываемой программной системы и использование его пользователем.

*Описание разрабатываемой программной системы с точки зрения программиста*. Данный раздел включает следующие подразделы:

*Объектное представление системы.* Из анализа функционирования разрабатываемой системы она представляется в виде взаимосвязанных объектов. Вкратце описываются типы объектов, их статическая и динамическая взаимосвязь, необходимые события и потоки.

*События.* Описываются все события, уведомления и их использование в разрабатываемой программной системе, их делегаты и обработчики.

*Потоки.* Описываются все потоки и их использование в разрабатываемой программной системе, особенности их работы. Уделяется особое внимание синхронизации параллельно выполняющихся потоков.

*Поэтапная разработка программной системы*. После предварительного анализа и объектно-ориентированного представления разрабатываемой программной системы в предыдущем разделе здесь излагается поэтапная её разработка. Разработка каждого этапа даётся отдельным подразделом. Для каждого этапа приводится его тестирующее приложение.

*Описание проблем, возникших при разработке программной системы*. Описываются ошибки, вызвавшие задержку в разработке программы. Особенно уделяется внимание на проблемах, возникших из-за очередной модификации среды разработки Visual Studio .NET фирмой Microsoft, которая неописана.

*Диаграммы классов разработанной программной системы*. Приводятся диаграмм классов разработанной программы

**5. ОФОРМЛЕНИЕ КУРСОВОЙ РАБОТЫ**

Пояснительная записка выполняется на листах формата

А4( 210\*297 мм) или близкого к нему потребительского формата. Текст набирается машинописным способом на одной стороне листа через 1,5 или 2 межстрочных интервалов в формате Times New Roman, размер шрифта 14. Ширина полей: слева 25, справа 10, сверху и снизу 15-20 мм.

Каждая страница с текстом или иллюстрацией должна быть пронумерована без пропусков, начиная с первого листа, включая титульный лист, содержание и приложения. Номера страниц ставятся в правых нижних углах листов.

Рисунки, схемы и другой иллюстративный материал должен иметь подписи.

Разделы пояснительной записки могут начинаться с нового листа, либо следовать за предыдущим разделом, продолжая его лист. Разделы, подразделы, и пункты нумеруются арабскими цифрами с точкой. Наименование раздела записывается в виде заголовка прописными буквами. Наименование подразделов записываются с абзаца строчными буквами (кроме первой прописной), переносы запрещены.

В содержании последовательно перечисляют номера и заголовки всех разделов и подразделов пояснительной записки. Слово «СОДЕРЖАНИЕ» записывают в виде заголовка прописными буквами. При цитировании материалов из литературных источников, патентной и технической документации обязательно должно быть указание на цитируемый источник и авторов. В перечень литературы заносятся только источники, на которые в тексте имеется ссылка.

Приложение помещают в конце пояснительной записки и размещают в нем вспомогательный материал. Каждое приложение должно начинаться с нового листа с указанием в правом верхнем углу листа слова «ПРИЛОЖЕНИЕ» прописными буквами. Приложение может иметь заголовок, который записывается симметрично тексту прописными буквами.

1. **ВОЗМОЖНЫЕ ЗАДАНИЯ НА КУРСОВУЮ РАБОТУ**

В нижеследующих заданиях каждый объект имеет свой собственный поток с потоковой функцией, циклически изменяющей координаты объекта и выдающей при этом событие с данными о координатах этого объекта и какой-либо иной информацией. Связь между объектами осуществляется только посредством этих событий.

Эти задания являются основой, которая корректируется в процессе выполнения курсовой работы и консультаций. Кроме прикладного окна, иллюстрирующего функционирование и взаимосвязь объектов, представленных в виде образов, програма должна содержать окно с управляющими интерфейсными элементами и окно, выдающее текущее состояние объектов информацию об их взаимодействии. Прямолинейные перемещения объектов желательно заменить иными в зависимости от образов, представленных объектами на экране.

1. Две кольцевые дороги имеют общий участок. По этим дорогам перемещаются n объектов с различными скоростями и в различных направлениях. На общем участке может перемещаться только два объекта. Среди множества свойств каждый объект имеет целочисленное свойство Important. Объекты связаны друг с другом событийно, передавая некоторые данные, при этом объект воспринимает данные только в том случае, если значение его свойства Important ниже значения свойства Important объекта-источника события.
2. По кругу перемещаются n объектов с различными скоростями и в различных направлениях. Встретившись, они приостанавливаются на указанный промежуток времени и передают событие другим объектам о замедлении их движения на время встречи.
3. В прямоугольнике перемещаются n объектов с различными скоростями, отталкиваясь от стенок этого прямоугольника. Оттолкнувшись от стенок, объект сообщает всем другим объектам об изменении их некоторых свойств. Внутри прямоугольника имеется другой прямоугольник. Попав во внутренний прямоугольник, объекты ускоряют свою скорость, восстановив её после выхода из этого прямоугольника.
4. В прямоугольнике перемещаются n объектов с различными скоростями, отталкиваясь от стенок этого прямоугольника. Внутри прямоугольника имеется другой прямоугольник. Попав во внутренний прямоугольник, объект ждёт вождения в этот прямоугольник хотя бы ещё одного объекта и затем, сообщив об этом объектам, находящимся вне внутреннего прямоугольника, продолжает движение.
5. В прямоугольнике горизонтально перемещаются n объектов с различными скоростями и в разных направлениях, отталкиваясь от вертикальных стенок этого прямоугольника. Оказавшись между полосами, объекты меняют свои характеристики (цвет, размер и т.д.). Через заданный промежуток времени на объекты поступает событие, изменяющее свойства объектов. Получив событие, объект может послать событие части или всем объектам.
6. В прямоугольнике горизонтально рисуются n синусоид с заданной частотой и амплитудой, толщиной и цветом. Каждая синусоида представлена объектом. Всякий раз, когда любая синусоида достигают максимума и минимума, возникают события, приводящие к различным изменениям в других синусоидах (цвет, толщина линии и др.).
7. В прямоугольнике снизу вертикально перемещаются n объектов с различными скоростями. Через некоторое случайное время объекты распадаются на указанное количество других объектов, которые разлетаются в разные стороны на некоторое расстояние и исчезают. После исчезновения каждого объекта возникает событие, сообщающее о необходимости изменения свойств других объектов.
8. В прямоугольнике горизонтально перемещаются n объектов с различными скоростями и в разных направлениях, отталкиваясь от вертикальных стенок этого прямоугольника. Встретившись друг с другом, объекты обмениваются их цветом. Оказавшись внутри заданной области, объект сообщает некоторым объектам о необходимости изменения их свойств.
9. Из пункта А в пункт С перемещаются n объектов с различными скоростями и в разных направлениях. Дойдя до пункта В, объекты дожидаются друг друга и затем друг за другом движутся с одинаковой скоростью в одном направлении к пункту А или к пункту В. О прохождении пунктов А, В и С каждый объект сообщает некоторому специальному объекту.
10. В прямоугольнике горизонтально перемещаются n объектов с различными скоростями и в разных направлениях, отталкиваясь от вертикальных стенок этого прямоугольника. Встретившись друг с другом, объекты обмениваются значениями одного из их свойств и сообщают об этом ближайшему объекту.
11. В прямоугольнике перемещаются две совокупности по n и m объектов соответственно с различными скоростями, отталкиваясь от стенок этого прямоугольника. Внутри прямоугольника имеется другой прямоугольник. Попав во внутренний прямоугольник, объекты только одной совокупности продолжают движение, пока все не выйдут из него. И только после этого объекты другой совокупности могут продолжить движение, проходя через внутренний прямоугольник О вхождении во внутренний прямоугольник и выходе из него объекты сообщают специальному объекту, формирующему список объектов, находящихся во внутреннем прямоугольнике.
12. Имеются два круга, связанных между собой двумя линиями (путями). По кругам перемещаются по n и m объектов с различными скоростями и в различных направлениях. Через некоторый промежуток времени объекты из одного круга перемещаются по путям в другой круг, если путь доступен. По одному пути может перемещаться только один объект, а по другому – два. Информация о своём местонахождении объектом передаётся с помощью события при переходе его из одного круга в другой специальному информационному объекту.

1. Имеется большое и малое кольцо, по которым перемещаются объекты. Кольца связаны линией (путём). Каждый объект большого кольца пытается переместиться в малое кольцо и, сделав там два круга, вернуться в большое кольцо. По пути может перемещаться только один объект.
2. В прямоугольнике перемещаются две совокупности по n и m объектов соответственно с различными скоростями. Одна совокупность объектов перемещается вертикально, а другая – горизонтально. Объекты, перемещающиеся вертикально должны пропускать объекты компонента, перемещающиеся горизонтально, приостанавливаясь на указанном расстоянии от них.
3. Один объект движется по окружности, а вокруг него перемещаются n объектов с различными скоростями и в разных направлениях.
4. В прямоугольнике горизонтально перемещаются n объектов с различными скоростями и в разных направлениях, отталкиваясь от вертикальных стенок этого прямоугольника. Через случайное время каждый объект опускается на нижнюю сторону прямоугольника, если он не встретился с одним из особых объектов.
5. В прямоугольнике перемещаются n объектов с различными скоростями, отталкиваясь от стенок этого прямоугольника. Встретившись друг с другом, объекты увеличивают скорость и изменяют свои некоторые свойства.

1. В прямоугольнике перемещаются n объектов с различными скоростями, отталкиваясь от стенок этого прямоугольника. Встретившись друг с другом, один из объектов опускается на нижнюю сторону прямоугольника.
2. В прямоугольнике перемещаются n объектов с различными скоростями, отталкиваясь от стенок этого прямоугольника. Коснувшись расположенного в прямоугольнике круга, объект начинает двигаться по этому кругу.
3. В прямоугольнике перемещаются n объектов с различными скоростями, отталкиваясь от стенок этого прямоугольника. Внутри прямоугольника перемещается внутренний прямоугольник, отскакивая от стен охватывающего его прямоугольника. Оказавшись внутри внутреннего прямоугольника, каждый объект меняет свои свойства.
4. В прямоугольнике двигаются n объектов с различными скоростями, отталкиваясь от стенок этого прямоугольника. Внутри прямоугольника перемещается внутренний прямоугольник, отскакивая от стен охватывающего его прямоугольника. Оказавшись внутри внутреннего прямоугольника, каждый объект меняет свои свойства.

22. В прямоугольнике перемещается другой прямоугольник и двигаются n объектов с различными скоростями, отталкиваясь от стенок этого перемещающегося прямоугольника и стенок внешнего прямоугольника. Только один объект может войти в перемещающийся прямоугольник и продолжить движение.

1. Три кольцевые дороги имеют общий участок. По этим дорогам перемещаются n объектов с различными скоростями и в различных направлениях. На общем участке может перемещаться только один объект.
2. Две кольцевые дороги пересекаются. По этим дорогам перемещаются n объектов с различными скоростями и в различных направлениях. На пересечении дорог находится светофор, который через указанный промежуток времени управляет перемещением объектов через перекрёсток. Среди объектов есть особый объект, который сообщает другим объектам о своих координатах. При приближении этого объекта к другим объектам эти объекты освобождают путь ему, сдвинувшись к краю дороги, а светофор игнорируется этим объектом.
3. По кругу перемещаются n объектов с различными скоростями и в различных направлениях. Встретившись в некоторой точке, они ускоряют движение, пока не встретят ещё один объект, после которого восстанавливают свою скорость и т.д. Объекты узнают о приближении других объектов, получая от них сообщения о координатах.
4. В прямоугольнике перемещаются n объектов с различными скоростями, отталкиваясь от стенок этого прямоугольника. Оттолкнувшись от стенок, объект превращается в два объекта, один из которых через указанное время исчезает. Если перед исчезновением объект приблизится к какому-либо объекту, то они оба исчезают.

**СПИСОК ЛИТЕРАТУРЫ**

а) основная литература

1. Медведев В.И. Особенности объектно-ориентированного программирования на C++/CLI, C# и Java. – Казань: РИЦ «Школа», 2008. – 360 c.: ил. – (Серия «Современная прикладная математика и информатика»).

2. Медведев В.И. Программирование на С++, С++**.**NET/C# и **.**NET компоненты. – Казань: *Мастер* *Лайн*, 2006. – 296 c.: ил.

3. Медведев В.И. Программирование на C++, C++.NET и C# (Серия “Современная прикладная математика и информатика”). – Казань: Мастер Лайн, 2005. – 270 c.: ил.

б) дополнительная литература:

1. Байдачный С.С. .NET Framework. Секреты создания Windows-приложений. – М.: СОЛОН-Пресс, 2004. – 496 с.: ил.

2. Гербердт Шилдт. C#: учебный курс. – СПб.: Питер; К.: Издательская группа BHV, 2003. – 512 c.: ил.

СПб.: Питер, 2002. – 464 с.

3. Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах/Пер. с англ. – М.: Издательско-торговый дом “Русская Редакция”, 2002. – 576 + 624 с.: ил.

4. П. Наутон, Г. Шилдт. Java 2. Наиболее полное руководство в подлиннике.– СПб.: БХВ-Петербург, 2000. – 1072 с.: ил.

5. Рамбо Дж., Якобсон А., Буч Г. UML: специальный справочник. – СПб.: Питер, 2002. – 656 c.

**ПРИЛОЖЕНИЕ . Пример курсовой работы**

В этом разделе приведён пример оформления пояснительной записки применительно к простому заданию, включающему обязательную часть курсовой работы, оцениваемой положительной оценкой. Усложнив это задание совместно с преподавателем в процессе выполнения работы, студент может заслужить более высокую оценку. В зависимости от способностей студента задания на курсовую работу могут быть связаны с научными интересами студентов или с разработкой каких-либо программных систем в рамках кафедральной научно-методической работы и т.д. Но в любом случае задание должно быть согласовано с преподавателем, и разрабатываемая программа должна быть реализована на языках C# и Java. Введение в курсовую работу изображений, дополнительных интерфейсных элементов, обогащающих управление программой, может привести к повышению оценки

Задание сформулировано следующим образом:

В прямоугольнике перемещаются 2 объекта шаров с различными скоростями, отталкиваясь от стенок этого прямоугольника. Оттолкнувшись от стенок, объекты шары сообщают об этом специальному объекту шару, который при этом “вспыхивает” на 2 сек.

Министерство общего и профессионального образования

Российской Федерации

## Казанский государственный технический университет

имени. А.Н. Туполева

------------------------------------------------------------------------------------------

Кафедра АСОИУ

Курсовая работа

по дисциплине «Объектно-ориентированное программирование»

(задание № 0)

#### Исполнитель: студент группы 4208 И.И. Иванов

Руководитель: доц. кафедры АСОИУ В.И. Медведев

##### Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2009 г.

### Казань 2009

**Содержание**

1. Задание

2. Уточнение задания

3. Описание разрабатываемой программы с точки зрения пользователя.

4. Описание разрабатываемой программы с точки зрения программиста. Данный раздел включает следующие подразделы:

* 1. Объектное представление программы.
  2. События.
  3. Потоки.

5. Поэтапная разработка программной системы.

5.1. Этап 1. Разработка класса обычного объекта

5.2. Этап 2. Разработка класса Balls

5.3. Этап 3. Разработка события ev

5.4. Этап 4. Разработка прикладного окна

5.5. Этап 5. Разработка дочернего окна

5.6. Этап 6. Добавление элемента списка

5.7. Этап 7. Разработка специального объекта

6. Описание проблем, возникших при разработке программной системы.

7. Список используемой литературы.

8. Приложение 1. Диаграмма классов разработанной программы.

9. Приложение 2. Текст программы на.языке C#.

10. Приложение 2. Текст программы на.языке Java.

### **1. Задание**

В прямоугольнике перемещаются 2 объекта шаров с различными скоростями, отталкиваясь от стенок этого прямоугольника. Оттолкнувшись от стенок, объекты-шары сообщают об этом специальному объекту-шару, который при этом “вспыхивает” на 2 сек.

1. **Уточнение задания**

Программа состоит их прикладного и дочернего окон. В области клиента прикладного окна перемещаются одноцветные шары, отталкиваясь от его сторон. Шары представлены в виде окружностей фиксированной толщины. В левом верхнем левом углу прикладного окна находится или появляется неподвижный специальный желтый шар. При отскакивании каждого обычного шара от стенок прикладного окна толщина круга специального шара увеличивается и через 2 сек. восстанавливается (для языка C#) или желтый круг появляется на 2 мс (для языка Java). Объект каждого шара содержит собственный поток, так что функционирование приложения сводится к функционированию совокупности потоковых объектов-шаров. Специальный шар и обычные шары взаимосвязаны посредством событий или уведомлений

Дочернее окно содержит кнопку и список цветов, После выбора в списке некоторого цвета и нажатия на эту кнопку цвет всех обычных шаров изменяется на выбранный цвет. При этом движение шаров при каждом нажатии на кнопку приостанавливается или возобновляется.

Программу реализовать в среде Visual Studio .NET на языке C# и на языке J#(Java).

**3. Описание разрабатываемой программы с точки зрения пользователя**

Программа представлена прикладным и дочерним окнами. В прикладном окне перемещаются 2 окружности, и в левом верхнем углу находится желтая окружность, которая вспыхивает (увеличивается ширина круга) при отталкивании шаров от стенок прикладного окна.

Дочернее окно содержит кнопку и список цветов. Если выбрать в списке некоторый цвет и нажать на кнопку, то цвет шаров изменится на выбранный цвет.

Диаграмма вариантов использования языка UML изображает действия, выполняемые пользователем программы. С точки зрения пользователя приложение предполагает три варианта использования:

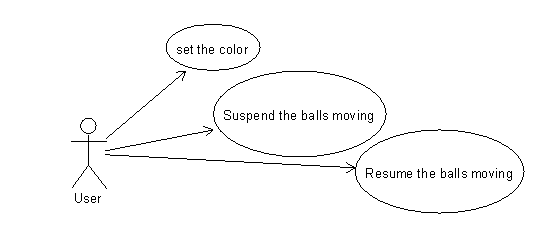
- установить цвет шаров (set the color),

- приостановить движение шаров (suspend the balls moving),

- возобновить движение шаров (resume the balls moving).

Если шары перемещались, то после нажатия кнопки они приостановятся. Если шары были неподвижны, то после нажатия кнопки они продолжат своё перемещение.

Диаграмма вариантов использования представлена на рис. 3.1.



**Рис. 3.1.** Диаграмма вариантов использования Balls-приложения

Прекращение выполнения программы происходит при закрытии прикладного окна.

**4. Описание разрабатываемой программы с точки зрения программиста**

**4.1. Объектное представление программы**

Программа включает объекты разных типов. Главными объектами программы являются объект прикладного окна класса User и объект дочернего окна класса Form или Frame. При создании объекта прикладного окна создаётся дочернее окно, 2 объекта обычных шаров класса Ball и один объект специального шара класса SpecialBall. Объекты обычных шаров сразу начинают перемещаться в границах прикладного окна, а специальный объект реагировать на отталкивание обычных шаров от стенок окна.

Каждый обычный объект шара функционирует независимо от других шаров в соответствии с потоковой функцией BallFunc или run() его собственного потока. Оттолкнувшись от стенки окна, объект каждого обычного шара генерирует событие evSpecial или уведомлдение, на которое подписан обработчик HandlerEvSpecial() или функция update() специального объекта шара. Обработчик запускает потоковую функцию BallFunc или run() этого специального объекта. В результате толщина d кольца специального объекта увеличивается и через 2 сек. восстанавливается, либо рисуется желтый круг в теченик 2 мс.

Дочернее окно содержит объект кнопки pBut и объект pListBox списка.

**4.2. События**

В программе имеются несколько событий.

Объект обычного шара генерирует событие ev или уведомление при каждом изменении его координат. Данное событие используется при перерисовке шаров в прикладном окне. На эти события всех обычных объектов должен быть подписан один обработчик HandlerEv или update(), инициирующий перерисовку окна.

Кроме этого события и уведомления объект обычного шара генерирует другое событие evSpecial или уведомление при каждом отталкивании шара от стенок прикладного окна. На это событие или уведомление должен быть подписан обработчик HandlerEvSpecial или update() специального объекта.

Также в программе используется предопределённые событие Click, генерируемое при нажатии кнопки, и событие, генерируемое при выборе цвета из интерфейсного элемента списка.

**4.3. Потоки**

В программе применяются кроме основного потока, реализуемого функцией Main() или main(), ещё рабочие потоки. Каждый из n+1 объектов шаров располагает собственным потоком с потоковой функцией, причём если функционирование потоковых функций обычных объектов шаров совпадают – они перемещают объекты шаров, то потоковая функция специального объекта шара отличается от них. Эта потоковая функция выполняется в течение 2 сек, поддерживая утолщение кольца или его отображение в окне.

1. **Поэтапная разработка программной системы**

Разработка приложения будет осуществляться поэтапно. Вначале разработаем класс Ball потокового объекта и протестируем его. Затем создадим и отладим класс Balls потоковых объектов, который управляет шарами. Для просмотра результатов тестирования мы воспользуемся консолью.

В дальнейшем мы наследуем класс Balls из базового класса Form или Frame и будем отображать шары в области клиента прикладного окна (в форме) в виде кругов одного цвета. Нам придётся модифицировать класс Ball потокового объекта, добавив в него координаты и их приращения. Для обеспечения перерисовки шаров мы воспользуемся событием или уведомлением, которое будет генерироваться после каждого изменения их координат.

Убедившись в совместном функционировании шаров и их отображении в области клиента прикладного окна, мы перейдём к разработке дочернего окна, на котором разместим управляющие элементы – кнопку и список.

При реализации Balls –приложения мы одновременно будем использовать консоль и окна. Отладочная информация на консоли отразит полезную информацию о внутреннем функционировании приложения. Отладив Balls–приложение, следует отменить выдачу на консоль.

На последнем этапе мы добавим специальный объект шара класса SpecialBall с обработчиком, а в класс Ball обычного объекта шара поместим особое событие и свяжем события и уведомления объектов класса Ball с этим обработчиком.

Для большей наглядности при создании приложений мы воспользуемся диаграммами классов языка UML.

В программе обычные объекты объединим в одномерном массиве длиной n= 2. При необходимости, изменив длину массива, можно создать необходимое количество обычных объектов.

**5.3.1. Этап 1. Разработка класса обычного объекта**

На первом этапе создадим основной класс Ball потокового объекта, который включает самые необходимые данные:

num – номер обычного объекта,

life – признак жизни потока (*life= true* - поток выполняется, *life= false*

поток завершён),

run – признак выполнения потока (*run= true* - поток выполняется,

*run= false* - поток приостановлен),

thread – поток класса *Thread*.

Включим в класс *Ball* функции:

Ball ( ) – конструктор класса,

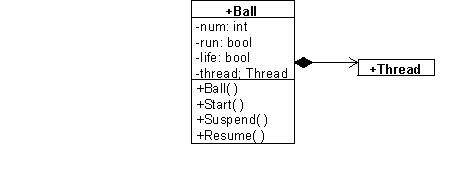
~Balls ( ) – деструктор класса,

Start ( ) – запустить обычный объект,

Suspend ( ) – приостановить выполнение обычного объекта,

Resume ( ) – возобновить выполнение обычного объекта.

Диаграмма класса *Ball* представлена на рис. 5.3.1.1.



**Рис. 5.3.1.1.** Диаграмма классов Balls–приложения первого этапа разработки

В примере 5.3.1.1 приведена реализация первого этапа на языке *C#*.

В примере 5.3.1.2 приведена реализация первого этапа на языке Java.

**Пример 5.3.1.1. Реализация Balls–приложения первого этапа на языке C#.**

////////////////////

//***C#*** **File** *BallsCs1.cs*

using System;

using System.Threading;

class Ball // Класс обычного объекта

{

int num;

Thread thread;

bool run;

bool life;

public Ball (int Num) // Конструктор

{

num= Num; life= false; run= false;

Start ( );

}

//---------------

~Ball ( ) {life= false;} // Деструктор

//---------------

public void Start ( ) // Стартовать

{

if (!run)

{

run= true;

life= true;

thread= new Thread (new ThreadStart (BallFunc));

thread.Start ( );

}

}

//---------------

public void Suspend ( ) // Приостановить

{

if (run)

{

run= false;

thread.Suspend ( );

}

}

//---------------

public void Resume ( ) // Возобновить

{

if (!run)

{

run= true;

thread.Resume ( );

}

}

//---------------

void BallFunc ( ) // Выполнить поток

{

int n= 0;

while (life)

{

Console.Write ("Объект {0}", num.ToString ( ));

Console.WriteLine (" n= {0}", n.ToString ( ));

n++;

Thread.Sleep (10);

}

Console.Write ("Объект {0}", num.ToString ( ));

Console.WriteLine (" завершился");

}

};

//-------------------------------------

class BallsCs1

{

static void Main (string[] args)

{

Ball ball1= new Ball (1); // Создать первый обычный объект

Ball ball2= new Ball (2); // Создать второй обычный объект

Console.WriteLine ("Объекты функционируют");

Thread.Sleep (20); // Объекты функционируют 20 мс

ball1.Suspend ( ); // Приостановить первый обычный объект

ball2.Suspend ( ); // Приостановить второй обычный й объект

Console.WriteLine ("Объекты приостановлены и возобновлены");

ball1.Resume ( ); // Возобновить выполнение первого объекта

ball2.Resume ( ); // Возобновить выполнение второго объекта

Thread.Sleep (20); // Объекты функционируют 20 мс

ball1.Suspend ( ); // Приостановить первый обычный объект

ball2.Suspend ( ); // Приостановить второй обычный объект

}

}

/\*

**Результат:**

Объекты функционируют

Объект 1 n= 0

Объект 2 n= 0

Объект 1 n= 1

Объект 2 n= 1

Объекты приостановлены и возобновлены

Объект 1 n= 2

Объект 2 n= 2

Объект 1 n= 3

Объект 2 n= 3

\*/

**Пример 5.3.1.2. Реализация Balls–приложения первого этапа на языке Java.**

package BallsCj1;

////////////////////

//C# File BallsCj1

class Ball implements Runnable // Класс обычного объекта

{

int num;

Thread thread;

boolean run;

boolean waiting= false; // Признак ожидания выполнения потока

boolean life;

public Ball (int Num) // Конструктор

{

num= Num; life= false; run= false;

Start ( );

}

public void Start ( ) // Стартовать

{

if (!run)

{

run= true;

life= true;

thread= new Thread (this);

thread.start ( ); // Запустить поток

}

}

//---------------

synchronized public void Suspend ( ) // Приостановить

{

if (run)

{

run= false;

waiting= true; // Перейти потоку в состояние ожидания

}

}

//---------------

synchronized public void Resume ( ) // Возобновить

{

if (!run)

{

run= true;

waiting=false; // Выйти потоку из состояния ожидания

notify ();

}

}

//---------------

public void run ( ) // Выполнить поток

{

int n= 0;

while (life)

{

System.out.print ("object "+num);

System.out.println (" n= "+n);

n++;

try

{

Thread.sleep (10);

synchronized (this)

{

if (waiting)

wait ();

}

}

catch(InterruptedException iE){}

}

System.out.print ("object "+num);

System.out.println (" finished ");

}

}

//-------------------------------------

class BallsCj1

{

public static void main ()

{

Ball ball1= new Ball (1); // Создать первый обычный объект

Ball ball2= new Ball (2); // Создать второй обычный объект

System.out.println("objects are working");

// Объекты функционируют 20 мс

try

{

Thread.sleep (20);

}

catch(InterruptedException iE){}

ball1.Suspend ( ); // Приостановить первый обычный объект

ball2.Suspend ( ); // Приостановить второй обычный й объект

System.out.println("objects are suspend and are resumed");

ball1.Resume ( ); // Возобновить выполнение первого объекта

ball2.Resume ( ); // Возобновить выполнение второго объекта

// Объекты функционируют 20 мс

try

{

Thread.sleep (20);

}

catch(InterruptedException iE){}

ball1.Suspend ( ); // Приостановить первый обычный объект

ball2.Suspend ( ); // Приостановить второй обычный объект

}

}

/\*

Результат:

objects are working

object 1 n= 0

object 2 n= 0

object 1 n= 1

object 2 n= 1

objects are suspend and are resumed

object 1 n= 2

object 2 n= 2

object 1 n= 3

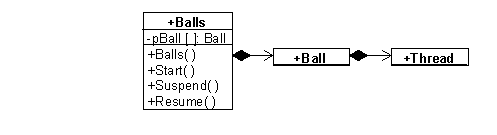
object 2 n= 3

\*/

**5.3.2. Этап 2. Разработка класса Balls**

На этом этапе реализуем класс Balls, содержащий массив обычных объектов класса Ball и осуществляющий их одновременный запуск, а также приостановку и возобновление функционирования. Дополнительно к указателю pBall на массив обычных объектов, класс Balls содержит функции, выполняющие запуск (Start), приостановку выполнения (Suspend) и возобновление выполнения (Resume) всей совокупности обычных объектов.

Диаграмма классов Balls–приложения второго этапа изображена на рис. 5.3.2.1.



**Рис 5.3.2.1.**  Диаграмма классов Balls–приложения второго этапа

Реализация второго этапа разработки Balls–приложения на языке C# дана в примере 5.3.2.1.

Реализация второго этапа разработки Balls–приложенияна языке Java дана в примере 5.3.2.2.

**Пример 5.3.2.1. Реализация второго этапа разработки Balls–приложения на языке C#.**

////////////////////

// ***C#*** **File** *BallsCs2.cs*

// Файл *BallsCs2.cs* полностью включает файл *BallsCs1.cs*, дополнив его

// нижеследующим описанием класса *Balls* и изменённой функцией *Main*

**. . .**

//-------------------------------------

class Balls // Класс объектов

{

Ball [] pBall;

public Balls ( )

{

pBall= new Ball [2];

for (int i= 0; i < 2; i++)

pBall[i]=new Ball (i);

Start ( );

}

//---------------

public void Start ( ) // Стартовать

{

for(int i=0; i < 2; i++)

{

pBall[i]. Start ( );

}

}

//---------------

public void Suspend ( ) // Возобновить

{

for (int i=0; i < 2; i++)

pBall[i].Suspend ( );

}

//---------------

public void Resume ( ) // Приостановить

{

for (int i= 0; i < 2; i++)

pBall[i].Resume ( );

}

};

//-------------------------------------

class BallsCs2

{

static void Main (string[] args)

{

Balls pBalls= new Balls ( ); // Создать объект класса *Balls*

Console.WriteLine ("Объекты функционируют");

Thread.Sleep (20); // Объект функционирует 20 мс

pBalls.Suspend ( ); // Приостановить функционирование

Console.WriteLine("Объекты приостановлены и возобновлены");

pBalls.Resume ( ); // Возобновить функционирование

Thread.Sleep (20); // Объект функционирует 20 мс

pBalls.Suspend ( ); // Приостановить функционирование

}

}

/\*

**Результат:**

Потоковые объекты функционируют

Объект 1 n= 0

Объект 2 n= 0

Объект 1 n= 1

Объект 2 n= 1

Объекты приостановлены и возобновлены

Объект 2 n= 2

Объект 1 n= 2

Объект 1 n= 3

Объект 2 n= 3

\*/

**Пример 5.3.2.2. Реализация второго этапа разработки Balls–приложения на языке Java.**

package BallsCj2;

////////////////////

//C# File BallsCj2

class Ball implements Runnable // Класс обычного объекта

{

int num;

Thread thread;

boolean run;

boolean waiting= false;

boolean life;

public Ball (int Num) // Конструктор

{

num= Num; life= false; run= false;

Start ( );

}

public void Start ( ) // Стартовать

{

if (!run)

{

run= true;

life= true;

thread= new Thread (this);

thread.start ( );

}

}

//---------------

synchronized public void Suspend ( ) // Приостановить

{

if (run)

{

run= false;

waiting=true;

}

}

//---------------

synchronized public void Resume ( ) // Возобновить

{

if (!run)

{

run= true;

waiting=false;

notify();

}

}

//---------------

public void run ( ) // Выполнить поток

{

int n= 0;

while (life)

{

System.out.print("object "+num);

System.out.println(" n= "+n);

n++;

try

{

Thread.sleep (10);

synchronized(this)

{

if(waiting)

wait();

}

}

catch(InterruptedException iE){}

}

System.out.print("object "+num);

System.out.println(" finished ");

}

}

//-------------------------------------

**class Balls** // Класс объектов

**{**

**Ball [] pBall;**

**public Balls ( )**

**{**

**pBall= new Ball [2];**

**for (int i= 0; i < 2; i++)**

**pBall[i]=new Ball (i+1);**

**Start ( );**

**}**

//---------------

**public void Start ( )** // Стартовать

**{**

**for(int i=0; i < 2; i++)**

**{**

**pBall[i]. Start ( );**

**}**

**}**

//---------------

**public void Suspend ( )** // Возобновить

**{**

**for (int i=0; i < 2; i++)**

**pBall[i].Suspend ( );**

**}**

//---------------

**public void Resume ( )** // Приостановить

**{**

**for (int i= 0; i < 2; i++)**

**pBall[i].Resume ( );**

**}**

**};**

class BallsCj1

{

public static void main ()

{

**Balls pBalls= new Balls ( );** // Создать объекты класса Balls

System.out.println("objects are working");

// Объекты функционируют 20 мс

try

{

Thread.sleep (20);

}

catch(InterruptedException iE){}

**pBalls.Suspend ( );** // Приостановить объекты

System.out.println("objects are suspend and are resumed");

**pBalls.Resume ( );** // Возобновить выполнение объектов

// Объекты функционируют 20 мс

try

{

Thread.sleep (20);

}

catch(InterruptedException iE){}

**pBalls.Suspend ( );** // Приостановить объекты

}

}

/\*

Результат:

objects are working

object 1 n= 0

object 2 n= 0

object 1 n= 1

object 2 n= 1

objects are suspend and are resumed

object 1 n= 2

object 2 n= 2

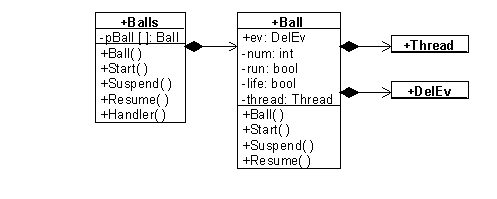
object 1 n= 3

object 2 n= 3

\*/

**5.3.3. Этап 3. Разработка события ev и уведомления**

Включим в класс Ball обычного объекта событие ev и уведомление, сигнализирующее об очередном цикле выполнения потока. Это событие и уведомление обрабатывается функцией HandlerEv или update() класса Balls, что будет использовано при перерисовке шаров в области клиента прикладного окна на четвёртом этапе. На рис. 5.3.3.1 и в примерах 5.3.3.1 и 5.3.3.2 представлены диаграмма классов третьего этапа и программы, реализующие её.



**Рис. 5.3.3.1.**  Диаграмма классов Balls–приложения третьего этапа

**Пример 5.3.3.1. Реализация третьего этапа разработки Balls–приложения.**

////////////////////

// ***С#*** **File** *BallsCs3.cs*

// Файл *BallsCs3.cs* полностью включает файл *BallsCs2.cs*, дополнив

// его объявлением события *ev* в классе *Ball* и функцией *HandlerEv*

// класса *Balls*, обеспечивающей обработку этого события.

// Изменения файла *BallsCs2.cs* выделены жирным шрифтом.

**. . .**

//-------------------------------------

**delegate void delEv ( );** // Объявление типа *delEv* делегата события *ev*

class Ball

{

**public event delEv ev;** // Объявление события *ev*

**. . .**

//---------------

void BallFunc ( ) // Выполнить поток

{

int n= 0;

while (life)

{

Console.Write ("Объект {0}", num.ToString ( ));

Console.WriteLine (" n= {0}", n.ToString ( ));

n++;

**if (ev != null)** // Если событие активизировано, то

**{**

**Console.WriteLine (" Event");**

**ev ( );** //свершить событие

**}**

Thread.Sleep (10);

}

Console.Write ("Объект {0}", num.ToString ( ));

Console.WriteLine (" завершён");

}

};

//-------------------------------------

class Balls // Класс потоковых объектов

{

Ball [] pBall;

**public void HandlerEv ( )** // Обработчик события *ev*

**{**

**Console.WriteLine (" HandlerEv");**

**}**

public Balls ( ) // Конструктор

{

pBall= new Ball [2];

for (int i= 0; i < 2; i++)

{

pBall[i]=new Ball (i);

**pBall[i].ev+= new delEv(HandlerEv );** // Добавить

}

Start ( );

}

**. . .**

}

class BallsCs3

{

static void Main (string[] args)

{

Balls pBalls= new Balls ( ); // Создать объект класса Balls

Console.WriteLine("Объекты функционируют");

Thread.Sleep (20); // Объект функционирует 20 мс

pBalls.Suspend ( ); // Приостановить функционирование

}

}

**Результат:**

Объекты функционируют

Объект 0 n= 0

Event

HandlerEv

Объект 1 n= 0

Event

HandlerEv

Объект 0 n= 1

Event

HandlerEv

Объект 1 n= 1

Event

HandlerEv

\*/

**Пример 5.3.3.2. Реализация третьего этапа разработки Balls–приложения на языке Java.**

package BallsCj3;

////////////////////

//C# File BallsCj3

**import java.util.\*;**

**class Obs**

**{**

**public int num, n;**

**public Obs (int num, int n)**

**{this.num= num; this.n= n;}**

**int getNumber(){return num;}**

**int getN(){return n;}**

**}**

class Ball **extends Observable** implements Runnable // Класс обычного объекта

{

int num;

Thread thread;

boolean run;

boolean waiting= false;

boolean life;

public Ball (int Num) // Конструктор

{

num= Num; life= false; run= false;

Start ( );

}

public void Start ( ) // Стартовать

{

if (!run)

{

run= true;

life= true;

thread= new Thread (this);

thread.start ( );

}

}

//---------------

synchronized public void Suspend ( ) // Приостановить

{

if (run)

{

run= false;

waiting=true;

}

}

//---------------

synchronized public void Resume ( ) // Возобновить

{

if (!run)

{

run= true;

waiting=false;

notify();

}

}

//---------------

public void run ( ) // Выполнить поток

{

int n= 0;

while (life)

{

System.out.print("object "+num);

System.out.println(" n= "+n);

n++;

// Уведомить

**setChanged();**

**notifyObservers(new Obs(num, n));**

try

{

Thread.sleep (10);

synchronized(this)

{

if(waiting)

wait();

}

}

catch(InterruptedException iE){}

}

System.out.print("object "+num);

System.out.println(" finished ");

}

}

//-------------------------------------

class Balls **implements Observer** // Класс объектов

{

Ball [] pBall;

public Balls ( )

{

pBall= new Ball [2];

for (int i= 0; i < 2; i++)

{

pBall[i]=new Ball (i+1);

**pBall[i].addObserver(this);**

}

Start ( );

}

//---------------

public void Start ( ) // Стартовать

{

for(int i=0; i < 2; i++)

{

pBall[i]. Start ( );

}

}

//---------------

public void Suspend ( ) // Возобновить

{

for (int i=0; i < 2; i++)

pBall[i].Suspend ( );

}

//---------------

public void Resume ( ) // Приостановить

{

for (int i= 0; i < 2; i++)

pBall[i].Resume ( );

}

**public void update (Observable r, Object ob)** // Обработчик события ev

**{**

**Obs obs= (Obs) ob;**

**System.out.print("object num= "+ obs.num);**

**System.out.println(" n= "+ obs.n);**

**}**

};

class BallsCj1

{

public static void main ()

{

Balls pBalls= new Balls ( ); // Создать объекты класса Balls

System.out.println("objects are working");

// Объекты функционируют 20 мс

try

{

Thread.sleep (20);

}

catch(InterruptedException iE){}

pBalls.Suspend ( ); // Приостановить объекты

System.out.println("objects are suspend and are resumed");

pBalls.Resume ( ); // Возобновить выполнение объектов

// Объекты функционируют 20 мс

try

{

Thread.sleep (20);

}

catch(InterruptedException iE){}

pBalls.Suspend ( ); // Приостановить объекты

}

}

/\*

Результат:

objects are working

object 1 n= 0

**object num= 1 n= 0**

object 2 n= 0

**object num= 2 n= 0**

object 1 n= 1

**object num= 1 n= 1**

object 2 n= 1

**object num= 2 n= 1**

objects are suspend and are resumed

object 1 n= 2

**object num= 1 n= 2**

object 2 n= 2

**object num= 2 n= 2**

object 1 n= 3

**object num= 1 n= 3**

object 2 n= 3

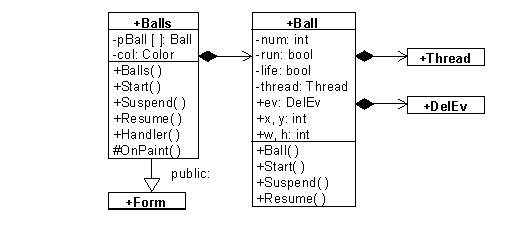
**object num= 2 n= 3**

\*/

**5.3.4. Этап 4. Разработка прикладного окна**

Наступил этап разработки, когда появится прикладное окно приложения с перемещающимися шарами.

Наследовав класс Form или Frame, класс Balls существенно расширит свою функциональность. Теперь можно создать прикладное окно (основную форму) и дочернее окно (дочернюю форму). Но дочернее окно будет создано потом в классе User, порождённым из класса Balls. Диаграмма классов и реализация этого этапа представлены на рис. 5.3.4.1 и в примерах 5.3.4.1 и 5.3.4.2.



**Рис. 5.3.4.1.** Диаграмма классов Balls–приложения четвёртого этапа

**Пример 5.3.4.1. Реализация четвёртого этапа разработки Balls–приложения на языке C#.**

////////////////////

// ***C#*** **File** *Balls4Cs.cs*

// Файл *BallsCc4.cs* полностью включает файл *BallsCs3.cs*, дополнив его

// данными класса *Ball* икласса *Balls*, необходимыми для рисования шаров.

// В результате наследования класса *Balls* из базового класса *Form* появилось

// прикладное окно, свойства которого установлены в конструкторе класса

// *Balls*. Переопределена функция *OnPaint* перерисовки. Функция *HandlerEv*,

// реагируя на событие *ev* из потоков, вызывает перерисовку шаров.

// Изменения файла *BallsCs3.cs* выделены жирным шрифтом

//

using System;

**using System.Drawing;**

**using System.Windows.Forms;**

using System.Threading;

delegate void delEv ( );

class Ball // Класс обычного объекта

{

public event delEv ev;

**public int x, y;**

**public int w, h;**

**public int dx, dy;**

**. . .**

**public Ball (int Num , int X, int Y, int Dx, int Dy)** // Конструктор

{

num= Num; life= false; run= false;

**w= 100; h= 100; x= X; y= Y; dx= Dx; dy= Dy;**

Start();

}

//---------------

**. . .**

void BallFunc ( ) // Выполнить поток

{

int n= 0;

while ((life)) //&&(n<=2))

{

Console.Write ("Объект {0}", num.ToString ( ));

Console.WriteLine (" n= {0}", n.ToString ( ));

n++;

**x += dx; y += dy;**

**if (x>w || x<0) dx= -dx;**

**if (y>h || y<0) dy= -dy;**

if (ev != null)

{

Console.WriteLine (" Event");

ev ( );

}

Thread.Sleep (100);

}

Console.Write ("Объект {0}", num.ToString ( ));

Console.WriteLine (" завершён");

}

}

//-------------------------------------

class Balls **: Form**

{

Ball [] pBall;

**Color col;**

public Balls ( )

{

**col= System.Drawing.Color.FromArgb (0, 0, 255);**

pBall= new Ball [2];

for (int i= 0; i < 2; i++)

{

**pBall[i]=new Ball (i, i\*3+10,**

**1\*3+15, i\*3+5, i\*3+5);**

pBall[i].ev+= new delEv(HandlerEv );

}

**Text= "Balls";**

Start ( );

}

**. . .**

**protected override void OnPaint (PaintEventArgs arg)**

**{**

**base.OnPaint (arg);**

**for (int i= 0; i < 2; i++)**

**{**

**arg .Graphics.DrawEllipse**

**(new Pen (col), pBall[i].x,**

**pBall[i].y, 20, 20);**

**pBall[i].w= Width;**

**pBall[i].h= Height;**

**}**

**}**

void HandlerEv ( )

{

Console.WriteLine (" HandlerEv");

**Invalidate ( );**

}

//-------------------------

static void Main()

{

**Application.Run(new Balls ( ));**

}

}

/\*

**Результат:**

*Появилось прикладное окно с двумя нарисованными шарами*.

Объект 0 n= 0

Event

HandlerEv

Объект 1 n= 0

Event

HandlerEv

Объект 0 n= 1

Event

HandlerEv

Объект 1 n= 1

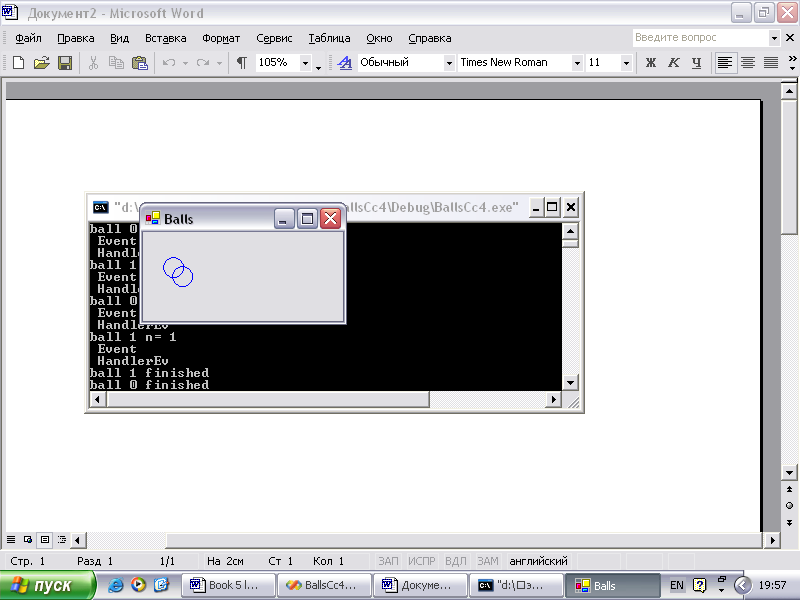
Event

HandlerEv

Объект 0 завершился

Объект 1 завершился

\*/



**Рис. 5.3.4.1.** Прикладное окно с консолью Balls–приложения 4 этапа

**Пример 5.3.4.2. Реализация четвёртого этапа разработки Balls–приложения на языке Java.**

package BallsCj4;

////////////////////

//C# File BallsCj4

import java.util.\*;

import java.awt.\*;

class Obs

{

public int num, n;

public Obs (int num, int n)

{this.num= num; this.n= n;}

int getNumber(){return num;}

int getN(){return n;}

}

class Ball extends Observable implements Runnable // Класс обычного объекта

{

int num;

Thread thread;

boolean run;

boolean waiting= false;

boolean life;

**public int x, y;**

**public int w, h;**

**public int dx, dy;**

public Ball (int Num , int X, int Y, int Dx, int Dy)// Конструктор

{

num= Num; life= false; run= false;

**w= 100; h= 100; x= X; y= Y; dx= Dx; dy= Dy;**

Start ( );

}

public void Start ( ) // Стартовать

{

if (!run)

{

run= true;

life= true;

thread= new Thread (this);

thread.start ( );

}

}

//---------------

synchronized public void Suspend ( ) // Приостановить

{

if (run)

{

run= false;

waiting=true;

}

}

//---------------

synchronized public void Resume ( ) // Возобновить

{

if (!run)

{

run= true;

waiting=false;

notify();

}

}

//---------------

public void run ( ) // Выполнить поток

{

int n= 0;

while ((life)) //&&(n<=2))

{

System.out.print("object "+num);

System.out.println(" n= "+n);

n++;

x += dx; y += dy;

if (x>w || x<0) dx= -dx;

if (y>h || y<0) dy= -dy;

// Уведомить

setChanged();

notifyObservers(new Obs(num, n));

try

{

Thread.sleep (10);

synchronized(this)

{

if(waiting)

wait();

}

}

catch(InterruptedException iE){}

}

System.out.print("object "+num);

System.out.println(" finished ");

}

}

//-------------------------------------

class Balls extends Frame implements Observer // Класс объектов

{

Ball [] pBall;

**Color col;**

public Balls ( )

{

**col= Color.red;**

pBall= new Ball [2];

for (int i= 0; i < 2; i++)

{

pBall[i]=new Ball (i, i\*3+10,

1\*3+15, i\*3+5, i\*3+5);

pBall[i].addObserver(this);

}

**this.setTitle("Balls");**

**this.setSize(400, 300);**

**this.show();**

Start ( );

}

//---------------

public void Start ( ) // Стартовать

{

for(int i=0; i < 2; i++)

{

pBall[i]. Start ( );

}

}

//---------------

public void Suspend ( ) // Возобновить

{

for (int i=0; i < 2; i++)

pBall[i].Suspend ( );

}

//---------------

public void Resume ( ) // Приостановить

{

for (int i= 0; i < 2; i++)

pBall[i].Resume ( );

}

public void update (Observable r, Object ob) // Обработчик события ev

{

Obs obs= (Obs) ob;

System.out.print("object num= "+ obs.num);

System.out.println(" n= "+ obs.n);

**repaint();**

}

**public void paint (Graphics g)**

**{**

**super.paint (g);**

**g.setColor(col);**

**for (int i= 0; i < 2; i++)**

**{**

**g.drawOval(pBall[i].x, pBall[i].y, 20, 20);**

**pBall[i].w= this.getSize().width;**

**pBall[i].h= this.getSize().height;**

**}**

**}**

}

class BallsCj1

{

public static void main ()

{

Balls pBalls= new Balls ( ); // Создать объекты класса Balls

pBalls.Start(); //!!

System.out.println("objects are working");

// Объекты функционируют 20 мс

try

{

Thread.sleep (20);

}

catch(InterruptedException iE){}

pBalls.Suspend ( ); // Приостановить объекты

System.out.println("objects are suspend and are resumed");

pBalls.Resume ( ); // Возобновить выполнение объектов

// Объекты функционируют 20 мс

try

{

Thread.sleep (20);

}

catch(InterruptedException iE){}

//pBalls.Suspend ( ); // Приостановить объекты //!!

}

}

/\*

Результат:

**Появилось прикладное окно с двумя нарисованными шарами.**

objects are working

object 1 n= 0

object num= 1 n= 0

object 2 n= 0

object num= 2 n= 0

object 1 n= 1

object num= 1 n= 1

object 2 n= 1

object num= 2 n= 1

objects are suspend and are resumed

object 1 n= 2

object num= 1 n= 2

object 2 n= 2

object num= 2 n= 2

object 1 n= 3

object num= 1 n= 3

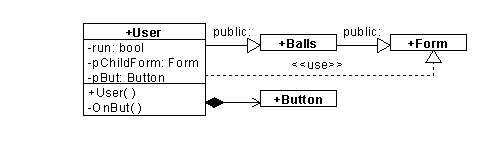
object 2 n= 3

object num= 2 n= 3

\*/

**5.3.5. Этап 5. Разработка дочернего окна с кнопкой**

Создадим дочернее окно с кнопкой, управляющей перемещением шаров в области клиента прикладного окна. Диаграмма классов и реализация пятого этапа представлены на рис. 5.3.5.1 и в примере 5.3.5.1.



**Рис 5.3.5.1.** Диаграмма классов Balls–приложения пятого этапа

**Пример 5.3.5.1. Реализация пятого этапа разработки Balls–приложения на языке C#.**

////////////////////

// ***C#*** **File** *BallsCs5.cs*

// Файл *BallsCs5.cs* полностью включает файл *BallsCs4.cs*, дополнив его

// новым классом *User*, порождённым из класса *Balls*. Класс *User* содержит

// управляющий элемент–кнопку, нажатие на которую приостанавливает

// или возобновляет перемещение шаров.

// Изменения файла *BallsCs4.cs* выделены жирным шрифтом

**. . .**

//-------------------------------------

class Ball // Класс обычного объекта

{ **. . .**

void BallFunc ( )

{

int n= 0;

while **(life)**

{

**. . .**

}

Console.Write ("Объект {0}", num.ToString ( ));

Console.WriteLine (" завершён");

}

} **. . .**

//-------------------------------------

**class User : Balls** // Класс пользователя шаров

**{**

**private bool run;**

**Form pChildForm;**

**Button pBut;**

**public User ( )**

**{**

**run= true;**

//---------------

**pBut= new Button();**

**pBut.Location = new Point (32, 24);**

**pBut.Name = "pBut";**

**pBut.Size = new System.Drawing.Size (32, 24);**

**pBut.Text = "OK";**

**pBut.Click += new EventHandler (OnBut);**

**ClientSize = new System.Drawing.Size (200, 150);**

//---------------

**pChildForm= new Form ( );**

**pChildForm.Location= new Point (250, 10);**

**pChildForm.Size= new Size (250, 250);**

**pChildForm.Text= "User";**

**pChildForm.Show ( );**

**pChildForm.Controls.Add (pBut);**

**pChildForm.ClientSize = new System.Drawing.Size (200, 62);**

**}**

//---------------

**void OnBut (object obj, EventArgs arg)** // Обработчик кнопки

**{**

**Console.WriteLine ("OkMouse");**

**if (run)**

**{**

**run= false;**

**Suspend ( );**

**}**

**else**

**if (!run)**

**{**

**run= true;**

**Resume ( );**

**}**

**}**

//-------------------------

static void Main ( )

{

Application.Run (new **User ( )**);

}

}

/\*

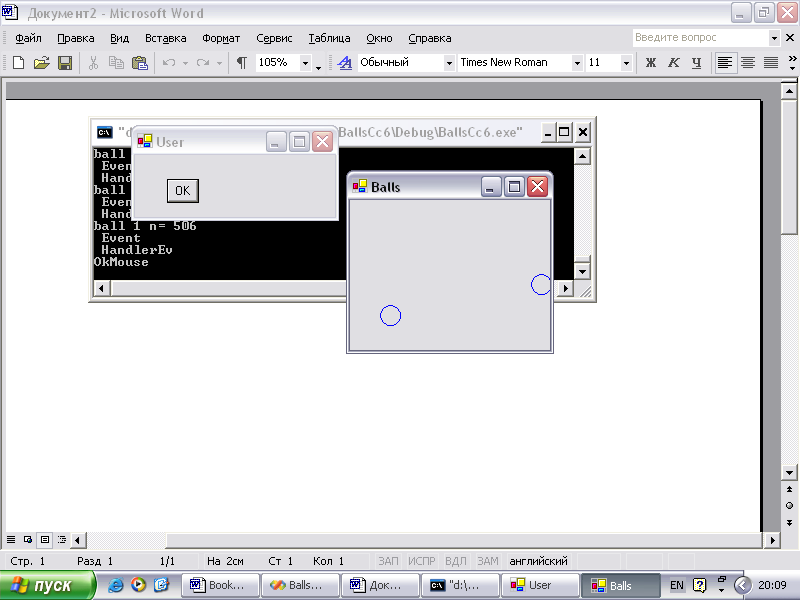
**Результат:**

*Появились прикладное окно с перемещающимися шарами и дочернее окно*

*с кнопкой. При нажатии на кнопку можно приостанавливать или возобновлять*

*движение шаров.*

\*/



**Рис. 5.3.5.1.** Две формы с консолью Balls–приложения 5 этапа

**Пример 5.3.5.2. Реализация пятого этапа разработки Balls–приложения на языке Java.**

package BallsCj5;

////////////////////

//C# File BallsCj5

import java.util.\*;

import java.awt.\*;

**import java.awt.event.\*;**

class Obs

{

public int num, n;

public Obs (int num, int n)

{this.num= num; this.n= n;}

int getNumber(){return num;}

int getN(){return n;}

}

class Ball extends Observable implements Runnable // Класс обычного объекта

{

int num;

Thread thread;

boolean run;

boolean waiting= false;

boolean life;

public int x, y;

public int w, h;

public int dx, dy;

public Ball (int Num , int X, int Y, int Dx, int Dy)// Конструктор

{

num= Num; life= false; run= false;

w= 100; h= 100;

x= X; y= Y; dx= Dx; dy= Dy;

Start ( );

}

public void Start ( ) // Стартовать

{

if (!run)

{

run= true;

life= true;

thread= new Thread (this);

thread.start ( );

}

}

//---------------

synchronized public void Suspend ( ) // Приостановить

{

if (run)

{

run= false;

waiting=true;

}

}

//---------------

synchronized public void Resume ( ) // Возобновить

{

if (!run)

{

run= true;

waiting=false;

notify();

}

}

//---------------

public void run ( ) // Выполнить поток

{

int n= 0;

while (life)

{

System.out.print("object "+num);

System.out.println(" n= "+n);

n++;

x += dx; y += dy;

if (x>w || x<0) dx= -dx;

if (y>h || y<0) dy= -dy;

// Уведомить

setChanged();

notifyObservers(new Obs(num, n));

try

{

Thread.sleep (100);

synchronized(this)

{

if(waiting)

wait();

}

}

catch(InterruptedException iE){}

}

System.out.print("object "+num);

System.out.println(" finished ");

}

}

//-------------------------------------

class Balls extends Frame implements Observer // Класс объектов

{

Ball [] pBall;

Color col;

public Balls ( )

{

this.setTitle("Balls");

this.setSize(400, 300);

col= Color.red;

pBall= new Ball [2];

for (int i= 0; i < 2; i++)

{

pBall[i]=new Ball (i, i\*3+10, 1\*3+15, i\*3+5, i\*3+5);

pBall[i].addObserver(this);

}

this.show();

Start ( );

}

//---------------

public void Start ( ) // Стартовать

{

for(int i=0; i < 2; i++)

{

pBall[i]. Start ( );

}

}

//---------------

public void Suspend ( ) // Возобновить

{

for (int i=0; i < 2; i++)

pBall[i].Suspend ( );

}

//---------------

public void Resume ( ) // Приостановить

{

for (int i= 0; i < 2; i++)

pBall[i].Resume ( );

}

public void update (Observable r, Object ob) // Обработчик события ev

{

Obs obs= (Obs) ob;

System.out.print("object num= "+ obs.num);

System.out.println(" n= "+ obs.n);

repaint();

}

public void paint (Graphics g)

{

super.paint (g);

g.setColor(col);

for (int i= 0; i < 2; i++)

{

g.drawOval(pBall[i].x, pBall[i].y, 20, 20);

pBall[i].w= this.getSize().width;

pBall[i].h= this.getSize().height;

}

}

}

**class User extends Balls** // Класс пользователя шаров

{

**private boolean run;**

**Frame pChildForm;**

**Button pBut;**

**public User ( )**

**{**

**run= true;**

//---------------

**this.setLayout(null);**

**pBut= new Button("OK");**

**pBut.setLocation(60, 60);**

**pBut.setSize (32, 24);**

**pBut.addActionListener(new ActionListener()**

**{**

**public void actionPerformed(ActionEvent aE)**

**{**

**System.out.println ("OkButton");**

**if (run)**

**{**

**run= false;**

**Suspend ( );**

**}**

**else**

**if (!run)**

**{**

**run= true;**

**Resume ( );**

**}**

**}**

**});**

**this.setSize(400, 300);**

//---------------

**pChildForm= new Frame ( );**

**pChildForm.setLocation(250, 10);**

**pChildForm.setSize(250, 250);**

**pChildForm.setTitle("User");**

**pChildForm.show ( );**

**pChildForm.add(pBut);**

**this.show();**

**}**

//-------------------------

public static void main ( )

{

**User u= new User();**

}

}

/\*

Результат:

Появились прикладное окно с перемещающимися шарами и дочернее окно

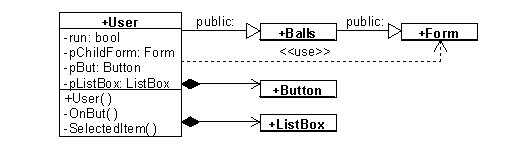
с кнопкой. При нажатии на кнопку можно приостанавливать или возобновлять

движение шаров.

\*/

**5.3.6. Этап 6. Добавление элемента списка**

Этот этап предпоследний. Добавим в дочернее окно управляющий элемент-список для выбора одного из трёх цветов шаров. Диаграмма классов и реализация последнего этапа представлены на рис. 4.3.6.1 и в примерах 4.3.6.1 и 4.3.6.2.



**Рис. 5.3.6.1.** Диаграмма классов Balls–приложения шестого этапа

**Пример 5.3.6.1.** **Реализация шестого этапа разработки Balls–приложения.**

////////////////////

// ***C#*** **File** *BallsCs6.cs*

// Файл *BallsCs6.cs* полностью включает файл *BallsCs5.cs*, дополнив класс

// *User* элементом списка и включив его в дочернее окно.Используя список,

*//* можно изменить цвет шаров.

// Изменения файла *BallsCs5.cs* выделены жирным шрифтом

**. . .**

//-------------------------------------

class Balls : Form // Класс объектов

{

Ball [] pBall;

**Color col;**

**. . .**

**public void SetColor (Color Col)** // Установить цвет

**{**

**col= Col;**

**Invalidate ( );**

**}**

**. . .**

}

//-------------------------------------

class User : Balls // Класс пользователя шаров

{

**. . .**

**Color col;**

**ListBox pListBox;**

public User ( ) // Конструктор

{

**. . .**

**pListBox= new ListBox ( );**

**pBut.MouseDown += new MouseEventHandler (OnBut);**

**pListBox.Items.Add ("red");**

**pListBox.Items.Add ("green");**

**pListBox.Items.Add ("blue");**

**pListBox.Location = new Point (104, 24);**

**pListBox.Name = "listBox";**

**pListBox.Size = new System.Drawing.Size (56, 17); pListBox.SelectedIndexChanged += new**

**System.EventHandler (SelectedItem); pChildForm.Controls.Add (pListBox);**

}

**. . .**

**void SelectedItem (object obj, EventArgs arg)**

**{**

**Console.WriteLine ("OkList");**

**if(pListBox.GetSelected (0))**

**col= Color.Red;**

**if(pListBox.GetSelected (1))**

**col= Color.Green;**

**if(pListBox.GetSelected (2))**

**col=Color.Blue;**

**SetColor (col);**

**}**

**. . .**

}

/\*

**Результат:**

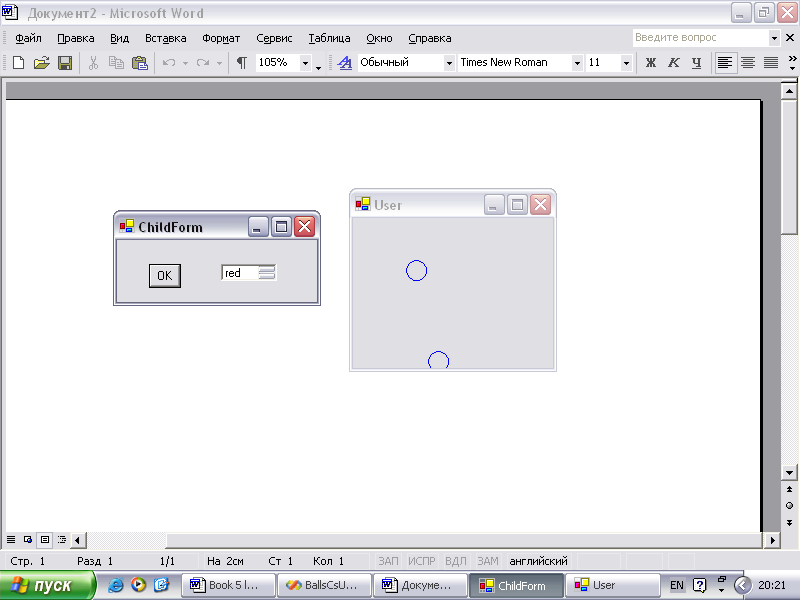
*Появились прикладное окно с перемещающимися шарами и дочернее окно*

*с кнопкой и со списком. При нажатии на кнопку можно приостанавливать*

*или возобновлять движение шаров. Используя список, можно изменить цвет*

*шаров.*

\*/



**Рис. 5.3.6.1.** Прикладное и дочернее окна Balls–приложения 6 этапа

**Пример 5.3.6.2.** **Реализация шестого этапа разработки Balls–приложения на языке Java.**

package BallsCj6;

////////////////////

//C# File BallsCj6

import java.util.\*;

import java.awt.\*;

import java.awt.event.\*;

//import java.awt.event.ActionListener;

class Obs

{

public int num, n;

public Obs (int num, int n)

{this.num= num; this.n= n;}

int getNumber(){return num;}

int getN(){return n;}

}

class Ball extends Observable implements Runnable // Класс обычного объекта

{

int num;

Thread thread;

boolean run;

boolean waiting= false;

boolean life;

public int x, y;

public int w, h;

public int dx, dy;

public Ball (int Num , int X, int Y, int Dx, int Dy)// Конструктор

{

num= Num; life= false; run= false;

w= 100; h= 100;

x= X; y= Y; dx= Dx; dy= Dy;

Start ( );

}

public void Start ( ) // Стартовать

{

if (!run)

{

run= true;

life= true;

thread= new Thread (this);

thread.start ( );

}

}

//---------------

synchronized public void Suspend ( ) // Приостановить

{

if (run)

{

run= false;

waiting=true;

}

}

//---------------

synchronized public void Resume ( ) // Возобновить

{

if (!run)

{

run= true;

waiting=false;

notify();

}

}

//---------------

public void run ( ) // Выполнить поток

{

int n= 0;

while (life)

{

System.out.print("object "+num);

System.out.println(" n= "+n);

n++;

x += dx; y += dy;

if (x>w || x<0) dx= -dx;

if (y>h || y<0) dy= -dy;

// Уведомить

setChanged();

notifyObservers(new Obs(num, n));

try

{

Thread.sleep (100);

synchronized(this)

{

if(waiting)

wait();

}

}

catch(InterruptedException iE){}

}

System.out.print("object "+num);

System.out.println(" finished ");

}

}

//-------------------------------------

class Balls extends Frame implements Observer // Класс объектов

{

Ball [] pBall;

Color col;

public Balls ( )

{

this.setTitle("Balls");

this.setSize(400, 300);

col= Color.red;

pBall= new Ball [2];

for (int i= 0; i < 2; i++)

{

pBall[i]=new Ball (i, i\*3+10, 1\*3+15, i\*3+5, i\*3+5);

pBall[i].addObserver(this);

}

this.show();

Start ( );

}

//---------------

public void SetColor (Color Col) // Установить цвет

{

col= Col;

repaint();

}

//---------------

public void Start ( ) // Стартовать

{

for(int i=0; i < 2; i++)

{

pBall[i]. Start ( );

}

}

//---------------

public void Suspend ( ) // Возобновить

{

for (int i=0; i < 2; i++)

pBall[i].Suspend ( );

}

//---------------

public void Resume ( ) // Приостановить

{

for (int i= 0; i < 2; i++)

pBall[i].Resume ( );

}

public void update (Observable r, Object ob) // Обработчик события ev

{

Obs obs= (Obs) ob;

System.out.print("object num= "+ obs.num);

System.out.println(" n= "+ obs.n);

repaint();

}

public void paint (Graphics g)

{

super.paint (g);

g.setColor(col);

for (int i= 0; i < 2; i++)

{

g.drawOval(pBall[i].x, pBall[i].y, 20, 20);

pBall[i].w= this.getSize().width;

pBall[i].h= this.getSize().height;

}

}

}

class User extends Balls implements **ActionListener,** // Класс пользователя шаров

**ItemListener**

{

private boolean run;

Frame pChildForm;

Button pBut;

**private Choice c;**// Ссылка на объект-список

public User ( )

{

run= true;

pBut= new Button("OK");

pBut.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent aE)

{

System.out.println ("OkButton");

if (run)

{

run= false;

Suspend ( );

}

else

if (!run)

{

run= true;

Resume ( );

}

}

});

this.setSize(400, 300);

//---------------

pChildForm= new Frame ( );

pChildForm.setLocation(250, 10);

pChildForm.setSize(250, 250);

pChildForm.setTitle("User");

pChildForm.show ( );

pChildForm.setLayout (new FlowLayout ( ));// Установить расстановку

pChildForm.add(pBut);

**c= new Choice ( );** // Создать объект списка

**c.addItem ("blue");** // Добавить элемент "blue"

**c.addItem ("green");** //Добавить элемент "green"

**c.addItem ("red");** //Добавить элемент "red"

**pBut.addActionListener (this);**

**pChildForm.add(c, new Point (100,20));** //Подсоединить список к окну

**pChildForm.setVisible (true);** // Показать кнопку и список в окне

}

**public void itemStateChanged (ItemEvent iE) { }**

**public void actionPerformed(ActionEvent aE)**

**{**

**switch (c.getSelectedIndex ( ))** // Получить индекс списка

{

**case 0: col= Color.blue;**

**break;**

**case 1: col= Color.green;**

**break;**

**case 2: col= Color.red;**

**break;**

**}**

**repaint ( );** // Перерисовать

}

//-------------------------

public static void main ( )

{

User u= new User();

}

}

/\*

Результат:

Появились прикладное окно с перемещающимися шарами и дочернее окно

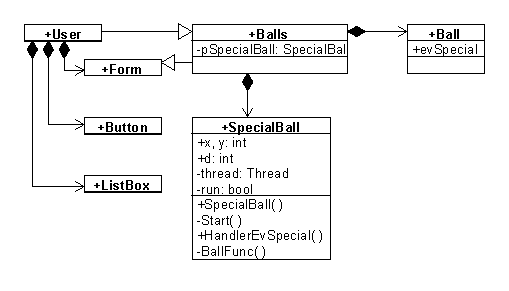
с кнопкой и элементом списка. При нажатии на кнопку можно приостанавливать или возобновлять

движение шаров. Выделив в списке цвет, можно изменить цвет шаров.

\*/

**5.3.7. Этап 7. Разработка специального объекта**

В последнем этапе разработаем класс SpecialBall специального объекта, добавим его объект в класс, содержащий массив обычных объектов. В соответствии с заданием этот специальный объект должен быть представлен в виде желтой окружности в левом верхнем углу прикладного окна, и эта оеружность вспыхивает (увеличивается ширина круга) при отталкивании шаров от стенок прикладного окна. Создадим специальный объект в теле конструктора класса Balls, подписав событие или уведомление каждого обычного объекта с обработчиком этого специального объекта. Добавим в дочернее окно управляющий элемент-список для выбора одного из трёх цветов шаров. Диаграмма классов и реализация последнего этапа представлены на рис. 5.3.7.1 и в примерах 5.3.7.1 и 5.3.7.2.



**Рис. 5.3.7.1.** Диаграмма классов Balls–приложения седьмого этапа

**Пример 5.3.7.1.** **Реализация седьмого этапа разработки Balls–приложения на языке C#.**

////////////////////

// ***C#*** **File** *BallsCs7.cs*

// Файл *BallsCs7.cs* полностью включает файл *BallsCs6.cs*, дополнив его новым

// классом SpecialBall специального объекта. Модифицирован класс Ball – в нём

*//* объявлено новое событие evSpecial. В классе Balls создан специальный объект,

*//* обработчик HandlerEvSpecial которого подписан на события всех обычных

// объектов. Изменения файла *BallsCs6.cs* выделены жирным шрифтом

// Изменения файла *BallsCs6.cs* выделены жирным шрифтом

**. . .**

class Ball // Класс обычного объекта

{

public event delEv ev; // Объявление события

**public event delEv evSpecial;** // **Объявление специального события**

**. . .**

void BallFunc ( ) // Выполнить поток

{

while (life) // Пока существует, выполнить

{

x += dx; y += dy;

if (x>w || x<0)

{

dx= -dx;

**if (evSpecial != null)**

**{**

**evSpecial ( );** // Генерировать событие

**}**

}

if (y>h || y<0)

{

dy= -dy;

**if (evSpecial != null)**

**{**

**evSpecial ( );** // Генерировать событие

**}**

}

if (ev != null)

{

ev ( ); // Генерировать событие

}

Thread.Sleep (100);

}

}

}

//-------------------------------------

**class SpecialBall** // Класс специального объекта

**{**

**public int x, y;** // Координаты шара

**public int d=1;** // Ширина пера

**Thread thread;** // Поток

**bool run;** // Признак выполнения потока

**public SpecialBall (int X, int Y)** // Конструктор

**{**

**run= false;**

**x= X; y= Y;**

**}**

**void Start ( )** // Стартовать

**{**

**if (!run)** // Если поток не выполнялся, то

**{**

**run= true;** // пусть выполняется и

**d= 5;** // утолщить кольцо

**thread= new Thread (new ThreadStart (BallFunc));**

**thread.Start ( );** // Стартовать поток

**}**

**}**

**public void HandlerEvSpecial ( )** // Обработать событие

**{**

**Start ( );**

**}**

**void BallFunc ( )** // Выполнить поток

**{**

**Thread.Sleep (2000);**

**d= 1;**

**run= false;**

**}**

**}**

//-------------------------------------

class Balls : Form // Класс потоковых объектов

{

public Ball [] pBall; // Массив потоковых объектов

**public SpecialBall pSpecialBall;**

bool run; // Признак выполнения потоковых объектов

Color col; // Цвет шаров

public Balls ( ) // Конструктор

{

run= true;

col= System.Drawing.Color.FromArgb (0, 0, 255);

**pSpecialBall= new SpecialBall (30, 30);**

pBall= new Ball [2];

for (int i= 0; i < 2; i++)

{

pBall[i]=new Ball (i\*3+10, 1\*3+15, i\*3+5, i\*3+5);

pBall[i].ev += new delEv (HandlerEv );

**pBall[i].evSpecial += new delEv (**

**pSpecialBall.HandlerEvSpecial);**

}

Text= "Balls"; // Заголовок прикладного окна

Start ( ); // Стартовать обычные объекты

}

**. . .**

//---------------

protected override void OnPaint (PaintEventArgs arg) // Перерисовать

{

for (int i= 0; i < 2; i++)

{

arg .Graphics.DrawEllipse (new Pen (col),

pBall[i].x, pBall[i].y, 20, 20);

pBall[i].w= Width;

pBall[i].h= Height;

}

**arg .Graphics.DrawEllipse (new Pen (Color.Yellow, pSpecialBall.d),**

**pSpecialBall.x, pSpecialBall.y, 20, 20);**

}

**. . .**

}

**. . .**

/\*

**Результат:**

*Появились прикладное окно с перемещающимися шарами и дочернее окно*

*с кнопкой и со списком. При нажатии на кнопку можно приостанавливать*

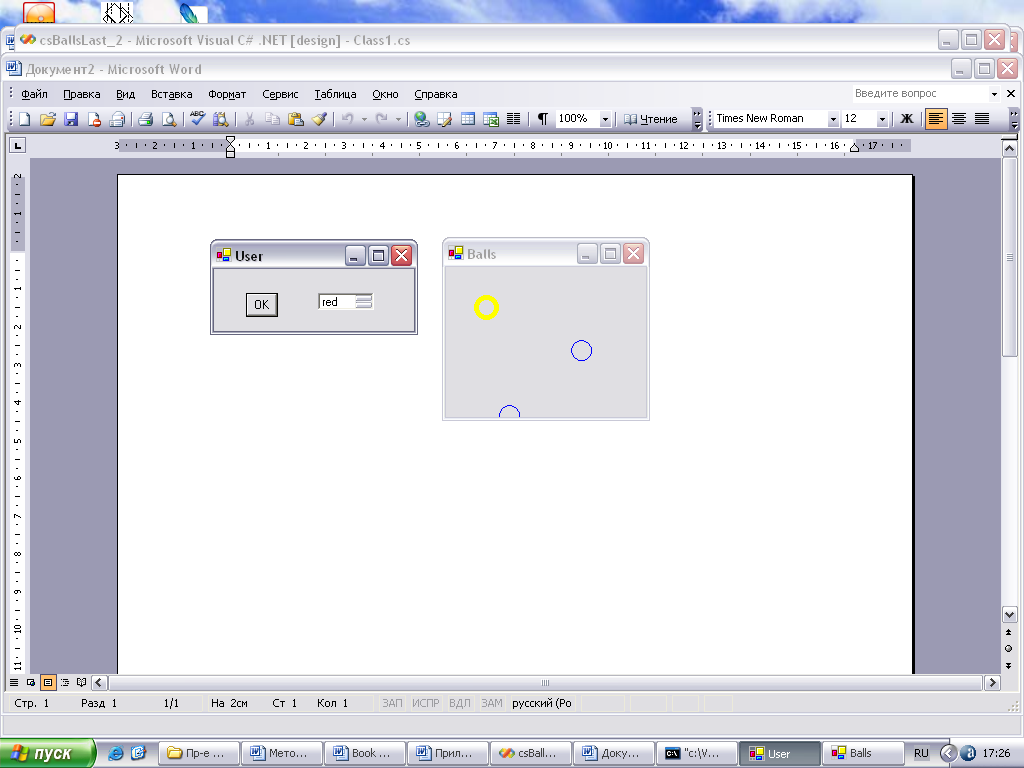
*или возобновлять движение шаров. Используя список, можно изменить цвет*

*шаров. В левом верхнем углу прикладного окна находится круг желтого цвета,*

*толщина которого утолщается при отскакивании кругов обычных объектов от*

*краёв прикладного окна.*

\*/



**Рис. 5.3.7.1.** Прикладное и дочернее окна Balls–приложения 7 этапа

**Пример 5.3.7.2. Реализация седьмого этапа разработки Balls–приложения на языке Java.**

package BallsCj7;

////////////////////

//C# File BallsCj7

import java.util.\*;

import java.awt.\*;

import java.awt.event.\*;

class Obs

{

private int num, n;

**private boolean special;** //

**public Obs (int num, int n, boolean special)** //

**{this.num= num; this.n= n;this.special= special;}**

int getNumber(){return num;}

int getN(){return n;}

**boolean getSpecial(){return special;}** //

}

class Ball extends Observable implements Runnable // Класс обычного объекта

{

int num;

Thread thread;

boolean run;

boolean waiting= false;

boolean life;

public int x, y;

public int w, h;

public int dx, dy;

public Ball (int Num , int X, int Y, int Dx, int Dy)// Конструктор

{

num= Num; life= false; run= false;

w= 100; h= 100;

x= X; y= Y; dx= Dx; dy= Dy;

Start ( );

}

public void Start ( ) // Стартовать

{

if (!run)

{

run= true;

life= true;

thread= new Thread (this);

thread.start ( );

}

}

//---------------

synchronized public void Suspend ( ) // Приостановить

{

if (run)

{

run= false;

waiting=true;

}

}

//---------------

synchronized public void Resume ( ) // Возобновить

{

if (!run)

{

run= true;

waiting=false;

notify();

}

}

//---------------

public void run ( ) // Выполнить поток

{

int n= 0;

while (life)

{

System.out.print("object "+num);

System.out.println(" n= "+n);

n++;

x += dx; y += dy;

if (x>w || x<0)

**{**

**setChanged();** //

**notifyObservers(new Obs(num, n, true));** //

dx= -dx;

**}**

if (y>h || y<0)

**{**

**setChanged();** //

**notifyObservers(new Obs(num, n, true));** //

dy= -dy;

**}**

// Уведомить

setChanged();

notifyObservers(new Obs(num, n, false)); //

try

{

Thread.sleep (100);

synchronized(this)

{

if(waiting)

wait();

}

}

catch(InterruptedException iE){}

}

System.out.print("object "+num);

System.out.println(" finished ");

}

}

**class SpecialBall implements Runnable,** // Класс специального объекта//

**Observer**

{

**public int x, y;** // Координаты шара

**Thread thread;** // Поток

**boolean yes;** // Признак выполнения потока //

**public SpecialBall (int X, int Y)** // Конструктор

**{**

**yes= false;**

**x= X; y= Y;**

**}**

**void Start ( )** // Стартовать

**{**

**if (!yes)** // Если поток не выполнялся, то

**{**

**yes= true;** // пусть выполняется

**thread= new Thread (this);**

**thread.start ( );** // Стартовать поток

}

}

**public void update(Observable ob, Object obj)** //

**{**

**Obs obs=(Obs) obj;**

**if(obs.getSpecial())**

**Start ( );**

**}**

**public void run ( )** // Выполнить поток//

**{**

**try**

**{**

**Thread.sleep (2000);**

**}**

**catch(InterruptedException iE){}**

**yes= false;**

**}**

}

//-------------------------------------

class Balls extends Frame implements Observer // Класс объектов

{

Ball [] pBall;

Color col;

**public SpecialBall pSpecialBall;** //

public Balls ( )

{

this.setTitle("Balls");

this.setSize(400, 300);

col= Color.red;

**pSpecialBall= new SpecialBall (30, 30);** //

pBall= new Ball [2];

for (int i= 0; i < 2; i++)

{

pBall[i]=new Ball (i, i\*3+10, 1\*3+15, i\*3+5, i\*3+5);

pBall[i].addObserver(this);

**pBall[i].addObserver(pSpecialBall);**

}

this.show();

Start ( );

}

//---------------

public void SetColor (Color Col) // Установить цвет

{

col= Col;

repaint();

}

//---------------

public void Start ( ) // Стартовать

{

for(int i=0; i < 2; i++)

{

pBall[i]. Start ( );

}

}

//---------------

public void Suspend ( ) // Возобновить

{

for (int i=0; i < 2; i++)

pBall[i].Suspend ( );

}

//---------------

public void Resume ( ) // Приостановить

{

for (int i= 0; i < 2; i++)

pBall[i].Resume ( );

}

public void update (Observable r, Object ob) // Обработчик события ev

{

Obs obs= (Obs) ob;

**if(!obs.getSpecial())**

**{**

System.out.print("object num= "+ obs.getNumber());

System.out.println(" n= "+ obs.getN());

repaint();

**}**

}

public void paint (Graphics g)

{

super.paint (g);

g.setColor(col);

for (int i= 0; i < 2; i++)

{

g.drawOval(pBall[i].x, pBall[i].y, 20, 20);

pBall[i].w= this.getSize().width;

pBall[i].h= this.getSize().height;

}

**if(pSpecialBall.yes)**

**{**

**g.setColor(Color.yellow);**

**g.drawOval(pSpecialBall.x, pSpecialBall.y, 20, 20);**

**}**

}

}

class User extends Balls implements ActionListener, // Класс пользователя шаров

ItemListener

{

private boolean run;

Frame pChildForm;

Button pBut;

private Choice c;// Ссылка на объект-список

public User ( )

{

run= true;

pBut= new Button("OK");

pBut.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent aE)

{

System.out.println ("OkButton");

if (run)

{

run= false;

Suspend ( );

}

else

if (!run)

{

run= true;

Resume ( );

}

}

});

this.setSize(400, 300);

//---------------

pChildForm= new Frame ( );

pChildForm.setLocation(250, 10);

pChildForm.setSize(250, 250);

pChildForm.setTitle("User");

pChildForm.show ( );

pChildForm.setLayout (new FlowLayout ( ));// Установить расстановку

pChildForm.add(pBut);

c= new Choice ( ); // Создать объект списка

c.addItem ("blue");// Добавить элемент "blue"

c.addItem ("green");//Добавить элемент "green"

c.addItem ("red");//Добавить элемент "red"

pBut.addActionListener(this);

pChildForm.add(c, new Point (100,20));//Подсоединить список к окну

pChildForm.setVisible (true);// Показать кнопку и список в окне

}

public void itemStateChanged (ItemEvent iE) { }

public void actionPerformed(ActionEvent aE)

{

switch (c.getSelectedIndex ( ))// Получить индекс списка

{

case 0: col= Color.blue;//

break;

case 1: col= Color.green;//

break;

case 2: col= Color.red;//

break;

}

repaint ( );// Перерисовать

}

//-------------------------

public static void main ( )

{

User u= new User();

}

}

/\*

Результат:

Появились прикладное окно с перемещающимися шарами и дочернее окно

с кнопкой и элементом списка. При нажатии на кнопку можно приостанавливать или возобновлять

движение шаров. Выделив в списке цвет, можно изменить цвет шаров.

**При отталкивании каждого шара от границ окна появляется в верхнем левом углу окна желтый шар и исчезает через 2000 мс.**

\*/

**6. Описание проблем, возникших при разработке программной системы.**

Поэтапная разработка программы позволила быстро разработать её и отладить. Проблем не было.

**7. Список используемой литературы**

1. Байдачный С.С. .NET Framework. Секреты создания Windows-приложений. – М.: СОЛОН-Пресс, 2004. – 496 с.: ил.

2. Гербердт Шилдт. C#: учебный курс. – СПб.: Питер; К.: Издательская группа BHV, 2003. – 512 c.: ил.

3. Джесс Либерти. Программирование на C#: 2-е издание. – СПб.: Символ-Плюс, 2003. – 688 c.: ил.

4. Медведев В.И. Программирование на С++, C++.NET и С#: – Казань: *Мастер* *Лайн*, 2005. – 270 c.: ил. – (Серия «Современная прикладная математика и информатика»).

5. Медведев В.И. Особенности объектно-ориентированного программирования на C++/CLI, C# и Java. – Казань: РИЦ «Школа», 2008. – 360 c.: ил. – (Серия «Современная прикладная математика и информатика»).

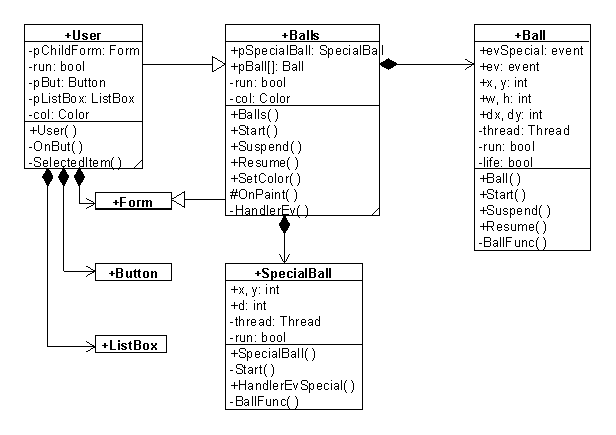
6. Орлов А.Технология разработки программного обеспечения. Учебник. – СПб.: Питер, 2002. – 464 с.

7. Петцольд Ч. Программирование для Microsoft Windows на C#. В 2-х томах / Пер. с англ. – М.: Издательско-торговый дом “Русская Редакция”, 2002. – 576 + 624 с.: ил.

8. П. Наутон, Г. Шилдт. Java 2. Наиболее полное руководство в подлиннике.– СПб.: БХВ-Петербург, 2000. – 1072 с.: ил.

9. Рамбо Дж., Якобсон А., Буч Г. UML: специальный справочник. – СПб.: Питер, 2002. – 656 c.

**8. Приложение 1. Диаграмма классов разработанной программы**



**9. Приложение 2. Текст программы на языке C#**

////////////////////

// C# File BallsCs.cpp

using System;

using System.Drawing;

using System.Windows.Forms;

using System.Threading;

delegate void delEv ( ); // Делегат события

class Ball // Класс обычного объекта

{

public event delEv ev; // Объявление события

public event delEv evSpecial; // Объявление специального события

public int x, y; // Координаты шара

public int w, h; // Ширина и высота области клиента

public int dx, dy; // Приращения координат шара

Thread thread; // Поток

bool run; // Признак выполнения потока

bool life; // Признак существования потока

public Ball (int X, int Y, int Dx, int Dy) // Конструктор

{

life= false; run= false; w= 100; h= 100;

x= X; y= Y; dx= Dx; dy= Dy;

}

//---------------

public void Start ( ) // Стартовать

{

if (!run) // Если поток не выполнялся, то

{

run= true; // пусть выполняется и

life= true; // пусть живёт.

thread= new Thread (new ThreadStart (BallFunc));

thread.Start ( );// Стартовать поток

}

}

//---------------

public void Suspend ( ) // Приостановить

{

if (run) // Если поток выполняется, то

{

run= false; // прекратить выполнение.

thread.Suspend ( ); // Приостановить

}

}

//---------------

public void Resume ( ) // Возобновить

{

if (!run) // Если поток приостановлен, то

{

run= true; // пусть выполняется

thread.Resume ( ); // Возобновить

}

}

//---------------

public void Finish ( ) // Завершить

{

life= false;

thread.Abort ( );

thread.Join ( );

}

//---------------

void BallFunc ( ) // Выполнить поток

{

while (life) // Пока существует, выполнить

{

x += dx; y += dy;

if (x>w || x<0)

{

dx= -dx;

if (evSpecial != null)

{

evSpecial ( ); // Генерировать событие

}

}

if (y>h || y<0)

{

dy= -dy;

if (evSpecial != null)

{

evSpecial ( ); // Генерировать событие

}

}

if (ev != null)

{

ev ( ); // Генерировать событие

}

Thread.Sleep (100);

}

}

}

class SpecialBall // Класс специального объекта

{

public int x, y; // Координаты шара

public int d=1; // Ширина пера

Thread thread; // Поток

bool run; // Признак выполнения потока

public SpecialBall (int X, int Y) // Конструктор

{

run= false;

x= X; y= Y;

}

//---------------

void Start ( ) // Стартовать

{

if (!run) // Если поток не выполнялся, то

{

run= true; // пусть выполняется и

d= 5; // утолщить кольцо

thread= new Thread (new ThreadStart (BallFunc));

thread.Start ( );// Стартовать поток

}

}

//---------------

public void HandlerEvSpecial ( ) // Обработать событие

{

Start ( );

}

//---------------

void BallFunc ( ) // Выполнить поток

{

Thread.Sleep (2000);

d= 1;

run= false;

}

}

//-------------------------------------

class Balls : Form // Класс объектов

{

public Ball [] pBall; // Массив обычных объектов

public SpecialBall pSpecialBall;

bool run; // Признак выполнения обычных объектов

Color col; // Цвет шаров

public Balls ( ) // Конструктор

{

run= true;

col= System.Drawing.Color.FromArgb (0, 0, 255);

pSpecialBall= new SpecialBall (30, 30);

pBall= new Ball [2];

for (int i= 0; i < 2; i++)

{

pBall[i]=new Ball (i\*3+10, 1\*3+15, i\*3+5, i\*3+5);

pBall[i].ev += new delEv (HandlerEv );

pBall[i].evSpecial += new delEv

(pSpecialBall.HandlerEvSpecial);

}

Text= "Balls"; // Заголовок прикладного окна

Start ( ); // Стартовать обычные объекты

}

//---------------

public void Start ( ) // Стартовать

{

for (int i=0; i < 2; i++)

pBall[i]. Start ( );

}

//---------------

public void Suspend ( ) // Приостановить

{

if (run) // Если обычные объекты выполнялись, то

{

run= false; // прекратить выполнение.

for (int i=0; i < 2; i++)

pBall[i].Suspend ( ); // Приостановить

}

}

//---------------

public void Resume ( ) // Возобновить

{

if (!run) // Если обычные объекты приостановлены, то

{

run= true; // продолжить выполнение.

for(int i= 0; i < 2; i++)

pBall[i].Resume ( ); // Возобновить

}

}

//---------------

public void SetColor (Color Col) // Установить цвет

{

col= Col;

Invalidate ( );

}

//---------------

protected override void OnPaint (PaintEventArgs arg) // Перерисовать

{

for (int i= 0; i < 2; i++)

{

arg .Graphics.DrawEllipse (new Pen (col),

pBall[i].x, pBall[i].y, 20, 20);

pBall[i].w= Width;

pBall[i].h= Height;

}

arg .Graphics.DrawEllipse (new Pen (Color.Yellow, pSpecialBall. d),

pSpecialBall.x, pSpecialBall.y, 20, 20);

}

//---------------

void HandlerEv ( ) // Обработать событие

{

Invalidate ( );

}

}

//----------

class User : Balls // Класс пользователя шаров

{

private bool run; // Признак перемещения шаров

Form pChildForm; // Дочерняя форма (окно)

Color col; // Цвет шаров

Button pBut; // Кнопка

ListBox pListBox; // Список

public User ( ) // Конструктор

{

run= true;

//---------------

pBut= new Button(); // Создать кнопку

pBut.Location = new Point (32, 24); // Разместить кнопку

pBut.Name = "pBut"; // Дать имя кнопке

// Задать размер кнопки в области клиента дочерней формы

pBut.Size = new System.Drawing.Size (32, 24);

pBut.Text = "OK"; // Поместить текст в кнопку

// Подписать обработчик EventHandler кнопки на событие Click

pBut.Click += new EventHandler (OnBut);

//---------------

pListBox= new ListBox ( ); // Создать список

pListBox.Items.Add ("red"); // Добавить red

pListBox.Items.Add ("green"); // Добавить green

pListBox.Items.Add ("blue"); // Добавить blue

pListBox.Location = new Point (104, 24); // Разместить список

pListBox.Name = "listBox"; // Дать имя списку

// Задать размер списка в области клиента дочерней формы

pListBox.Size = new System.Drawing.Size (56, 17);

// Подписать обработчик списка на событие

// SelectedIndexChanged

pListBox.SelectedIndexChanged += new System.EventHandler

(SelectedItem);

// Задать размер области клиента прикладного окна

ClientSize = new System.Drawing.Size (200, 150);

//---------------

pChildForm= new Form ( ); // Создать дочернее окно (форму)

pChildForm.Location= new Point (250, 10);// и разместить его.

pChildForm.Size= new Size (250, 250); // Определить размер

pChildForm.Text= "User"; // Задать заголовок дочернего окна

pChildForm.Show ( ); // Показать дочернее окно

pChildForm.Controls.Add (pBut); // Добавить кнопку

pChildForm.Controls.Add (pListBox); // и список.

pChildForm.ClientSize = new System.Drawing.Size (200, 62);

}

//---------------

void OnBut (object obj, EventArgs arg) // Обработать кнопку

{

if (run) // Если шары перемещаются, то

{

run= false; // пусть перестанут перемещаться.

Suspend ( ); // Приостановить

}

else // Иначе,

if(!run) // если шары неподвижны, то

{

run= true; // пусть перемещаются.

Resume ( ); // Возобновить

}

}

//---------------

void SelectedItem (object obj, EventArgs arg) // Выбрать из списка

{

if(pListBox.GetSelected (0))

col= Color.Red;

if(pListBox.GetSelected (1))

col= Color.Green;

if(pListBox.GetSelected (2))

col=Color.Blue;

SetColor (col);

}

//-------------------------

static void Main()

{

User user= new User( );

Application.Run(user); // Выполнить приложение

for (int i= 0; i < 2; i++) // Завершить и удалить потоки

user.pBall[i].Finish ( );

}

}

**10. Приложение 3. Текст программы на языке Java**

package BallsCj7;

////////////////////

//C# File BallsCj7

import java.util.\*;

import java.awt.\*;

import java.awt.event.\*;

class Obs

{

private int num, n;

private boolean special;//

public Obs (int num, int n, boolean special)//

{this.num= num; this.n= n;this.special= special;}//

int getNumber(){return num;}

int getN(){return n;}

boolean getSpecial(){return special;}//

}

class Ball extends Observable implements Runnable // Класс обычного объекта

{

int num;

Thread thread;

boolean run;

boolean waiting= false;

boolean life;

public int x, y;

public int w, h;

public int dx, dy;

public Ball (int Num , int X, int Y, int Dx, int Dy)// Конструктор

{

num= Num; life= false; run= false;

w= 100; h= 100;

x= X; y= Y; dx= Dx; dy= Dy;

Start ( );

}

public void Start ( ) // Стартовать

{

if (!run)

{

run= true;

life= true;

thread= new Thread (this);

thread.start ( );

}

}

//---------------

synchronized public void Suspend ( ) // Приостановить

{

if (run)

{

run= false;

waiting=true;

}

}

//---------------

synchronized public void Resume ( ) // Возобновить

{

if (!run)

{

run= true;

waiting=false;

notify();

}

}

//---------------

public void run ( ) // Выполнить поток

{

int n= 0;

while (life)

{

System.out.print("object "+num);

System.out.println(" n= "+n);

n++;

x += dx; y += dy;

if (x>w || x<0)

{

setChanged();//

notifyObservers(new Obs(num, n, true));//

dx= -dx;

}

if (y>h || y<0)

{

setChanged();//

notifyObservers(new Obs(num, n, true));//

dy= -dy;

}

// Уведомить

setChanged();

notifyObservers(new Obs(num, n, false));//

try

{

Thread.sleep (100);

synchronized(this)

{

if(waiting)

wait();

}

}

catch(InterruptedException iE){}

}

System.out.print("object "+num);

System.out.println(" finished ");

}

}

class SpecialBall implements Runnable, // Класс специального объекта//

Observer

{

public int x, y; // Координаты шара

Thread thread; // Поток

boolean yes; // Признак выполнения потока //

public SpecialBall (int X, int Y) // Конструктор

{

yes= false;

x= X; y= Y;

}

void Start ( ) // Стартовать

{

if (!yes) // Если поток не выполнялся, то

{

yes= true; // пусть выполняется

thread= new Thread (this);//new ThreadStart (BallFunc));

thread.start ( ); // Стартовать поток

}

}

public void update(Observable ob, Object obj)//

{

Obs obs=(Obs) obj;

if(obs.getSpecial())

Start ( );

}

public void run ( ) // Выполнить поток//

{

try

{

Thread.sleep (2000);

}

catch(InterruptedException iE){}

yes= false;

}

}

//-------------------------------------

class Balls extends Frame implements Observer // Класс объектов

{

Ball [] pBall;

Color col;

public SpecialBall pSpecialBall;//

public Balls ( )

{

this.setTitle("Balls");

this.setSize(400, 300);

col= Color.red;

pSpecialBall= new SpecialBall (30, 30);//

pBall= new Ball [2];

for (int i= 0; i < 2; i++)

{

pBall[i]=new Ball (i, i\*3+10, 1\*3+15, i\*3+5, i\*3+5);

pBall[i].addObserver(this);

pBall[i].addObserver(pSpecialBall);

}

this.show();

Start ( );

}

//---------------

public void SetColor (Color Col) // Установить цвет

{

col= Col;

repaint();

}

//---------------

public void Start ( ) // Стартовать

{

for(int i=0; i < 2; i++)

{

pBall[i]. Start ( );

}

}

//---------------

public void Suspend ( ) // Возобновить

{

for (int i=0; i < 2; i++)

pBall[i].Suspend ( );

}

//---------------

public void Resume ( ) // Приостановить

{

for (int i= 0; i < 2; i++)

pBall[i].Resume ( );

}

public void update (Observable r, Object ob) // Обработчик события ev

{

Obs obs= (Obs) ob;

if(!obs.getSpecial())

{

System.out.print("object num= "+ obs.getNumber());

System.out.println(" n= "+ obs.getN());

repaint();

}

}

public void paint (Graphics g)

{

super.paint (g);

g.setColor(col);

for (int i= 0; i < 2; i++)

{

g.drawOval(pBall[i].x, pBall[i].y, 20, 20);

pBall[i].w= this.getSize().width;

pBall[i].h= this.getSize().height;

}

if(pSpecialBall.yes)

{

g.setColor(Color.yellow);

g.drawOval(pSpecialBall.x, pSpecialBall.y, 20, 20);

}

}

}

class User extends Balls implements ActionListener, // Класс пользователя шаров

ItemListener

{

private boolean run;

Frame pChildForm;

Button pBut;

private Choice c;// Ссылка на объект-список

public User ( )

{

run= true;

pBut= new Button("OK");

pBut.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent aE)

{

System.out.println ("OkButton");

if (run)

{

run= false;

Suspend ( );

}

else

if (!run)

{

run= true;

Resume ( );

}

}

});

this.setSize(400, 300);

//---------------

pChildForm= new Frame ( );

pChildForm.setLocation(250, 10);

pChildForm.setSize(250, 250);

pChildForm.setTitle("User");

pChildForm.show ( );

pChildForm.setLayout (new FlowLayout ( ));// Установить расстановку

pChildForm.add(pBut);

c= new Choice ( ); // Создать объект списка

c.addItem ("blue");// Добавить элемент "blue"

c.addItem ("green");//Добавить элемент "green"

c.addItem ("red");//Добавить элемент "red"

pBut.addActionListener(this);

pChildForm.add(c, new Point (100,20));//Подсоединить список к окну

pChildForm.setVisible (true);// Показать кнопку и список в окне

}

public void itemStateChanged (ItemEvent iE) { }

public void actionPerformed(ActionEvent aE)

{

switch (c.getSelectedIndex ( ))// Получить индекс списка

{

case 0: col= Color.blue;//

break;

case 1: col= Color.green;//

break;

case 2: col= Color.red;//

break;

}

repaint ( );// Перерисовать

}

//-------------------------

public static void main ( )

{

User u= new User();

}

}

/\*

Результат:

Появились прикладное окно с перемещающимися шарами и дочернее окно

с кнопкой и элементом списка. При нажатии на кнопку можно приостанавливать или возобновлять

движение шаров. Выделив в списке цвет, можно изменить цвет шаров.

При отталкивании каждого шара от границ окна появляется в верхнем левом углу окна желтый шар

и исчезает через 2000 мс.

\*/

**11. Приложение 3. Текст модифицированной программы на языке Java**

package BallsCj7;

////////////////////

//C# File BallsCj7

// Синхронизировать потоки с разделяемым ресурсом типа SpecialBall

import java.util.\*;

import java.awt.\*;

import java.awt.event.\*;

class Obs

{

private int num, n;

private boolean special;//

public Obs (int num, int n, boolean special)//

{this.num= num; this.n= n;this.special= special;}//

int getNumber(){return num;}

int getN(){return n;}

boolean getSpecial(){return special;}//

}

class Ball extends Observable implements Runnable // Класс обычного объекта

{

int num;

Thread thread;

boolean run;

boolean waiting= false;

boolean life;

public int x, y;

public int w, h;

public int dx, dy;

**SpecialBall sB;**

//public Ball (int Num , int X, int Y, int Dx, int Dy)// Конструктор

**public Ball (int Num , int X, int Y, int Dx, int Dy, SpecialBall SB)**

{

num= Num; life= false; run= false;

w= 100; h= 100;

x= X; y= Y; dx= Dx; dy= Dy;

**sB= SB;**

Start ( );

}

public void Start ( ) // Стартовать

{

if (!run)

{

run= true;

life= true;

thread= new Thread (this);

thread.start ( );

}

}

//---------------

synchronized public void Suspend ( ) // Приостановить

{

if (run)

{

run= false;

waiting=true;

}

}

//---------------

synchronized public void Resume ( ) // Возобновить

{

if (!run)

{

run= true;

waiting=false;

notify();

}

}

//---------------

public void run ( ) // Выполнить поток

{

int n= 0;

while (life)

{

//synchronized(sB)

{

System.out.print("object "+num);

System.out.println(" n= "+n);

n++;

x += dx; y += dy;

**synchronized(sB)**

**{**

if (x>w || x<0)

{

setChanged();//

notifyObservers(new Obs(num, n, true));//

dx= -dx;

}

if (y>h || y<0)

{

setChanged();//

notifyObservers(new Obs(num, n, true));//

dy= -dy;

}

**}**

// Уведомить

setChanged();

notifyObservers(new Obs(num, n, false));//

try

{

Thread.sleep (100);

//Thread.sleep (50);

synchronized(this)

{

if(waiting)

wait();

}

}

catch(InterruptedException iE){}

}

}

System.out.print("object "+num);

System.out.println(" finished ");

}

}

class SpecialBall implements Runnable, // Класс специального объекта//

Observer

{

public int x, y; // Координаты шара

Thread thread; // Поток

boolean yes; // Признак выполнения потока //

public SpecialBall (int X, int Y) // Конструктор

{

yes= false;

x= X; y= Y;

}

void Start ( ) // Стартовать

{

if (!yes) // Если поток не выполнялся, то

{

yes= true; // пусть выполняется

thread= new Thread (this);//new ThreadStart (BallFunc));

thread.start ( ); // Стартовать поток

}

}

public void update(Observable ob, Object obj)//

{

Obs obs=(Obs) obj;

if(obs.getSpecial())

Start ( );

}

public void run ( ) // Выполнить поток//

{

try

{

Thread.sleep (2000);

}

catch(InterruptedException iE){}

yes= false;

}

}

//-------------------------------------

class Balls extends Frame implements Observer // Класс объектов

{

Ball [] pBall;

Color col;

public SpecialBall pSpecialBall;//

public Balls ( )

{

this.setTitle("Balls");

this.setSize(400, 300);

col= Color.red;

pSpecialBall= new SpecialBall (30, 30);//

//pBall= new Ball [2];

pBall= new Ball [4];

//for (int i= 0; i < 2; i++)

**for (int i= 0; i < 4; i++)**

{

//pBall[i]=new Ball (i, i\*3+10, 1\*3+15, i\*3+5, i\*3+5);

**pBall[i]=new Ball (i, i\*3+10, 1\*3+15, i\*3+5, i\*3+5**

**, pSpecialBall);**

pBall[i].addObserver(this);

pBall[i].addObserver(pSpecialBall);

}

this.show();

Start ( );

}

//---------------

public void SetColor (Color Col) // Установить цвет

{

col= Col;

repaint();

}

//---------------

public void Start ( ) // Стартовать

{

for(int i=0; i < 2; i++)

{

pBall[i]. Start ( );

}

}

//---------------

public void Suspend ( ) // Возобновить

{

for (int i=0; i < 2; i++)

pBall[i].Suspend ( );

}

//---------------

public void Resume ( ) // Приостановить

{

for (int i= 0; i < 2; i++)

pBall[i].Resume ( );

}

public void update (Observable r, Object ob) // Обработчик события ev

{

Obs obs= (Obs) ob;

if(!obs.getSpecial())

{

System.out.print("object num= "+ obs.getNumber());

System.out.println(" n= "+ obs.getN());

repaint();

}

}

public void paint (Graphics g)

{

super.paint (g);

g.setColor(col);

//for (int i= 0; i < 2; i++)

**for (int i= 0; i < 4; i++)**

{

g.drawOval(pBall[i].x, pBall[i].y, 20, 20);

pBall[i].w= this.getSize().width;

pBall[i].h= this.getSize().height;

}

if(pSpecialBall.yes)

{

g.setColor(Color.yellow);

g.drawOval(pSpecialBall.x, pSpecialBall.y, 20, 20);

}

}

}

class User extends Balls implements ActionListener, // Класс пользователя шаров

ItemListener

{

private boolean run;

Frame pChildForm;

Button pBut;

private Choice c;// Ссылка на объект-список

public User ( )

{

run= true;

pBut= new Button("OK");

pBut.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent aE)

{

System.out.println ("OkButton");

if (run)

{

run= false;

Suspend ( );

}

else

if (!run)

{

run= true;

Resume ( );

}

}

});

this.setSize(400, 300);

//---------------

pChildForm= new Frame ( );

pChildForm.setLocation(250, 10);

pChildForm.setSize(250, 250);

pChildForm.setTitle("User");

pChildForm.show ( );

pChildForm.setLayout (new FlowLayout ( ));// Установить расстановку

pChildForm.add(pBut);

c= new Choice ( ); // Создать объект списка

c.addItem ("blue");// Добавить элемент "blue"

c.addItem ("green");//Добавить элемент "green"

c.addItem ("red");//Добавить элемент "red"

pBut.addActionListener(this);

pChildForm.add(c, new Point (100,20));//Подсоединить список к окну

pChildForm.setVisible (true);// Показать кнопку и список в окне

// Примененить внутренний анонимный класс

this.addWindowListener (new WindowAdapter ( )

{

public void windowClosing (WindowEvent wE)

{

System.exit(0);

}

});

}

public void itemStateChanged (ItemEvent iE) { }

public void actionPerformed(ActionEvent aE)

{

switch (c.getSelectedIndex ( ))// Получить индекс списка

{

case 0: col= Color.blue;//

break;

case 1: col= Color.green;//

break;

case 2: col= Color.red;//

break;

}

repaint ( );// Перерисовать

}

//-------------------------

public static void main ( )

{

User u= new User();

}

}

/\*

Результат:

Появились прикладное окно с перемещающимися шарами и дочернее окно

с кнопкой и элементом списка. При нажатии на кнопку можно приостанавливать или во-зобновлять

движение шаров. Выделив в списке цвет, можно изменить цвет шаров.

При отталкивании каждого шара от границ окна появляется в верхнем левом углу окна желтый шар

и исчезает через 2000 мс.

\*/

Применять оператор **synchronized** надлежит там, где ***он охватывает обращение только к разделяемому ресурсу***, иначе выполнение потоков будет существенно замедляться, поскольку они будут ждать выполнения иными потоками действий, не связанных с ресурсом.