Міністерство освіти і науки України

Київський національний університет будівництва і архітектури

**МЕТОДИЧНІ ВКАЗІВКИ**

**ДО ЛАБОРОТОРНИХ РОБІТ**

З КУРСУ

Програмування на мові JAVA

Київ – 2019

#

Оглавление

[Введение 4](#_Toc19732182)

[1. JAVA - объектно-ориентированный язык программирования 7](#_Toc19732183)

[1.1. Базовые типы данных 8](#_Toc19732184)

[1.2. Операции (operators) в языке Java 8](#_Toc19732185)

[1.2.1. Операции сравнения 9](#_Toc19732186)

[1.2.2. Операции инкремента, декремента 9](#_Toc19732187)

[1.3. Литералы (константы) 10](#_Toc19732188)

[1.4. Операторы 10](#_Toc19732189)

[1.5. Массивы в Java 10](#_Toc19732190)

[1.6. Комментарии 12](#_Toc19732191)

[1.7. Первая программа на языке Java 12](#_Toc19732192)

[1.8. Цель, требования и рекомендации к выполнению задания 13](#_Toc19732193)

[1.9. Задания 14](#_Toc19732194)

[2. Абстрактные классы и Интерфейсы 15](#_Toc19732195)

[2.1. Абстрактные классы 15](#_Toc19732196)

[2.2. Интерфейсы 16](#_Toc19732197)

[2.3. Цель, требования и рекомендации к выполнению задания 17](#_Toc19732198)

[2.4. Задания 17](#_Toc19732199)

[2.5. Пример выполнения задания. 19](#_Toc19732200)

[3. Знакомство с библиотекой Swing 20](#_Toc19732201)

[3.1. Основы оконной графики 20](#_Toc19732202)

[3.2. Модель событий в Swing 21](#_Toc19732203)

[3.3. Цель, требования и рекомендации к выполнению задания 22](#_Toc19732204)

[3.4. Задания 22](#_Toc19732205)

[3.5. Пример выполнения задания. 23](#_Toc19732206)

[4. Апплеты 25](#_Toc19732207)

[4.1. Проблема безопасности 26](#_Toc19732208)

[4.2. Создание апплетов 27](#_Toc19732209)

[4.3. Апплеты и приложения 28](#_Toc19732210)

[4.4. Цель, требования и рекомендации к выполнению задания 29](#_Toc19732211)

[4.5. Задания 30](#_Toc19732212)

[4.6. Пример выполнения задания. 30](#_Toc19732213)

[4.6.1. Запуск апплетов 30](#_Toc19732214)

[4.6.2. Пример выполнения задания 2 31](#_Toc19732215)

[5. Множественные нити выполнения (Multiple threads). 37](#_Toc19732216)

[5.1. Реализация многопоточности в Java 38](#_Toc19732217)

[5.1.1. Методы класса Thread 39](#_Toc19732218)

[5.1.2. Реализация интерфейса Runnable 39](#_Toc19732219)

[5.3. Завершение и останов нити 40](#_Toc19732220)

[5.4. Приоритеты нитей 41](#_Toc19732221)

[5.5. Средства синхронизации нитей в Java 41](#_Toc19732222)

[5.6. Применение многопоточности для анимации 42](#_Toc19732223)

[5.7. Цель, требования и рекомендации к выполнению задания 43](#_Toc19732224)

[5.8. Задания 44](#_Toc19732225)

[5.9. Пример выполнения задания. 46](#_Toc19732226)

# Введение

Практикум по программированию представляет собой учебно-методические материалы по курсу “Программирование на языке JAVA”, который ориентирован на использование при проведении практических занятий, лабораторных работ.

В практикуме приведены методы и рекомендации по решению типовых задач, необходимые сведения с средствах языка JAVA, примеры, формулировки заданий для лабораторных работ.

Предполагается, что типовые задачи сначала обсуждаются на практических занятиях: рассматривается постановка задачи, общая схема решения, особенности применяемых типов и языковых конструкций, особенности реализации при программировании на конкретных примерах. Затем выдается индивидуальное задание, которое необходимо выполнить с учетом требований и рекомендаций. Задание должно быть выполнено на компьютере по принятой на занятиях технологии, программа отлажена, результаты продемонстрированы преподавателю в удобной для анализа форме.

Язык Java — это объектно-ориентированный язык программирования, ведущий свою историю от известного языка C++. Но в отличие от последнего Java является языком интерпретируемым, программы, написанные на нем, способны работать в разных местах сети и не зависят от платформы, на которой выполняются написанные на нем приложения.

Изучая Java, вы будете приятно удивлены тем, что его синтаксис близок к синтаксису языка C++. Унаследовав самое лучшее от языка программирования C++, язык Java при этом избавился от некоторых недостатков C++, в результате чего на нем стало проще программировать. В этом языке нет, например, указателей, которые сложны в использовании и потенциально могут послужить причиной доступа программы к не принадлежащей ей области памяти. Нет множественного наследования и шаблонов, хотя функциональные возможности языка Java от этого не пострадали. Если вы умеете программировать на C++, для вас не составит особого труда изучить язык Java.

Огромное преимущество Java заключается в том, что на этом языке можно создавать приложения, способные работать на различных платформах. К сети Internet подключены компьютеры самых разных типов - Pentium PC, Macintosh, рабочие станции Sun и так далее. Даже в рамках компьютеров, созданных на базе процессоров Intel, существует несколько платформ, например, Microsoft Windows 2000, Windows NT, Windows ХР, OS/2, Solaris, различные разновидности операционной системы UNIX с графической оболочкой X­Windows. Приложения Java предназначены для работы на различных платформах и не зависят от конкретного типа процессора и операционной системы.

Программы, составленные на языке программирования Java, можно разделить по своему назначению на две большие группы.

К первой группе относятся приложения Java, предназначенные для автономной работы под управлением специальной интерпретирующей машины Java. Реализации этой машины созданы для всех основных компьютерных платформ.

Вторая группа - это так называемые апплеты (applets). Апплеты представляют собой разновидность приложений Java, которые интерпретируются виртуальной машиной Java, встроенной практически во все современные браузеры.

Приложения, относящиеся к первой группе (мы будем называть их просто приложениями Java), - это обычные автономные программы. Так как они не содержат машинного кода и работают под управлением специального интерпретатора, их производительность заметно ниже, чем у обычных программ, составленных, например, на языке программирования C++. Однако не следует забывать, что программы Java без перетрансляции способны работать на любой платформе, что само по себе имеет большое значение в плане разработок для Internet.

Для повышения производительности приложений Java в современных браузерах используется компиляция "на лету"- Just-In-Time compilation (JIT). При первой загрузке апплета его код транслируется в обычную исполнимую программу, которая сохраняется на диске и запускается. В результате общая скорость выполнения апплета Java увеличивается в несколько раз.

Язык Java является объектно-ориентированным и поставляется с достаточно объемной библиотекой классов. Так же как и библиотеки классов систем разработки приложений на языке C++, библиотеки классов Java значительно упрощают разработку приложений, представляя в распоряжение программиста мощные средства решения распространенных задач. Поэтому программист может больше внимания уделить решению прикладных задач, а не таких, как, например, организация динамических массивов, взаимодействие с операционной системой или реализация элементов пользовательского интерфейса.

Под жизненным циклом мы будем понимать процесс, необходимый для создания работающего приложения. Для программ на Java он отличается от жизненного цикла программ на других языках программирования. (рис.1 ).

Из рисунка видно, что исходная Java-программа должна быть в файле с расширением java. Программа транслируется в байт-код компилятором javac.exe. Оттранслированная в байт-код программа имеет расширение class. Для запуска программы нужно вызвать интерпретатор java.exe, указав в параметрах вызова, какую программу ему следует выполнять. Кроме того, ему нужно указать, какие библиотеки нужно использовать при выполнении программы. Библиотеки размещены в файлах с расширением jar.

Программные файлы (\*.java)

Транслятор (javac.exe)

Файлы Р-кода (\*.class)

Библиотеки (\*.jar)

Интерпретатор (java.exe)

Фактически выполняет программу на Java

Рис. 1. Жизненный цикл программ на языке Java

Как уже было отмечено, Java - объектно-ориентированный язык. Это дало возможность зафиксировать достаточно компактное ядро языка, ограничив его сравнительно небольшим числом различных синтаксических конструкций, а большую часть возможностей языка вводить с помощью классов. Так, нити включены в язык с помощью классов Thread и ThreadGroup. В виде классов реализованы и такие базовые языковые понятия, как функции обработки строк и обработка исключений. Развитие языка Java тоже ведется путем включения в него новых классов и пакетов (пакеты заменяют в системе программирования Java файлы-заголовки окружения C/C++, однако, в отличие от файлов-заголовков, пакеты содержат как спецификацию классов, так и их реализацию; подробнее о пакетах см. [14 ]). Такой способ расширения языка удобен тем, что старые компиляторы остаются пригодными и для расширенного языка. Многие новые свойства языка, введенные в него через новые классы и пакеты

Что же было исключено из C++ при разработке Java? Прежде всего, были исключены указатели. Указатели или адреса в памяти - наиболее мощное средство написания высокоэффективных программ в окружении C/C++, но это и наиболее опасное средство этих языков. И дело даже не в том, что, как отмечают авторы языка Java, при недостаточно аккуратном обращении с указателями могут возникать трудно устранимые ошибки в C++-программе. Более существенные трудности в работе с указателями выявляются при разработке распределенных программ, когда требуется осуществить удаленный вызов функции (метода), среди параметров которой есть указатели.

# 1. JAVA - объектно-ориентированный язык программирования

Java является объектно-ориентированным языком программирования. Если сравнивать в этом смысле Java и C++, то между ними есть существенные различия. В ней нет средств, позволяющих писать не объектно-ориентированные программы. Из этого сразу следует один вывод. Нельзя научиться программировать на Java, не овладев основами объектно-ориентированного подхода. Отметим 5 принципов ООР -

● Все является объектом. Все данные программы хранятся в объектах. Каждый объект создается (есть средства для создания объектов), существует какое-то время, потом уничтожается

● Программа есть группа объектов, общающихся друг с другом. Кроме того, что объект хранит какие-то данные, он умеет выполнять различные операции над своими данными и возвращать результаты этих операций. Теоретически эти операции выполняются как реакция на получение некоторого сообщения данным объектом. Практически это происходит при вызове метода данного объекта

● Каждый объект имеет свою память, состоящую из других объектов и/или элементарных данных. Объект хранит некоторые данные. Эти данные - это другие объекты, входящие в состав данного объекта и/или данные элементарных типов, такие как целое, вещественное, символ, и т.п

● Каждый объект имеет свой тип (класс). Т.е. в объектно-ориентированном подходе не рассматривается возможность создания произвольного объекта, состоящего из того, например, что мы укажем в момент его создания. Все объекты строго типизированы. Мы должны сначала описать (создать) тип (класс) объекта, указав в этом описании из каких элементов (полей) будут состоять объекты данного типа. После этого мы можем создавать объекты этого типа. Все они будут состоять из одних и тех же элементов (полей).

● Все объекты одного и того же типа могут получать одни и те же сообщения. Кроме описания структуры данных, входящих в объекты данного типа, описание типа содержит описание всех сообщений, которые могут получать объекты данного типа (всех методов данного класса). Более того, в описании типа мы должны задать не только перечень и сигнатуру сообщений данного типа, но и алгоритмы их обработки

## 1.1. Базовые типы данных

В языке Java определено восемь базовых типов данных. Для каждого базового типа данных отводится конкретный размер памяти. Этот размер, не зависит от платформы, на которой выполняется приложение Java:

|  |  |  |
| --- | --- | --- |
| Тип данных | Размер занимаемой области памяти | Значение по умолчанию |
| boolean | 8 | false |
| byte | 8 | 0 |
| char | 16 | 'x0' |
| short | 16 | 0 |
| int | 32 | 0 |
| long | 64 | 0 |
| float | 32 | 0.0F |
| double | 64 | 0.0D |

Все базовые типы данных по умолчанию инициализируются, поэтому программисту не нужно об этом беспокоиться. Вы можете также инициализировать переменные базовых типов в программе или при их определении.

## 1.2. Операции (operators) в языке Java

Большинство операций Java просты и интуитивно понятны. Это такие операции, как +, -, \*, /, <,> и др. Операции имеют свой порядок выполнения и приоритеты. В выражении сначала выполняется умножение, а потом сложение, поскольку приоритет у операции умножения выше, чем у операции сложения. Но операции Java имеют и свои особенности. Не вдаваясь в детальное описание простейших операций, остановимся на особенностях.

Начнем с присваивания. В отличии от ряда других языков программирования в Java присваивание - это не оператор, а операция. Операция присваивания обозначается символом '='. Она вычисляет значение своего правого операнда и присваивает его левому операнду, а также выдает в качестве результата присвоенное значение. Это значение может быть использовано другими операциями. Последовательность из нескольких операций присваивания выполняется справа налево.

В простейшем случае все выглядит как обычно.

x = a + b;

Здесь происходит именно то, что мы интуитивно подразумеваем, - вычисляется сумма a и b, результат заносится в x. Но вот два других примера.

a = b = 1; a+b;

В первом сначала 1 заносится в b, результатом операции является 1, потом этот результат заносится в a. Во втором примере вычисляется сумма a и b и результат теряется. Это бессмысленно, но синтаксически допустимо

### 1.2.1. Операции сравнения

Это операции >, <, >=, <=, != и ==. Следует обратить внимание, что сравнение на равенство обозначается двумя знаками '='. Операндами этих операций могут быть арифметические данные, результат - типа boolean.

Операции отношений != и == работают для всех объектов, но их смысл может быть не сразу очевиден. Вот пример

**Public class** Op{

……………………

**Integer** n1 = **new** Integer(47);

**Integer** n2 = **new** Integer(47);

System.out.print (n1 == n2);

System.out.print (n1 != n2);

}

Выражение System.out.print (n1 ==n2) выведет результат булевского сравнения, содержащегося в скобках. Конечно, результат должен быть истина (true), а во втором случае – false, так как оба целых имеют одинаковые значения. Но в то время, как содержимое объектов одинаковое, ссылки на них разные а операторы != и == сравнивают именно ссылки. Поэтому результат первого выражения false, а второго – true. Как же действительно сравнить содержимое объектов? Вы должны использовать метод equals(), существующий для всех объектов.

### 1.2.2. Операции инкремента, декремента

Это операции ++ и -- Так y++ (инкремент) является сокращенной записью y = y +1, аналогично и с операцией декремента (--). Но с этими операциями есть одна тонкость. Они существуют в двух формах - префиксной (++y) и постфиксной (y++). Действие этих операций одно и то же - они увеличивают (операции декремента - уменьшают) свой операнд на 1, а вот результат у них разный. Префиксная форма в качестве результата выдает уже измененное на 1 значение операнда, а постфиксна - значение операнда до изменения.

## 1.3. Литералы (константы)

Существуют арифметические (для указания типа константы применяются суффиксы: l (или L) - long, f (или F) - float, d (или D) – double), логические (это true (истина) и false (ложь)), строковые (записываются в двойных кавычках), символьные (записываются в апострофах, например 'F', 'ш')

## 1.4. Операторы

|  |  |
| --- | --- |
| Оператор | Синтаксис |
| Выражение | <выражение> |
| Условный оператор | **if** ( <условие> ) <оператор1>[**else** <оператор2>] |
| цикла по предусловию (**while**) | **while** ( <условие> ) <оператор> |
| цикла по постусловию (**do while**) | **do** <оператор>**while** ( <условие> ); |
| цикла "со счетчиком" (**for**) | **for** (<инициализация>; <условие>; <инкремент> ) <оператор> |
| Операторы **break** и **continu**e | Операторы **break** и **continue** являются структурированными аналогами goto. Они могут применяться в циклах, а **break** еще и в операторе выбора (**switch**). Выполнение оператора **break** приводит к немедленному завершению цикла. Оператор **continue** вызывает окончание текущего витка цикла и начало нового |
| Оператор выбора (**switch**) | **switch** ( <выражение> ) { **case** <константа1>: <операторы1> **case** <константа2>: <операторы2> . . . [**default**: <операторы\_D>]} |

##

## 1.5. Массивы в Java

В Java есть как одномерные, так и многомерные массивы. Но реализация массивов в Java имеет свои особенности. Во-первых массив в Java это объект. Этот объект состоит из размера массива (поле length) и собственно массива.

Рассмотрим сначала простейшие одномерные массивы базовых типов - int intAry[]; или int[] intAry;. Это описание переменной (или поля) intAry - ссылки на массив. Соответственно, в этом описании размер массива не указывается и сам массив не создается. Как и любой другой объект массив должен быть создан операцией new - intAry = new int[10];

Для массивов допустима инициализация списком значений. int intAry[] = {1, 2, 3, 4}; Здесь описан массив из 4-х элементов и сразу определены их начальные значения. Элементы массивов в Java нумеруются с 0. При обращении к элементу массивы его индексы задаются в квадратных скобках. Java жестко контролирует выход за пределы массива. При попытке обратиться к несуществующему элементу массива возникает IndexOutOfBoundException.

Для двумерных массивов ставится не одна пара скобок, а две, для трехмерных - три и т.д. Например. s = sAray[i][0]; tAray[i][j][k] = 10; Двумерный массив - это массив ссылок на объекты-массивы. Трехмерный массив - это массив ссылок на массивы, которые, в свою очередь, являются массивами ссылок на массивы. Как уже указывалось, массив является объектом, который, в частности, хранит поле length - размер массива. Это позволяет задавать обработку массивов произвольно. Они строятся по принципу "массив массивов". Возможные способы инициализации массивов:

1. явное создание

int ary[][] = new int[3][3];

2. использование списка инициализации

int ary[][] = new int[][] {

 {1, 1, 1},

 {2, 2, 2},

 {1, 2, 3}, };

3. массивы в языке Java являются объектами некоторого встроенного класса, для этого класса существует возможность определить размер массива, обратившись к элементу данных класса с именем length, например:

int[] nAnotherNumbers;

nAnotherNumbers = new int[15];

**for**(**int** i = 0; i < nAnotherNumbers.length; i++)

{

 nAnotherNumbers[i] = nInitialValue;

}

## 1.6. Комментарии

В языке Java используются однострочные и блочные комментарии // и /\* \*/, аналогичные комментариям, применяемым в C++ . Введен также новый вид комментария /\*\* \*/, который может содержать дескрипторы вида:

@author - задает сведения об авторе;

@exception - задает имя класса исключения;

@param - описывает параметры, передаваемые методу;

@return - описывает тип, возвращаемый методом;

@throws - описывает исключение, генерируемое методом.

Из java-файла, содержащего такие комментарии, соответствующая утилита javadoc.exe может извлекать информацию для документирования классов и сохранения ее в виде HTML-документа.

## 1.7. Первая программа на языке Java

Элементарные строительные блоки в Java называются классами (как и в C++). Класс состоит из данных и кода для работы с ними. В средствах для разработки на языке Java все стандартные классы, доступные программисту, объединены для удобства в упаковки — еще одни элементарные блоки Java-программ.

Вот простейшая программа, приводимая во многих учебниках по Java:

**class** JavaTest

{

**public static void** main(String args[])

 System.out.println("Hello, World!");

}}

Откомпилируем программу. Если Вы набрали текст правильно, то в результате компиляции на экран будет выведено одно слово Hello. Если же нет, то на экран будут выданы сообщения об ошибках.

Данный пример примитивен, но, тем не менее, на этом примере можно познакомиться с очень важными понятиями.

Рассмотрим, что он демонстрирует:

● весь программный код в Java заключен внутри классов. Не может быть никакого программного текста (за исключением нескольких специальных директив) вне класса (или интерфейса).

● Каждый файл с именем **Name.java** должен содержать класс с именем **Name** (причем, учитывается регистр). Каждый **public**-класс с именем **Name** должен быть в своем файле **Name.java**.

● Внутри указанного файла могут быть и другие классы, но их имена должны отличаться от **Name** и они не должны быть **public**.

● Внутри класса может быть конструкция

**public static void** main(String[] args) {

 . . .

}

Это метод класса. Здесь main - имя метода, **public, static, void** - это описатели, Для описания ограничений доступа используются ключевые слова **public, private, protected**. Они являются опциональными описателями и дают нам три варианта ограничений доступа плюс четвертый вариант, если не указан не один из этих описателей.

.● Указанный метод main является специальным случаем. При запуске Java-программы мы указываем имя класса, и Java-машина ищет этот класс среди всех доступных ей файлов \*.class, и в этом классе запускает на выполнение метод main

● Описание метода main должно быть в точности таким, как приведено в примере (можно разве что изменить имя args на какое-то другое).

● В скобках после имени метода указываются параметры метода. Для main-метода параметры должны быть такими как указано. Это - массив строк. При вызове программы на Java можно задать параметры вызова. Java-машина обработает их и сформирует массив строк, который будет передан в main-метод в качестве параметра.

## 1.8. Цель, требования и рекомендации к выполнению задания

Цель выполнения задания: ознакомление с основными конструкциями языка Java, операциями, литералами, различными типами операторов.

Требования и рекомендации к выполнению задания:

1. проанализировать полученное задание, выделить информационные объекты и действия;
2. разработать программу с использованием требуемых типов и операторов.

## 1.9. Задания

1. Ввести с консоли n целых чисел и поместить их в массив. На консоль вывести четные и нечетные числа.
2. Ввести с консоли n целых чисел и поместить их в массив. На консоль вывести числа, которые делятся на 3 или на 9.
3. Ввести с консоли n целых чисел и поместить их в массив. На консоль вывести числа, которые делятся на 5 или на 10.
4. Ввести с консоли n целых чисел и поместить их в массив. На консоль вывести наибольший общий делитель и наименьшее общее кратное этих чисел.
5. Ввести с консоли n целых чисел и поместить их в массив. На консоль вывести простые числа.
6. Ввести с консоли n целых чисел и поместить их в массив. На консоль вывести "счастливые" числа.
7. Для произвольной цифры от 0 до 9 вывести на консоль ее значение прописью. Например, для цифры 9 на консоли должна быть напечатана строка «Девять».
8. Создайте приложение, которое покажет, что для выражения an+bn=cn (теорема Ферма) нет натуральных решений от 1 до 100 и n>2. Убедитесь, что есть решения для n=2, и выведите их в консоль.
9. Вычислить выражение 1- 1/2 +1/3 -1/4 + +1/ 9999-1/10000 , используя оператор условия.
10. Ввести с консоли n целых чисел и поместить их в массив. На консоль вывести числа Фибоначчи: f0 = f1 = 1, f (n) = f (n-1) + f (n-2).
11. Ввести с консоли n целых чисел и поместить их в массив. На консоль вывести числа-полиндромы, значения которых в прямом и обратном порядке совпадают.
12. Ввести с консоли n целых чисел и поместить их в массив. На консоль вывести период десятичной дроби р = m/n для первых двух целых положительных чисел n и m, расположенных подряд.
13. Ввести с консоли n целых чисел и поместить их в массив. Построить треугольник Паскаля для первого положительно го числа.
14. Создайте приложение, которое осуществит перевод чисел из десятичной системы счисления в двоичную и шестнадцатеричную.
15. Написать приложение, выводящее n строк с переходом и без перехода на новую строку.
16. Написать приложение для ввода пароля из командной строки и сравнения его со строкой-образцом.
17. Написать программу ввода целых чисел как аргументов командной строки, подсчета их суммы (произведения) и вывода результата на консоль.
18. Написать приложение, выводящее фамилию разработчика, дату и время получения задания, а также дату и время сдачи задания. Для получения последней даты и времени использовать класс Date из пакета Java.util (Объявление объекта Date d = new Date();) или статический метод класса System.currentTimeMillis ().

# 2. Абстрактные классы и Интерфейсы

## 2.1. Абстрактные классы

Класс может представлять собой как бы заготовку, в которой часть методов реализована, а часть - нет. В этом случае в описании класса перед словом class должен стоять описатель abstact и при описании нереализованных методов тоже должен использоваться этот описатель. Пример

**public abstract class** D {

 . . .

 **int** g1(int s) {

 . . . }

 **public abstract** g2(String str);

 . . .}

В классе D метод g1 - это обычный метод, g2 - абстрактный, он содержит только заголовок, но не содержит реализации. Как видно из примера, тело абстрактного метода отсутствует, сразу после заголовка метода стоит точка с запятой.

Абстрактный класс не может использоваться непосредственно для порождения объектов. Для этого необходимо, используя этот класс как базовый, породить другой класс, в котором нужно определить все абстрактные методы. Тогда можно будет создавать объекты.

С другой стороны не запрещено описывать переменные абстрактного класса. Просто им нужно присваивать ссылки на объекты неабстрактных классов.

##

## 2.2. Интерфейсы

Понятие интерфейса похоже на абстрактный класс. Интерфейс — это полностью абстрактный класс, не содержащий никаких полей, кроме констант (static final - поля).

Существует, однако, серьезное отличие интерфейсов от классов вообще и от абстрактных классов, в частности. Интерфейсы допускают множественное наследование. Т.е. один класс может удовлетворять нескольким интерфейсам сразу.

Это связано с тем, что интерфейсы не порождают проблем с множественным наследованием, поскольку они не содержат полей.

Синтаксис:

**public interface** Stack{

 **int** size = 100;

 **public void** push (int x);

 **public int** pop();

 **int** f(String s);

}

Это описание интерфейса Stack. Внутри скобок могут находиться только описания методов (без реализации) и описания констант (static final — полей). В данном случае интерфейс Stack содержит, в частности метод f(...).

**public class** R **implements** Serializable, Stack {

……………

}

Класс R реализует интерфейсы Serializable и Stack.

Внутри класса, реализующего некоторый интерфейс, должны быть реализованы все методы, описанные в этом интерфейсе. Поскольку Stack имеет метод f(...), то в классе R он должен быть реализован:

**public class** R **implements** Serializable, Stack {

 . . .

 **public int** f(String s) {

 ... }

...}

В интерфейсе f(...) описан без описателя public,в классе R с описателем public. Дело в том, что все методы интерфейса по умолчанию считаются public, так что этот описатель там можно опустить. А в классе R необходимо его использовать явно. Еще одним общим моментом интерфейсов и абстрактных классов является то, что хотя и нельзя создавать объекты интерфейсов, но можно описывать переменные типа интерфейсов.

Интерфейсы широко используются при написании различных стандартов. Стандарт определяет, в частности, набор каких-то интерфейсов. И, кроме того, он содержит вербальное описание семантики этих интерфейсов. После этого, прикладные разработчики могут писать программы, используя интерфейсы стандарта. А фирмы-разработчики могут разрабатывать конкретные реализации этих стандартов. При внедрении (deployment) прикладного программного обеспечения можно взять продукт любой фирмы-разработчика, реализующий данный стандарт (на практике, конечно, все несколько сложнее).

## 2.3. Цель, требования и рекомендации к выполнению задания

Цель выполнения задания: реализовать абстрактные классы или интерфейсы, а также наследование и полиморфизм для следующих классов:

**Требования и рекомендации к выполнению задания:**

1. проанализировать полученное задание, выделить информационные объекты и действия;
2. разработать программу с использованием абстрактных классов или интерфейсов;
3. использовать при разработке наследование и полиморфизм.

## 2.4. Задания

1. Абстрактный класс Книга (Шифр, Автор, Название, Год, Издательство). Подклассы Справочник и Энциклопедия.
2. interface Abiturient←abstract class Student←class StudentOfFaculty.
3. interface Сотрудник←class Инженер←class Руководитель.
4. interface Учебное Заведение←class Колледж←class Университет.
5. .interface Здание←abstractclassОбщественноеЗдание←class Театр.
6. .interface Mobile←abstract class Siemens Mobile←class Model.
7. interface Корабль←abstract class Военный Корабль←class Авианосец.
8. interface Врач←class Хирург←class Нейрохирург.
9. .interface Корабль←class Грузовой Корабль←class Танкер.
10. interface Диск←abstractclass Директория←class Файл.

**Требования и рекомендации к выполнению следующего задания:**

В следующих заданиях требуется создать суперкласс (абстрактный класс, интерфейс) и определить общие методы для данного класса. Создать подклассы, в которых добавить специфические свойства и методы. Часть методов переопределить. Создать массив объектов суперкласса и заполнить объектами подклассов. Объекты подклассов идентифицировать конструктором по имени или идентификационному номеру. Использовать объекты подклассов для моделирования реальных ситуаций и объектов

1. Создать суперкласс Транспортное средство и подклассы Автомобиль, Велосипед, Повозка. Подсчитать время и стоимость перевозки пассажиров и грузов каждым транспортным средством.
2. Создать суперкласс Грузоперевозчик и подклассы Самолет, Поезд, Автомобиль. Определить время и стоимость перевозки для указанных городов и расстояний.
3. Создать суперкласс Пассажир-оперевозчик и подклассы Самолет, Поезд, Автомобиль. Определить время и стоимость передвижения.
4. Создать суперкласс Учащийся и подклассы Школьник и Студент. Создать массив объектов суперкласса и заполнить этот массив объектами. Показать отдельно студентов и школьников.
5. Создать суперкласс Музыкальный инструмент и классы Ударный, Струнный, Духовой. Создать массив объектов Оркестр. Выдать состав оркестра.
6. Определить суперкласс Множество и подкласс Кольцо (операции сложения и умножения, обе коммутативные и ассоциативные, связанные законом дистрибутивности). Ввести кольца целых чисел многочленов, систему классов целых чисел, сравнимых по модулю. Кольцо является полем, если в нем определена операция деления, кроме деления на ноль.
7. Создать абстрактный класс Работник фирмы и подклассы Менеджер, Аналитик, Программист, Тестировщик, Дизайнер.
8. Создать суперкласс Домашнее животное и подклассы Собака, Кошка, Попугай. С помощью конструктора установить имя каждого животного и его характеристики.
9. Создать базовый класс Садовое дерево и производные классы Яблоня, Вишня, Груша и другие. С помощью конструктора автоматически установить номер каждого дерева. Принять решение о пересадке каждого дерева в зависимости от возраста и плодоношения.

##

## 2.5. Пример выполнения задания.

Программа через каждые 10 секунд выдает сообщение о выходе.

Класс javax. swing содержит класс Timer, который можно использовать для отсчета интервалов времени. Например, если в программе предусмотрены часы, то с помощью класса Тimer можно отсчитывать каждую секунду и обновлять циферблат часов.

Устанавливая таймер, задаем интервал времени и указываем, что должно произойти по его истечении. Как указать таймеру, что он должен делать? Классы из стандартной библиотеки языка Java используют объектно-ориентированный подход. Программист должен передать таймеру объект некоторого класса. После этого таймер вызывает один из методов данного объекта. Передача объекта — более гибкий механизм, чем вызов функции, поскольку объект может нести с собой дополнительную информацию.

Разумеется, таймер должен знать, какой метод он должен вызвать. Для этого таймеру нужно указать объект класса, реализующего интерфейс ActionListener из Java.awt.event. Вот как выглядит этот интерфейс

public interface ActionListener

{

 void actionPerformed(ActionEvent  event);

}

По истечении заданного интервала времени таймер вызывает метод actionPerformed.

import java.awt.\*;

import java.awt.event.\*;

import java.util.\*;

import javax.swing.\*;

import javax.swing.Timer;

**public class** TestTimer

{

 **public static void** main (String[] args)

 {

 ActionListener listner = **new** TimePrinter();

// Создание таймера, который вызывает блок прослушивания

// каждые 10 секунд

 Timer t = **new** Timer(1000, listner); t.start();

 JOptionPane.showMessageDialog(null, "Выход?"); System.exit(0);

}}

**class** TimePrinter **implements** ActionListener

{

 **public void** actionPerformed(ActionEvent event)

 {

Date now = **new** Date();

System.out.println("Текущее время "+now);

Toolkit.getDefaultToolkit().beep(); }

}

# 3. Знакомство с библиотекой Swing

## 3.1. Основы оконной графики

Для поддержки создания пользовательских интерфейсов существуют библиотеки классов, позволяющих создавать и поддерживать окна, использовать элементы управления (кнопки, меню, полосы прокрутки и др.), применять инструменты для создания графических приложений. Графические инструменты в языке Java реализованы с помощью двух библиотек:

1.·Пакет AWT (загружается java.awt) содержит набор классов, позволяющих выполнять графические операции и создавать элементы управления.

2.·Пакет Swing (загружается javax. swing, имя javax обозначает, что пакет не является основным, а только расширением языка) содержит новые классы, по большей части аналогичные AWT. К именам этих классов добавляется J (JButton, JLabel и т. д.). Пакет является частью библиотеки JFC (Java Foundation Class).

Работа с окнами и графикой в Java осуществляется в апплетах и графических приложениях. Апплеты - это небольшие программы, встраиваемые в Web-документ и использующие для своей визуализации средства Web-браузера. Графические приложения сами отвечают за свою прорисовку.

Библиотека Swing, в отличие от AWT, более полно реализует парадигму объектно-ориентированного программирования. Однако библиотека Swing, хотя и является более современной, но полностью не заменяет собой AWT. В частности, обработка событий остается неизменной. К преимуществам библиотеки Swing следует отнести повышение надежности, расширение возможностей пользовательского интерфейса, а также независимость от платформы. Кроме того, эту библиотеку легче использовать и она визуально более привлекательна.

В ранних (1.0.x) версиях Java использовались "тяжелые" компоненты AWT, связанные с аппаратными платформами и имеющие ограниченные возможности. Дальнейшее развертывание концепции "write once, run everywhere" ("написать однажды, запускать везде") привело к тому, что в версии 1.1.x наметился переход к таким компонентам, которые бы не были завязаны на конкретные "железо" и операционные системы. Такого рода классы компонентов, написанные на Java, были объединены в библиотеку под названием Swing. Эти классы доступны разработчикам в составе как JDK, так и отдельного продукта JFC (Java Foundation Classes). Причем для совместимости со старыми версиями JDK старые компоненты из AWT остались нетронутыми, хотя компания JavaSoft, отвечающая за выпуск JDK, рекомендует не смешивать в одной и той же программе старые и новые компоненты. Кроме пакета Swing указанные библиотеки содержат большое число компонентов JavaBeans, которые могут использоваться как для ручной, так и для визуальной разработки пользовательских интерфейсов.

## 3.2. Модель событий в Swing

В модели событий любой графический компонент может порождать (initiate) c событие( запускать (fire) событие). Каждое отдельное событие представлено определенным классом. Когда событие запущено, оно приходит к одному или нескольким слушателям (listener), которые его обрабатывают. Таким образом, места возникновения события и обработки события могут быть разделены. Компоненты Swing обычно используются без изменений, как есть, однако код для обработки их событий каждый раз пишется новый.

Слушатель каждого события – это объект класса, реализующего определенный вид интерфейса, характерного для всех слушателей этого события. Необходимо создать для обработки события объект-слушатель и зарегистрировать его в компоненте, который запускает интересующее его событие. Регистрация проводится с помощью подходящего метода addXXXListener(), находящегося в компоненте, который запускает событие, где ХХХ обозначает тип события, которое будет обрабатываться слушателем

Можно легко узнать, какие события можно обрабатывать в компоненте, для этого нужно посмотреть имена методов addXXXListener. Таким образом, все действия по обработке события будут находиться в классе слушателя. При создании класса слушателя единственным необходимым условием является реализация подходящего интерфейса

## 3.3. Цель, требования и рекомендации к выполнению задания

Цель выполнения задания: освоение приемов применения компонентов библиотеки Swing

**Требования и рекомендации к выполнению задания:**

1. проанализировать полученное задание, выделить информационные объекты и действия;
2. разработать программу с использованием соответствующих элементов (классов) библиотеки Swing;

## 3.4. Задания

1. Создать приложение. Поместить на него текстовое поле JTextField, кнопку JButton и метку JLabel. В метке отображать все ввденные символы, разделяя их пробелами.
2. Поместить в приложение две панели JPanel и кнопку. Первая панель содержит поле ввода и метку "Поле ввода"; вторая - поле вывода и метку "Поле вывода". Для размещения в окне двух панелей и кнопки "Скопировать" использовать менеджер размещения BorderLayout.
3. Изменить задачу 2 так, чтобы при нажатии на кнопку "Скопировать" текст из поля ввода переносится в поле вывода, а поле ввода очищается.
4. Задача 2 модифицируется так, что при копировании поля ввода нужно, кроме собственно копирования, организовать занесение строки из поля ввода во внутренний список. При решении использовать коллекцию, в частности ArrayList.
5. К условию задачи 2 добавляется еще одна кнопка с надписью "Печать". При нажатии на данную кнопку весь сохраненный список должен быть выведен в консоль. При решении использовать коллекцию, в частности TreeSet.
6. Написать программу для построения таблицы значений функции y = a-√x\*cos(ax). Использовать метку JLabel, содержащую текст "Функция: у = a-√x \* cos(ax)"; панель, включающую три текстовых поля JTextField, содержащих значения параметра, шага (например, 0.1) и количества точек. Начальное значение х=0. С каждым текстовым полем связана метка, содержащая его название. В приложении должно находиться текстовое поле со скроллингом, содержащее полученную таблицу.
7. Создать форму с набором кнопок так, чтобы надпись на первой кнопке при ее нажатии передавалась на следующую, и т.д.
8. Создать форму с выпадающим списком так, чтобы при выборе элемента списка на экране появлялись GIF-изображения, двигающиеся в случайно выбранном направлении по приложению.
9. В приложении изобразить прямоугольник (окружность, эллипс, линию). Направление движения объекта по экрану изменяется на противоположное щелчком по клавише мыши. При этом каждый второй щелчок меняет цвет фона.
10. Создать фрейм с изображением окружности. Длина дуги окружности изменяется нажатием клавиш от 1 до 9.
11. Создать фрейм с кнопками. Кнопки "вверх", "вниз", "вправо", "влево" двигают в соответствующем направлении линию. При достижении границ фрейма линия появляется с противоположной стороны.
12. Создать фрейм и разместить на нем окружность (одну или не сколько). Объект должен "убегать" от указателя мыши. При приближении на некоторое расстояние объект появляется в другом месте фрейма.
13. Создать приложение с изображением графического объекта. Объект на экране движется к указателю мыши, когда последний находится в границах приложения.
14. Изменить задачу 12 так, чтобы количество объектов зависело от размеров приложения и изменялось при "перетягивании" границы в любом направлении.
15. Промоделировать в приложении вращение спутника вокруг планеты по эллиптической орбите. Когда спутник скрывается за планетой, то не он виден.
16. Промоделировать в приложении аналоговые часы (со стрелками) с кнопками для увеличения/уменьшения времени на час/минуту.

## 3.5. Пример выполнения задания.

Познакомимся с классом JTabbedPane Tabbed\_Prog.java. Рассмотрим простой пример, чтобы познакомиться с этим классом подробнее.

//Tabbed\_Prog.java

// Визуальное приложения с JTabbedPane.

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

**public class** Tabbed\_Prog extends JFrame {

 Tabbed\_Prog () {

 super("Визульное приложения с JTabbedPane");

 try {

 UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

 }

 catch(Exception e) {

 }

 setSize(400, 200); Container c = getContentPane();

 JTabbedPane tp = new JTabbedPane();

 c.add(tp, BorderLayout.CENTER); JPanel pn1 = new JPanel();

 tp.add(pn1, "Hello"); JPanel pn2 = new JPanel();

 tp.add(pn2, "Good-bye");

 WindowListener wndCloser = new WindowAdapter() {

 public void windowClosing(WindowEvent e) {

 System.exit(0); }

 };

 addWindowListener(wndCloser); setVisible(true); }

 public static void main(String[] args) {

 Tabbed\_Prog d = new Tabbed\_Prog ();

 }

}

Оттранслируем и запустим приложение. Приложение формирует такое окно рис.2



Здесь две закладки, между которыми можно переключаться. Рассмотрим, как это реализовано. Сначала создается объект JTabbedPane и заносится в центр экрана

 JTabbedPane tp = new JTabbedPane();

 c.add(tp, BorderLayout.CENTER);

Потом создаются две панели pn1 и pn2 и добавляются на панель с закладками.

 JPanel pn1 = new JPanel();

 tp.add(pn1, "Hello");

 JPanel pn2 = new JPanel();

 tp.add(pn2, "Good-bye");

При этом применяется метод add с двумя параметрами, второй из которых содержит текст. Именно этот текст выводится на данной закладке. Количество закладок определяется количеством панелей добавленных к экземпляру класса JTabbedPane. Т.е. сформировать панели с закладками довольно просто. Класс JTabbedPane имеет два конструктора

 **public** JTabbedPane()

Конструктор по умолчанию. Формирует горизонтальную линейку закладок вверху.

 **public** JTabbedPane(int tabPlacement)

Позволяет разместить закладки как горизонтально, так и вертикально, внизу и вверху панели. Соответствующие константы: TOP, BOTTOM, LEFT, RIGHT.

Попробуем в нашем приложении заменить конструктор по умолчанию на конструктор с параметром JTabbedPane.LEFT, т.е.

 JTabbedPane tp = **new** JTabbedPane(JTabbedPane.LEFT);

Оттранслируем и посмотрим результат.

Каждая из закладок - это панель, на которую можно поместить свои визуальные компоненты. Это позволяет разместить на одном и том же участке экрана достаточно большое количество информации

# 4. Апплеты

Апплеты это маленькие прикладные программы на языке Java, которые размещаются на серверах Internet, транспортируются клиенту по сети, автоматически устанавливаются и запускаются на месте как часть документа WWW. Апплеты позволяют вставлять в документ, распространяемый по сети, поля, содержание которых меняется во времени (например, текущая дата, текущий курс валюты и т.п.), организовывать в нем "бегущие строки", мультипликацию.

Исходный текст апплета на языке Java

Компилятор Java

Загрузчик байт-кода

Верификаторбайт-кода

Интерпретатор байт-кода

Компьютер пользователя

Байт-код Java

Интернет

Динамический компилятор

Поддержка выполнения системы Java

Рис. 3. Выполнение Java-апплета

.Апплетами могут быть небольшие прикладные программы, распространяемые по Web, такие как калькулятор, преобразователь алгебраических выражений, вычислитель интегралов для первокурсников, различные игры и т.п. Апплеты компилируются в байт-код на компьютере их разработчика и распространяются по Web в виде байткодных файлов.

## 4.1. Проблема безопасности

Суть проблемы в том, что апплет - это, как правило, программа, полученная из внешнего источника. Соответственно, она опасна. По злому умыслу или из-за ошибки она может нанести вред. Более того, апплет как бы составляет одно целое с web-страничкой и для пользователя может быть даже не совсем очевидно, что безобидная web-страница скрывает за собой потенциально опасный программный код.

В целях безопасности апплеты, по умолчанию, сильно ограничены в своих правах. Эти ограничения определяются настройками конкретного браузера. Они могут быть ослаблены самим пользователем как в отношении всех апплетов, так и для конкретных апплетов.

Ограничений достаточно много, но мы рассмотрим только основные из них.

Апплеты не могут ни читать, ни писать на локальный диск. Они могут передавать информацию только в тот адрес, откуда загружен апплет. Окно апплета специально выделяется или делается какая-то другая пометка для того, чтобы пользователь представлял себе, что он имеет дело с апплетом, а не с обычной web-страницей.

## 4.2. Создание апплетов

Техника написания апплетов базируется на классе JApplet пакета javax.swing.

Этот класс имеет много своих методов и ряд методов, унаследованных от класса Applet. Однако, их изучение по документации мало что даст для овладения техникой построения апплетов. Просто нужно знать, как построить апплет при помощи этих методов.

Для построения апплета нужно создать класс - наследник класса JApplet и переопределить в нем ряд методов класса Applet. В классе JApplet эти методы реализованы как пустые заглушки, которые ничего не делают. При работе апплета внутри браузера он вызывает эти методы в определенных ситуациях. Если мы определим свои методы, то браузер вызовет их, а не методы класса Applet.

Рассмотрим эти методы.

 **public void** init()

Вызывается браузером сразу после загрузки аплета перед первым вызовом метода start(). Этот метод нужно переопределять практически всегда, если в аплете требуется хоть какая-то инициализация.

 **public void** start()

Вызывается браузером при каждом "посещении" данной страницы. Имеется в виду, что можно загрузить данную страницу, потом загрузить другую, не убирая данную, а потом вернуться к данной. Используется обычно в комбинации с методом stop для экономии ресурсов в том случае, например, если апплет выполняет некоторую анимацию. Тогда stop может ее остановить, а start запустить снова.

 **public void** stop()

Вызывается браузером при деактивизации данной страницы как в случае загрузки новой страницы без выгрузки данной, так и в случае выгрузки данной. В последнем случае stop вызывается перед destroy.

 **public void** destroy()

Вызывается браузером перед выгрузкой данной страницы.

Обычно при создании аплета переопределяют метод init() и реализуют в нем формирование экрана. При этом вся функциональность апплета обеспечивается слушателями (listeners) полей, кнопок и других активных визуальных компонент.

## 4.3. Апплеты и приложения

Практически любое диалоговое приложение на Java может быть реализовано так, что оно сможет работать и как программа и как апплет (не нужно только забывать об ограничениях, налагаемых на аплпеты по доступу к ресурсам машины).

Ниже приведен порядок, в котором вызываются методы класса Applet, с пояснениями, нужно или нет переопределять данный метод.

**init**

Метод init вызывается первым. В нем вы должны инициализировать свои переменные.

**start**

Метод start вызывается сразу же после метода init. Он также используется в качестве стартовой точки для возобновления работы после того, как апплет был остановлен. В

то время, как метод init вызывается только однажды—при загрузке апплета, start вызывается каждый раз при выводе HTML-документа, содержащего апплет, на экран. Так, например, если пользователь перейдет к новой WWW-странице, а затем вернется назад к странице с апплетом, апплет продолжит работу с метода start.

**paint**

Метод paint вызывается каждый раз при повреждении апплета. AWT следит за состоянием окон в системе и замечает такие случаи, как, например, перекрытие окна апплета другим окном. В таких случаях, после того, как апплет снова оказывается видимым, для восстановления его изображения вызывается метод paint.

**update**

Используемый по умолчанию метод update класса Applet сначала закрашивает апплет цветом фона по умолчанию, после чего вызывает метод paint. Если вы в методе paint заполняете фон другим цветом, пользователь будет видеть вспышку цвета по умолчанию при каждом вызове метода update—то есть, всякий раз, когда вы перерисовываете апплет. Чтобы избежать этого, нужно заместить метод update. В общем случае нужно выполнять операции рисования в методе update, а в методе paint, к которому будет обращаться AWT, просто вызвать update.

**stop**

Метод stop вызывается в тот момент, когда браузер покидает HTML-документ, содержащий апплет. При вызове метода stop апплет еще работает. Вы должны использовать этот метод для приостановки тех подпроцессов, работа которых необязательна при невидимом апплете. После того, как пользователь снова обратится к этой странице, вы должны будете возобновить их работу в методе start.

**destroy**

Метод destroy вызывается тогда, когда среда (например, браузер Netscape) решает, что апплет нужно полностью удалить из памяти. В этом методе нужно освободить все ресурсы, которые использовал апплет.

**repaint**

Метод repaint используется для принудительного перерисовывания апплета. Этот метод, в свою очередь, вызывает метод update. Однако, если ваша система медленная или сильно загружена, метод update может и не вызваться. Близкие по времени запросы на перерисовку могут объединяться AWT, так что метод update может вызываться спорадически. Если вы хотите добиться ритмичной смены кадров изображения, воспользуйтесь методом repaint (time)—это позволит уменьшить количество кадров, нарисованных не вовремя.

**repaint(time)**

Вы можете вызывать метод repaint, устанавливая крайний срок для перерисовки (этот период задается в миллисекундах относительно времени вызова repaint).

**repaint (x, у, w, h)**

Эта версия ограничивает обновление экрана заданным прямоугольником, изменены будут только те части экрана, которые в нем находятся.

**repaint(time, x, у, w, h)**

Этот метод—комбинация двух предыдущих.

## 4.4. Цель, требования и рекомендации к выполнению задания

Цель выполнения задания: реализовать соответствующий апплет, осуществить его запуск и выполнение.

**Требования и рекомендации к выполнению задания:**

1. проанализировать полученное задание, выделить информационные объекты и действия;
2. разработать программу апплета с использованием соответствующих элементов (классов) библиотеки Swing, графических элементов и др.;

## 4.5. Задания

Создать апплет по заданию из третьей лабораторной работы (пункт 3.4).

## 4.6. Пример выполнения задания.

В данном чисто демонстрационном примере присутствует метка, с надписью "Первый аплет", кнопка и текстовое поле, в которое при нажатии на кнопку выводится текст "Привет".

// AppDemo.java Файл AppDemo.java:

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

**public class** AppDemo extends JApplet {

 JTextField txt = new JTextField(15);

 **public** AppDemo() {}

 **public** void init() {

 Container c = getContentPane();

 c.setLayout(**new** FlowLayout());

 JLabel lbl = **new** JLabel("Первый аплет");

 c.add(lbl);

 JButton btn = **new** JButton("Нажать один раз");

 btn.addActionListener(**new** ActionListener() {

 public void actionPerformed(ActionEvent e) {

 txt.setText("Привет");

 }

 });

 c.add(btn);

 c.add(txt);

 } }

Отметим одну деталь. Как и для JFrame для JApplet следует добавлять компоненты в панель getContentPane(). Т.е. апплет выглядит очень похоже на обычную диалоговую программу на Java, здесь экран формируется в методе init().

### 4.6.1. Запуск апплетов

Апплет запускается браузером при просмотре web-страницы, в которой имеется ссылка на этот апплет. Ранее для запуска апплета было достаточно написать совсем простой код в составе HTML-странички. Например, для запуска вышеприведенного AppDemo можно было создать такой файл

<html>

<head><title>First Applet</title></head>

<body>

Какой-то текст на странице

<applet code=AppDemo width=400 height=200>

</applet>

</body>

</html>

Для тех, кто знаком с основами HTML, понять этот код не составит труда. Здесь для подключения апплета использован таг <applet>. Этот таг имеет параметры code, width и height. Ссылка на апплет реализована при помощи параметра code. Если апплет, как рекомендовано, заархивирован в jar-файл, для ссылки на архив нужно использовать параметр archive тага applet. Если попробовать воспользоваться вышеприведенным кодом AppDemo.shtml, то результат будет примерно таким, как показано на рис.4.



Рис.4. Результаты работы файла AppDemo.shtml.

### 4.6.2. Пример выполнения задания 2

В апплете FormDemo мы покажем приемы работы с компонентами, такими как переключатели, кнопки, текстовые поля и списки.

Мы разместили несколько таких компонент в окне этого апплета (рис. 7) таким образом, что они образуют собой форму. В этой форме вы можете ввести имя и фамилию, выбрать один из трех режимов работы, а также цвет.



Рис. 5. Окно аплета FormDemo

Переключатели First и Second активизируют однострочные текстовые поля редактирования Enter your first name и Enter your second name. После того как пользователь нажмет кнопку Ready, содержимое активных полей, а также состояние переключателей Mode 1, Mode 2 и Mode 3 будет отображено в многострочном поле редактирования. Это поле находится в нижней части окна апплета.

С помощью списка, расположенного справа от переключателя Mode 3, можно задавать цвет фона многострочного поля. Цвет устанавливается сразу после выбора новой строки из этого списка. К сожалению, при изменении размеров окна апплета находящиеся в нем компоненты изменяют свое расположение. Этот недостаток вы сможете устранить используя систему Layout Manager, с помощью которой можно управлять размещением компонент в окне апплета.

Листинг . Файл FormDemo.java

import java.applet.Applet;

import java.awt.\*;

import java.util.\*;

public class FormDemo extends Applet

{

 Button btReady; // Поле btReady хранит ссылку на кнопку с надписью Ready

// В полях chbox1 и chbox2 записаны ссылки на переключатели с независимой

// фиксацией, которые используются для активизации однострочных

// текстовых полей

 Checkbox chbox1;

 Checkbox chbox2;

// Поле grRadio хранит ссылку на группу переключателей с зависимой фиксацией,

//определяющих режимы работы Mode 1, Mode 2 и Mode 3:

 CheckboxGroup grRadio;

// Ссылки на эти переключатели находятся в следующих трех полях

 Checkbox rd1;

 Checkbox rd2;

 Checkbox rd3;

// В поле ch1 хранится ссылка на список, предназначенный для выбора цвета

 Choice ch1;

//Слева от однострочных полей редактирования в нашем окне имеются подписи,

//реализованные как объекты класса Label. Ссылки на эти объекты находятся

//в полях lbFirstName и lbSecondName

 Label lbFirstName;

 Label lbSecondName;

 TextField txtFirstName; // ссылка на многострочное текстовое поле хранится в поле

 TextField txtSecondName; // с именем txtFirstName и txtSecondName

 TextArea txta;

// Метод init занимается созданием компонент и добавлением их в окно апплета.

// Кроме того, этот метод изменяет цвет фона окна апплета и окон добавляемых

// компонент.

// Прежде всего метод **init** создает два переключателя с независимой фиксацией,

// два объекта класса Label и два однострочных поля редактирования текста

 public void init()

 {

 chbox1 = **new** Checkbox("First");

 add(chbox1);

 lbFirstName = **new** Label("Enter your first name:");

 add(lbFirstName);

 txtFirstName = **new** TextField(" ", 30);

 add(txtFirstName);

 chbox2 = **new** Checkbox("Second");

 add(chbox2);

 lbSecondName = **new** Label("Enter your second name:");

 add(lbSecondName);

 txtSecondName = **new** TextField(" ", 30);

// Поля создаются при помощи конструкторов, а добавляются в окно апплета

// методом add. Согласно схемы расположения компонент, установленой по

// умолчанию, добавляемые компоненты размещаются   сверху вниз и слева

// направо.

 add(txtSecondName);

// Для группы переключателей с зависимой фиксацией мы создаем объект

// класса CheckboxGroup

 grRadio = **new** CheckboxGroup();

 rd1 = **new** Checkbox("Mode 1", grRadio, true);

 rd2 = **new** Checkbox("Mode 2", grRadio, false);

 rd3 = **new** Checkbox("Mode 3", grRadio, false);

 add(rd1); //Переключатели добавляются в окно апплета при помощи метода add

 add(rd2);

 add(rd3);

 ch1 = **new** Choice(); // Список цветов создается как объект класса Choice

 ch1.addItem("White");

 ch1.addItem("Green");

 ch1.addItem("Yellow");

 add(ch1); // Вслед за этим мы добавляем сформированный список в окно апплета

 setBackground(Color.yellow);

// В этом случае метод вызывается для текущего объекта, то есть для нашего

// апплета. Чтобы установить цвет фона в окнах компонент, мы вызываем метод

// setBackground для соответствующих объектов

 lbFirstName.setBackground(Color.yellow);

 lbSecondName.setBackground(Color.yellow);

 rd1.setBackground(Color.yellow);

 rd2.setBackground(Color.yellow);

 rd3.setBackground(Color.yellow);

 chbox1.setBackground(Color.yellow);

 chbox2.setBackground(Color.yellow);

// Многострочное текстовое поле создается как объект класса TextArea.

// В нем 6 строк и 45 столбцов

 txta = **new** TextArea("", 6, 45);

 add(txta);

 txta.setBackground(Color.white);

// последнее что делает метод init перед тем как вернуть управление,

// - создает кнопку с надписью Ready и добавляет ее в окно апплета

 btReady = **new** Button("Ready");

 add(btReady);

 }

 **public String** getAppletInfo()

 {

 return "Name: FormDemo";

 }

 **public void** paint(Graphics g)

 {

 Dimension dimAppWndDimension =

 getSize();

 g.setColor(Color.black);

 g.drawRect(0, 0,

 dimAppWndDimension.width - 1,

 dimAppWndDimension.height - 1);

 }

// В начале своей работы метод action определяет, какой компонент

// вызвал событие. Для этого анализируется поле evt.target:

 **public boolean** action(Event evt, Object obj)

 {

 Button btn;

 **String** str1, str2;

 **if**(evt.target instanceof Button)

 {

// Наш метод action обрабатывает события, вызываемые объектами классов

// Button и Choice. Если событие вызвано компонентом, относящимся к

// какому-либо другому классу, метод возвращает значение false. Этим он

// сигнализирует, что обработка события не выполнялась.

// В случае успешной обработки события метод action возвращает значение true.

// Если событие вызвано кнопкой, наш метод action проверяет, какой именно.

// Обработка выполняется только в том случае, если через поле evt.target

// передается ссылка на кнопку btReady:

 **if**(evt.target.equals(btReady))

 {

// В противном случае метод action возвращает значение false, отказываясь от

// обработки события. Что делает обработчик события, создаваемого кнопкой?

// Прежде всего, он сохраняет ссылку на кнопку в рабочей переменной

// (просто для того чтобы показать, как это делается):

 btn = (Button)evt.target;

//Далее наш обработчик события извлекает текстовые строки из однострочных

//текстовых полей, вызывая для этого метод getText. Эти строки записываются в

//рабочие переменные str1 и str2

 str1 = txtFirstName.getText();

 str2 = txtSecondName.getText();

// Затем метод action проверяет состояние переключателей с независимой

// фиксацией chbox1 и chbox2. Если они включены, содержимое соответствующих

// временных переменных добавляется в многострочное текстовое поле txta:

 **if**(chbox1.getState())

 txta.append(str1);

  **if**(chbox2.getState())

 txta.append(str2);

// Аналогичным образом проверяется состояние переключателей с зависимой

// фиксацией

 **if**(rd1.getState())

 txta.append("\nMode 1\n");

  **if**(rd2.getState())

 txta.append("\nMode 2\n");

 **if**(rd3.getState())

 txta.append("\nMode 3\n");

 }

 **else**

 {

 **return** false;

 }

 **return** true;

 }

 **else if**(evt.target instanceof Choice)

 {

// Если событие вызвано списокм цветов ch1, то метод action определяет, какая

// строка списка стала выделенной и устанавливает в многострочном поле

// редактирования соответствующий цвет фона. Для определения выделенной

// строки применяется метод getSelectedIndex

 **if**(evt.target.equals(ch1))

 {

  **if**(ch1.getSelectedIndex() == 0)

 txta.setBackground(Color.white);

 **if**(ch1.getSelectedIndex() == 1)

 txta.setBackground(Color.green);

 **if(**ch1.getSelectedIndex() == 2)

 txta.setBackground(Color.yellow);

 }

 }

 **return** false;

 }

# 5. Множественные нити выполнения (Multiple threads).

Все современные операционные системы (ОС) работают в мультипрограммном режиме, когда параллельно на одном компьютере выполняются много различных программ. ОС организует переключение между ними, деля время процессора (или процессоров) между различными программами. Все это происходит, так сказать, за сценой. А для пользователя это выглядит как параллельное выполнение нескольких программ одновременно.

Существует два под-варианта реализации мультипрограммного режима работы — при помощи задач (многозадачная или мультизадачная обработка) и при помощи нитей или потоков (мультипотоковая обработка). При этом они могут сосуществовать параллельно. Не вдаваясь в подробности, можно сказать, что мультизадачная обработка более громоздкая и сложная, но имеет больше возможностей. Каждой задаче выделяется свое пространство памяти. В отличии от задач, нити более легковесны и работают все на одном пространстве памяти.

Каждый из этих режимов имеет свое применение. В рамках конкретной ОС существуют те или иные верхнеуровневые механизмы создания задач. Так, запуская программу в Windows, мы тем самым запускаем задачу. Параллельно мы можем запустить другие программы. Для запуска нитей также существуют механизмы операционной системы. Но нити — это параллельные процессы в рамках одной программы. Соответственно, никаких верхнеуровневых (доступных обычному пользователю) средств для запуска нитей нет и быть не может. Эти механизмы могут быть доступны только в рамках той или иной системы программирования.

Термин thread можно перевести и как поток, и как нить. Но в английском есть термин stream, который тоже переводится, как поток и применяется тогда, когда речь идет о потоках ввода/вывода. Поэтому, для однозначности, для thread мы будем использовать перевод нить. В то же время, термин мульти-какая-то-там-нитяная обработка в русском языке сформировать нельзя. Поэтому мы будем использовать мультипотоковая обработка. Неоднозначность здесь не возникает, т.к. к потокам ввода/вывода приставка мульти- обычно не применяется.

В Java реализована полная поддержка нитей. Более того, сама стандартная библиотека Java во многом опирается на мультипотоковую обработку. И многие программы, даже не использующие явно средства организации нитей, тем не менее, работают в мультипотоковой среде. Это, в частности, относится ко всем диалоговым программам. Как пакет AWT, так и пакет Swing существенно используют мультипотоковую обработку.

Простейшим способом создания нити является создание класса производного от класса Thread . В этом классе нужно переопределить метод run , потом породить объект созданного класса и вызвать его метод start . С этого момента начнет свое существование новая нить. Она будет выполняться параллельно с другими нитями, конкурируя с ними за время процессора. Метод start выполняет системные действия по созданию нити, после чего вызывает переопределенный нами метод run . Когда произойдет выход из метода run нить завершит свою работу.

## 5.1. Реализация многопоточности в Java

Для реализации многопоточности должны воспользоваться классом java.lang.Thread. В этом классе определены все методы, необходимые для создания потоков, управления их состоянием и синхронизации.

Как пользоваться классом Thread? Есть две возможности.

Во-первых, вы можете создать свой дочерний класс на базе класса **Thread**. При этом вы должны переопределить метод run. Ваша реализация этого метода будет работать в рамках отдельного потока.

Во-вторых, ваш класс может реализовать интерфейс **Runnable**. При этом в рамках вашего класса необходимо определить метод run, который будет работать как отдельный поток.

Второй способ особенно удобен в тех случаях, когда ваш класс должен быть унаследован от какого-либо другого класса (например, от класса Applet) и при этом вам нужна многопоточность. Так как в языке программирования Java нет множественного наследования, невозможно создать класс, для которого в качестве родительского будут выступать классы Applet и Thread. В этом случае реализация интерфейса Runnable является единственным способом решения задачи.

### 5.1.1. Методы класса Thread

В классе Thread определены три поля, несколько конструкторов и большое количество методов, предназначенных для работы с потоками. Ниже мы привели краткое описание полей, конструкторов и методов.

С помощью конструкторов вы можете создавать потоки различными способами, указывая при необходимости для них имя и группу. Имя предназначено для идентификации потока и является необязательным атрибутом. Что же касается групп, то они предназначены для организации защиты потоков друг от друга в рамках одного приложения.

Методы класса Thread предоставляют все необходимые возможности для управления потоками, в том числе для их синхронизации.

### 5.1.2. Реализация интерфейса Runnable

Описанный выше способ создания потоков как объектов класса Thread или унаследованных от него классов кажется достаточно естественным. Однако этот способ не единственный. Если вам нужно создать только один поток, работающую одновременно с кодом аплета, проще выбрать второй способ с использованием интерфейса Runnable.

Идея заключается в том, что основной класс аплета, который является дочерним по отношению к классу Applet, дополнительно реализует интерфейс Runnable, как это показано ниже:

**public class** MultiTask extends

 **Applet implements** Runnable

{

 Thread m\_MultiTask = null;

 . . .

 **public void** run()

 {

 . . .

 }

 **public void** start()

 {

 **if** (m\_MultiTask == null)

 {

 m\_MultiTask = **new** Thread(this);

 m\_MultiTask.start();

 }

 }

 **public void** stop()

 {

 **if** (m\_MultiTask **!=** null)

 {

 m\_MultiTask.stop();

 m\_MultiTask = null;

 }

 }

}

Внутри класса необходимо определить метод run, который будет выполняться в рамках отдельного потока. При этом можно считать, что код апплета и код метода run работают одновременно как разные потоки.

Для создания потока используется оператор new. Поток создается как объект класса Thread, причем конструктору передается ссылка на класс апплета:

m\_MultiTask **= new** Thread(this);

При этом, когда поток запустится, управление получит метод run, определенный в классе апплета. Как запустить поток? Запуск выполняется, как и раньше, методом start. Обычно поток запускается из метода start апплета, когда пользователь отображает страницу сервера Web, содержащую апплет. Остановка потока выполняется методом stop.

## 5.3. Завершение и останов нити

Как уже указывалось, нормальное завершение нити происходит при выходе из метода run. Кроме того, иногда в приложении требуется выполнить временный останов нити, с тем, чтобы потом возобновить ее работу.

В классе Thread есть методы stop (завершить), suspend (приостановить) и resume (возобновить), но все они объявлены deprecated (устаревшими) и их использование не рекомендовано. О причинах можно почитать подробнее в документации по Java и их здесь рассматривать не будем.

Как же поступить в случае, если нужно приостановить выполнение процесса? Для этого нужно не останавливать процесс, а обеспечить такой алгоритм, при котором он "работает в холостую".

## 5.4. Приоритеты нитей

Если процесс создал несколько потоков, то все они выполняются параллельно, причем время центрального процессора (или нескольких центральных процессоров в мультипроцессорных системах) распределяется между этими потоками.

Распределением времени центрального процессора занимается специальный модуль операционной системы - планировщик. Планировщик по очереди передает управление отдельным потокам, так что даже в однопроцессорной системе создается полная иллюзия параллельной работы запущенных потоков.

Распределение времени выполняется по прерываниям системного таймера. Поэтому каждому потоку дается определенный интервал времени, в течении которого он находится в активном состоянии.

Заметим, что распределение времени выполняется для потоков, а не для процессов. Потоки, созданные разными процессами, конкурируют между собой за получение процессорного времени.

Для оптимизации параллельной работы нитей в Java имеется возможность устанавливать приоритеты нитей. Нити с большим приоритетом имеют преимущество в получении времени процессора перед нитями с более низким приоритетом.

Работа с приоритетами обеспечивается следующими методами класса Thread:

**public final void** setPriority(int newPriority)

Устанавливает приоритет нити.

**public final int** getPriority()

Позволяет узнать приоритет нити.

Значение параметра в методе setPriority не может произвольным. Оно должно находиться в пределах от MIN\_PRIORITY до MAX\_PRIORITY. При своем создании нить имеет приоритет NORM\_PRIORITY.

## 5.5. Средства синхронизации нитей в Java

Проанализируем эту проблему с другой точки зрения. У нас есть некоторый ресурс, в данном случае - случайное число (переменная randValue), доступ к которому нужно упорядочить. Т.е. нужно заблокировать доступ к этому ресурсу для всех нитей, кроме одной, пока эта нить не выполнит над ним необходимые действия.

В Java есть возможности по синхронизации нитей, построенные на этом принципе. Т.е. определяется некоторый ресурс, который может быть заблокирован. Таким ресурсом может быть любой объект (но не данное элементарного типа). Далее определяются критические участки программы. При входе в такой участок, ресурс блокируется и становится недоступным для всех других нитей (с некоторой оговоркой, которую мы рассмотрим позже). При выходе из критического участка выполняется разблокировка ресурса.

В качестве критического участка программы в Java может быть определен некоторый нестатический метод или блок. При вызове метода блокируется объект, для которого вызван данный метод (this). Для блока блокируемый объект указывается явно. Синтаксис определения критических участков следующий. Для метода мы просто при описании метода указываем описатель synchronized, например,

 **public void** synchronized f() {

 ...

 }

Для блока мы непосредственно перед блоком ставим конструкцию synchronized (объект), где объект - это ссылка на блокируемый объект. Например,

 synchronized(ref) {

 ...

 }

Здесь в качестве критического участка определяется блок, а в качестве блокируемого объекта - объект, на который ссылается ref.

## 5.6. Применение многопоточности для анимации

Одно из наиболее распространенных применений апплетов - это создание анимационных эффектов типа бегущей строки, мерцающих огней или аналогичных, привлекающих внимание пользователя. Для того чтобы достичь такого эффекта, необходим какой либо механизм, позволяющий выполнять перерисовку всего окна апплета или его части периодически с заданным временным интервалом.

Работа апплетов, так же как и обычных приложений операционной системы Windows, основана на обработке событий. Для классического приложения Windows событие - это приход сообщения в функцию окна. Основной класс апплета обрабатывает события, переопределяя те или иные методы базового класса Applet.

Проблема с периодическим обновлением окна апплета возникает из-за того, что в языке Java не предусмотрено никакого механизма для создания генератора событий, способного вызывать какой-либо метод класса апплета с заданным интервалом времени. Вы не можете поступить так, как поступали в этой ситуации, разрабатывая обычные приложения Windows - создать таймер и организовать обработку периодически поступающих от него сообщений WM\_TIMER.

Напомним, что перерисовка окна апплета выполняется методом paint, который вызывается виртуальной машиной Java асинхронно по отношению к выполнению другого кода апплета.

Можно ли воспользоваться методом paint для периодической перерисовки окна аплета, организовав в нем, например, бесконечный цикл с задержкой?

К сожалению, так поступать ни в коем случае нельзя. Метод paint после перерисовки окна апплета должен сразу возвратить управление, иначе работа апплета будет заблокирована.

Единственный выход из создавшейся ситуации - создание потока (или нескольких потоков), которые будут выполнять рисование в окне апплета асинхронно по отношению к работе кода апплета. Например, вы можете создать поток, который периодически обновляет окно апплета, вызывая для этого метод repaint, или рисовать из потока непосредственно в окне апплета, получив предварительно для этого окна контекст отображения.

## 5.7. Цель, требования и рекомендации к выполнению задания

Цель выполнения задания ознакомление с основными принципами многопоточности в языке Java, возможностью написания программ с использованием нитей.

Требования и рекомендации к выполнению задания:

1. проанализировать полученное задание, выделить информационные объекты и действия;
2. разработать программу с использованием нитей.

## 5.8. Задания

1. Создать апплет, используя поток: строка движется горизонтально, отражаясь от границ апплета и меняя при этом случайным образом свой цвет.
2. Создать апплет, используя поток: строка движется по диагонали. При достижении границ апплета все символы строки случайным образом меняют регистр.
3. Организовать сортировку массива методами Шелла, Хоара, пузырька, на основе бинарного дерева в разных потоках.
4. Реализовать сортировку графических объектов, используя алгоритмы из задания 3.4.
5. Создать апплет с точкой, движущейся по окружности с постоянной угловой скоростью. Сворачивание браузера должно приводить к изменению угловой скорости движения точки для следующего запуска потока.
6. Изобразить точку, пересекающую с постоянной скоростью окно слева направо (справа налево) параллельно горизонтальной оси. Как только точка доходит до границы окна, в этот момент от левого (правого) края с вертикальной координатной у, выбранной с помощью датчика случайных чисел, начинает свое движение другая точка и т.д. Цвет точки также можно выбирать с помощью датчика случайных чисел. Для каждой точки создается собственный поток.
7. Изобразить в приложении правильные треугольники, вращающиеся в плоскости экрана вокруг своего центра. Каждому объекту соответствует поток с заданным приоритетом.
8. Условия предыдущих задач изменяются таким образом, что центр вращения перемещается от одного края окна до другого с постоянной скоростью параллельно горизонтальной оси.
9. Создать фрейм с тремя шариками, одновременно летающими в окне. С каждым шариком связан свой поток со своим приоритетом.
10. Два изображения выводятся в окно. Затем они постепенно исчезают с различной скоростью в различных потоках (случайным образом выбираются точки изображения, и их цвет устанавливается в цвет фона).
11. Условие предыдущей задачи изменить на применение эффекта постепенного "проявления" двух изображений.
12. Создать апплет «Бегущая строка»
13. Реализовать приложение, в котором пользователь имеет возможность указывать маски файлов для поиска и набор путей, по которым эти файлы нужно искать (например, список логических дисков).
14. Разработать аплет, реализующий игру, который должен содержать следующие элементы:
- основа пользовательского интерфейса — графические объекты, созданные на основе пакетов java.awt, java.awt.geom и классов java.awt.Graphics и java.awt.Graphics2D.
- не менее двух движущихся объектов, созданных на основе тех же пакетов и классов. Каждый объект должен управляется своим потоком, что должно быть визуально заметно (например, с помощью задания разных значений в соответствующих методах Thread.sleep()).
Управление игрой должно осуществляться с клавиатуры и/или мышью. Если используется только мышь, то нельзя использовать курсор, заданный по умолчанию, (вместо него надо использовать любой другой, подходящий по смыслу, или создать свой).
дополнительно– в игре может вестись подсчет очков.
В качестве возможного варианта игры м.б., например, «тир» или «футбол».
15. Написать секундомер - класс Stopwatch - для замера времени в отдельном потоке выполнения. В классе должны быть реализованы следующие методы
* start – начинает отчет времени
* stop – прерывает отчет времени
* reset – сбрасывает текущее значение секундомера
* getTime – возвращает отсчитанное время в миллисекундах

Для демонстрации работы секундомера написать консольное приложение. Пользователю должны быть доступными следующие команды:

* start N – запустить секундомер и дать ему идентификатор N
* stop N – остановить секундомер с идентификатором N
* reset N – сбросить время у секундомера с идентификатором N
* time N – показать время у секундомера с идентификатором N
* help – список команд
* exit – выход

 16. Подсчет простых чисел. Написать Swing-приложение для подсчета простых чисел в параллельных потоках. Каждый поток подсчитывает простые числа из заданного диапазона. Всего должно быть запущено три потока подсчета. Полученные числа выдаются на экран. Можно запускать/останавливать подсчет несколько раз.

Пользователю доступны следующие элементы управления:

* Кнопка запуска всех потоков
* Кнопка остановки всех потоков
* Поле (JTextArea) для вывода результата в виде:

<номер потока>: <число>

Например:

1: 11

2: 23

2: 31

1: 17

4 поля ввода диапазонов чисел для подсчета. Получается 3 диапазона (между 4-мя числами) для каждого потока.

Три строки с информацией о состоянии каждого потока

## 5.9. Пример выполнения задания.

Напишем апплет Rectangles (рис. 6). Он создает три потока. Первый поток рисует в окне аплета прямоугольники случайного размера и цвета, второй - эллипсы, а третий управляет потоком рисования эллипсов.



Рис. 6. Окно аплета Rectangles

Расположение прямоугольников и эллипсов также выбирается случайно.

В этом приложении мы создаем на базе класса Thread три класса. Первый из них предназначен для создания потока рисования прямоугольников, второй - для создания потока рисования закрашенных эллипсов, а третий - для управления потоком рисования эллипсов.

Что же касается основного класса аплета, то он унаследован, как обычно, от класса Applet и не реализует интерфейс Runnable:

import java.applet.\*;

import java.awt.\*;

**public class** Rectangles extends Applet

{

 DrawRectangles m\_DrawRectThread = null;

 DrawEllipse m\_DrawEllipseThread = null;

 NotifyTask m\_NotifyTaskThread = null

// Эти поля являются ссылками на классы, соответственно DrawRectangles,

// DrawEllipse и NotifyTask . Первый из них создан для рисования

// прямоугольников, второй - эллипсов, а третий - для управления потоком

 // рисования эллипсов. Указанные поля инициализируются значением null,

// что соответствует неработающим или не созданным задачам.

 **public String** getAppletInfo()

 {

 **return** "Name: Rectangles";

 }

 **public void** paint(Graphics g)

 {

 Dimension dimAppWndDimension = getSize();

 g.setColor(Color.yellow);

 g.fillRect(0, 0,

 dimAppWndDimension.width - 1,

 dimAppWndDimension.height - 1);

 g.setColor(Color.black);

 g.drawRect(0, 0,

 dimAppWndDimension.width - 1,

 dimAppWndDimension.height - 1);

 }

 **public void** start()

 {

// Метод start класса Rectangles Этот метод последовательно создает

// три потока и запускает их на выполнение

 **if** (m\_DrawRectThread == null)

 {

 m\_DrawRectThread =

 new DrawRectangles(this);

 m\_DrawRectThread.start(); // 1-ый поток

 }

 **if** (m\_DrawEllipseThread == null)

 {

 m\_DrawEllipseThread =

 **new** DrawEllipse(**this**);

 m\_DrawEllipseThread.start(); // 2-ой поток

 }

 **if** (m\_NotifyTaskThread == null)

 {

 m\_NotifyTaskThread =

 new NotifyTask(m\_DrawEllipseThread);

 m\_NotifyTaskThread.start();// 3-ий поток

 }

 }

// Когда пользователь покидает страницу сервера Web с апплетом, метод

// **stop** класса Rectangles последовательно останавливает рисования

// прямоугольников и эллипсов, а также управляющий поток:

 **public void** stop()

 {

 **if** (m\_DrawRectThread != null)

 {

 m\_DrawRectThread.stop();

 m\_DrawRectThread = null;

 }

 **if** (m\_DrawEllipseThread == null)

 {

 m\_DrawEllipseThread.stop();

 m\_DrawEllipseThread = null;

 }

 **if** (m\_NotifyTaskThread != null)

 {

 m\_NotifyTaskThread.stop();

 m\_NotifyTaskThread = null;

 }

 }

}

// Класс DrawRectangles определен для потока рисования прямоугольников

// В поле g класса хранится контекст отображения окна апплета, а в поле

// dimAppWndDimension – размеры этого окна: Значения этих полей

// определяются конструктором класса по ссылке на главный класс апплета.

**class** DrawRectangles **extends** Thread

{

 Graphics g;

 Dimension dimAppWndDimension;

 public DrawRectangles(Applet Appl)

 {

 g = Appl.getGraphics(); // В качестве параметра конструктору передается

 // ссылка на класс апплета

 dimAppWndDimension = Appl.getSize();

//Конструктор использует эту ссылку для получения и сохранения в полях

// класса контекста отображения и размеров окна апплета

 }

// Программный код метода run работает в рамках отдельного потока.

// Он рисует в окне апплета закрашенные прямоугольники. Прямоугольники

// имеют случайные координаты, расположение и цвет. Для того чтобы

// рисовать, необходимо получить контекст отображения. Этот контекст был

// получен конструктором класса DrawRectangles и может быть использован

// методом run. // Вооружившись контекстом отображения и размерами окна

// апплета, поток входит в бесконечный цикл рисования прямоугольников.

// В качестве генератора случайных чисел мы используем метод random из

// класса Math, который при каждом вызове возвращает новое случайное

// число типа double, лежащее в диапазоне значений от 0.0 до 1.0.

// Координаты по осям X и Y рисуемого прямоугольника определяются

// простым умножением случайного числа, полученного от метода random,

// соответственно, на ширину и высоту окна апплета:

 **public void** run()

 {

 **while** (true)

 {

 **int** x, y, width, height;

 **int** rColor, gColor, bColor;

 x = (int)(dimAppWndDimension.width \* Math.random());

 y = (int)(dimAppWndDimension.height \* Math.random());

 width = (int)(dimAppWndDimension.width \* Math.random()) / 2;

 height = (int)(dimAppWndDimension.height \* Math.random()) / 2;

 rColor = (int)(255 \* Math.random()); // Для случайного выбора цвета

 gColor = (int)(255 \* Math.random());// прямоугольника вычисляем

 bColor = (int)(255 \* Math.random()); // отдельные цветовые компоненты,

 // умножая значение, полученное от метода random, на 255

 g.setColor(new Color(rColor, gColor, bColor)); //цвет устанавливается в

 // контексте отображения методом setColor

 g.fillRect(x, y, width, height);

 **try** // После рисования прямоугольника метод run задерживает свою

 // работу на 50 миллисекунд, вызывая метод sleep:

 {

 Thread.sleep(50);

 }

 **catch** (InterruptedException e)

 {

 stop();

 } // Для обработки исключения **InterruptedException**, которое может

 } // возникнуть во время работы этого метода, предусмотриv блок.

 } // try – catch. При возникновении указанного исключения работа

} // потока останавливается вызовом метода **stop**

**class** DrawEllipse **extends** Thread

{ // Класс DrawEllipse очень похож на только что

 Graphics g; // рассмотренный класс DrawRectangles

 Dimension dimAppWndDimension;

 **public** DrawEllipse(Applet Appl)

 {

 g = Appl.getGraphics();

 dimAppWndDimension = Appl.getSize();

 }

 **public** synchronized **void** run()

 {

 **while** (true)

 {

 **int** x, y, width, height;

 **int** rColor, gColor, bColor;

 x = (int)(dimAppWndDimension.width \* Math.random());

 y = (int)(dimAppWndDimension.height \* Math.random());

 width = (int)(dimAppWndDimension.width \* Math.random()) / 2;

 height = (int)(dimAppWndDimension.height \* Math.random()) / 2;

 rColor = (int)(255 \* Math.random());

 gColor = (int)(255 \* Math.random());

 bColor = (int)(255 \* Math.random());

 g.setColor(new Color(rColor, gColor, bColor));

 g.fillOval(x, y, width, height);

 **try**

 {

 this.wait();

 }

 **catch** (InterruptedException e)

 {

 }

 }

 }

}

**class** NotifyTask **extends** Thread

{ // В классе NotifyTask мы определили одно поле STask класса Thread.

 // Это поле которое хранит ссылку на поток, работой которого управляет

 // данный класс:

 Thread STask;

 **public** NotifyTask(Thread SynchroTask)

 {

 STask = SynchroTask;

 }

// Метод **run** класса **NotifyTask** периодически разблокирует поток рисования

// эллипсов, вызывая для этого метод notify в цилке с задержкой 30

// миллисекунд. Обращение к объекту STask, который хранит ссылку

// на поток рисования эллипсов, выполняется с использованием синхронизации:

 **public void** run()

 {

 **while** (true)

 {

 **try**

 {

 Thread.sleep(30);

 }

 **catch** (InterruptedException e)

 {

 }

 synchronized(STask)

 {

 STask.notify();

 } } } }

#### Дополнительная информация и список литературы

1. Программирование на Java: подробное руководство
<http://ru.sun.com/java/books/online/index.html>
2. Примеры программ <http://ru.sun.com/java/start/examples/examples.html>
3. Sun Microsystems. <http://www.sun.ru/java>.
4. "Thinking in Java", Bruce Eckel. <http://www.bruceeckel.com/>.
5. Документация от Sun <http://java.sun.com/products/jdk/1.3/docs/index.html>.
6. Краткое описание JAVA
<http://java.sun.com/products/jdk/1.3/devdocs-vs-specs.html>.
7. Обучающие курсы для разработчиков <http://developer.java.sun.com/developer/onlineTraining/>.
8. Русскоязычный сайт по Java <http://www.javable.com/>.
9. Популярный англоязычный сайт <http://www.javaworld.com/>.
10. Java FAQ (на русском) <http://www.sun.ru/java/start/questions/faq/faq.html>Фролов
11. . А.В., Фролов Г.В. "Создание приложений Java" <http://www.sun.ru/java/books/online/index.html> или <http://athena.vvsu.ru/docs/c-java/java_f/>.
12. Swing'у: Swing by Matthew Robinson and Pavel Vorobiev. <http://manning.spindoczine.com/sbe/>.
13. Брюс Эккель. Философия JAVA. Библиотека программиста. – СПб:Питер, 2001.- 880 с.: с ил.
14. Ноутон П., Шилдт Г. JAVA 2. СПб : БХВ-Петербург, 2000.
15. Арнольд, Кен, Гослинг, Джеймс, Холмс, Дэвид. Язык программирования JAVA.3-е изд. Пер. с англ..- М. Издательский дом Вильямс, 2001. -624 с.