**Т.В. ГЕРАСИМОВА**

Программирование на языке JAVA

Лабораторный практикум.

Санкт-Петербург

2014

Содержит основные сведение, приемы и упражнения по программированию на объектно-ориентированном языке JAVA. Практикум ориентирован на использование при проведении практических занятий, лабораторных работ по дисциплине “Программирование на языке JAVA”.

**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.



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.** [**Структуры данных. Коллекции Java**](http://www.4stud.info/java-programming/collections.html)

**2.1. Цель работы**

При программировании на Java операций над группой однотипных объектов важно выбирать наиболее эффективную структуру данных (класс) для хранения этих объектов. В языке java определены специальные классы для хранения однотипных объектов, которые называются коллекциями, определяющими такие структуры как список, множество, очередь. В лабораторной работе рассматриваются способы использования коллекций при разработке java-приложений.

**2.2. Указания к работе**

Выбор определенного класса для работы с коллекциями определяет набор методов, которые будут доступны для объекта этого класса. Например, если используется список (который определяет интерфейс List), то существуют богатый выбор для его реализации: ArrayList, LinkedList, Vector, Stack. Конкретный выбор реализации списка сказывается на эффективности манипуляций с объектами списка. Так, ArrayList хранит элементы в виде массива, а значит, доступ и замена будет выполняться относительно быстро. В то же время LinkedList хранит элементы в виде связного списка, что влечет за собой относительно медленный поиск элементов и быструю операцию добавления/удаления. Рекомендуется ознакомиться с описаниями следующих коллекций по JSDK -документации:

– java.util.Collection ;

– java.util.ArrayList;

– java.util.HashMap;

– java.util.HashSet.

Важно также знание основных классов для работы с коллекциями. Особое внимание при изучении коллекций заслуживают классы Comporator, Collections, Iterator.

Ниже приведен краткий разбор наиболее важных классов при работе с коллекциями, а также решения одного из индивидуальных заданий.

**Интерфейс Collection**

Интерфейс Collection содержит набор общих методов, которые используются в большинстве коллекций. Рассмотрим основные из них:

* add(Object item) – добавляет в коллекцию новый элемент, если элементы коллекции каким-то образом упорядочены, новый элемент добавляется в конец коллекции;
* clear() – удаляет все элементы коллекции;
* contains(Object obj) – возвращает true, если объект obj содержится в коллекции и false, если нет;
* isEmpty() – проверяет, пуста ли коллекция;
* remove(Object obj) – удаляет из коллекции элемент obj, возвращает false, если такого элемента в коллекции не нашлось;
* size() – возвращает количество элементов коллекции.

**Интерфейс List**

Интерфейс List описывает упорядоченный список. Элементы списка пронумерованы, начиная с нуля и к конкретному элементу можно обратиться по целочисленному индексу. Интерфейс List является наследником интерфейса Collection, поэтому содержит все его методы и добавляет к ним несколько своих:

* add(int index, Object item) – вставляет элемент item в позицию index, при этом список раздвигается (все элементы, начиная с позиции index, увеличивают свой индекс на 1);
* get(int index) – возвращает объект, находящийся в позиции index;
* indexOf(Object obj) – возвращает индекс первого появления элемента obj в списке;
* lastIndexOf(Object obj) – возвращает индекс последнего появления элемента obj в списке;
* add(int index, Object item) – заменяет элемент, находящийся в позиции index объектом item;
* subList(int from, int to) – возвращает новый список, представляющий собой часть данного (начиная с позиции from до позиции to-1 включительно).

**Интерфейс Set**

Интерфейс Set описывает множество. Элементы множества не упорядочены, множество не может содержать двух одинаковых элементов. Интерфейс Set унаследован от интерфейса Collection, но никаких новых методов не добавляет. Изменяется только смысл метода add(Object item) – он не добавляет объект item, если он уже присутствует во множестве.

**Интерфейс Queue**

Интерфейс Queue описывает очередь. Элементы могут добавляться в очередь только с одного конца, а извлекаться с другого (аналогично очереди в магазине). Интерфейс Queue так же унаследован от интерфейса Collection. Специфические для очереди методы:

* poll() – возвращает первый элемент и удаляет его из очереди.

Методы интерфейса Queue:

* peek() – возвращает первый элемент очереди, не удаляя его.
* offer(Object obj) – добавляет в конец очереди новый элемент и возвращает true, если вставка удалась.

**Класс Vector**

Vector (вектор) – набор упорядоченных элементов, к каждому из которых можно обратиться по индексу. По сути эта коллекция представляет собой обычный список.

Класс Vector реализует интерфейс List, основные методы которого названы выше. К этим методам добавляется еще несколько. Например, метод firstElement() позволяет обратиться к первому элементу вектора, метод lastElement() – к его последнему элементу. Метод removeElementAt(int pos) удаляет элемент в заданной позиции, а метод removeRange(int begin, int end) удаляет несколько подряд идущих элементов. Все эти операции можно было бы осуществить комбинацией базовых методов интерфейса List, так что функциональность принципиально не меняется.

**Класс ArrayList**

Класс ArrayList – аналог класса Vector. Он представляет собой список и может использоваться в тех же ситуациях. Основное отличие в том, что он не синхронизирован и одновременная работа нескольких параллельных процессов с объектом этого класса не рекомендуется. В обычных же ситуациях он работает быстрее.

**Класс Stack**

Stack – коллекция, объединяющая элементы в стек. Стек работает по принципу LIFO (последним пришел – первым ушел). Элементы кладутся в стек «друг на друга», причем взять можно только «верхний» элемент, т.е. тот, который был положен в стек последним. Для стека характерны операции, реализованные в следующих методах класса Stack:

* push(Object item) – помещает элемент на вершину стека;
* pop() – извлекает из стека верхний элемент;
* peek() – возвращает верхний элемент, не извлекая его из стека;
* empty() – проверяет, не пуст ли стек;
* search(Object item) – ищет «глубину» объекта в стеке. Верхний элемент имеет позицию 1, находящийся под ним – 2 и т.д. Если объекта в стеке нет, возвращает –1.

Класс Stack является наследником класса Vector, поэтому имеет все его методы (и, разумеется, реализует интерфейс List). Однако если в программе нужно моделировать именно стек, рекомендуется использовать только пять вышеперечисленных методов.

**Интерфейс Iterator**

Преимущество использования массивов и коллекций заключается не только в том, что можно поместить в них произвольное количество объектов и извлекать их при необходимости, но и в том, что все эти объекты можно комплексно обрабатывать. Например, вывести на экран все шашки, содержащиеся в списке checkers. В случае массива мы пользуемся циклом:

for (int i = 1; i < array.length; i++){

// обрабатываем элемент array[i]

}

Имея дело со списком, мы можем поступить аналогичным образом, только вместо array[i] писать array.get(i). Но мы не можем поступить так с коллекциями, элементы которых не индексируются (например, очередью или множеством). А в случае индексированной коллекции надо хорошо знать особенности ее работы: как определить количество элементов, как обратиться к элементу по индексу, может ли коллекция быть разреженной (т.е. могут ли существовать индексы, с которыми не связано никаких элементов) и т.д.

Для навигации по коллекциям в Java предусмотрено специальное архитектурное решение, получившее свою реализацию в интерфейсе Iterator. Идея заключается в том, что к коллекции «привязывается» объект, единственное назначение которого — выдать все элементы этой коллекции в некотором порядке, не раскрывая ее внутреннюю структуру.

Интерфейс Iterator имеет всего три метода:

* next() – возвращает очередной элемент коллекции, к которой «привязан» итератор (и делает его текущим). Порядок перебора определяет сам итератор.
* hasNext() – возвращает true, если перебор элементов еще не закончен
* remove() – удаляет текущий элемент.

Интерфейс Collection помимо рассмотренных ранее методов, имеет метод iterator(), который возвращает итератор для данной коллекции, готовый к ее обходу. С помощью такого итератора можно обработать все элементы любой коллекции следующим простым способом:

Iterator iter = coll.iterator(); // coll - коллекция

while (iter.hasNext()) {

// обрабатываем объект, возвращаемый методом iter.next()

}

Для коллекций, элементы которых проиндексированы, определен более функциональный итератор, позволяющий двигаться как в прямом, так и в обратном направлении, а также добавлять в коллекцию элементы. Такой итератор имеет интрефейс ListIterator, унаследованный от интерфейса Iterator и дополняющий его следующими методами:

* previous() – возвращает предыдущий элемент (и делает его текущим);
* hasPrevious() – возвращает true, если предыдущий элемент существует (т.е. текущий элемент не является первым элементом для данного итератора);
* add(Object item) – добавляет новый элемент перед текущим элементом;
* set(Object item) – заменяет текущий элемент;
* nextIndex() и previousIndex() – служат для получения индексов следующего и предыдущего элементов соответственно.

В интерфейсе List определен метод listIterator(), возвращающий итератор ListIterator для обхода данного списка.

**Интерфейс Map**

Интерфейс Map из пакета java.util описывает коллекцию, состоящую из пар «ключ – значение», которые широко используются для хранения настроек в файлах конфигурации (см., например, [/etc/services](http://www.4stud.info/networking/work3.html)). У каждого ключа только одно значение, что соответствует математическому понятию однозначной функции или отображения (Mар). Интерфейс Map содержит следующие методы, работающие с ключами и значениями:

* boolean containsKey (Object key) — проверяет наличие ключа key;
* boolean containsValue (Object value) — проверяет наличие значения value;
* Set entry Set() – представляет коллекцию в виде множества, каждый элемент которого – пара из данного отображения, с которой можно работать методами вложенного интерфейса Map.Entry;
* Оbject get(Object key) – возвращает значение, отвечающее ключу key; S et key S et() – представляет ключи коллекции в виде множества;
* Object put (Object key, Object value ) — добавляет пару « key — value », если такой пары не было, и заменяет значение ключа key, если такой ключ уже есть в коллекции;
* void putAll (Map m) – добавляет к коллекции все пары из отображения m;
* collection values () – представляет все значения в виде коллекции.

В интерфейс Mар вложен интерфейс Map.Entry, содержащий методы работы с отдельной парой.

**Вложенный интерфейс Map.Entry**

Этот интерфейс описывает методы работы с парами, полученными методом entrySet() из объекта типа Map.

Методы getKey() и getValue() позволяют получить ключ и значение пары, метод setValue (Оbject value) меняет значение в данной паре.

**2.3. Задания к лабораторной работе**

1. Ввести строки из файла, записать их в стек. Вывести строки в файл в обратном порядке.
2. Ввести число, занести его цифры в стек. Вывести в число, у которого цифры идут в обратном порядке.
3. Сложить два многочлена заданной степени, если коэффициенты многочленов хранятся в объекте HashMap.
4. Создать стек из элементов каталога.
5. Не используя вспомогательных объектов, переставить отрицательные элементы данного списка в конец, а положительные - в начало этого списка.
6. Организовать вычисления в виде стека.
7. Выполнить попарное суммирование произвольного конечного ряда чисел следующим образом: на первом этапе суммируются попарно рядом стоящие числа, на втором этапе суммируются результаты первого этапа и т.д. до тех пор, пока не останется одно число.
8. Задать два стека, поменять информацию местами.
9. Определить класс Stack. Объявить объект класса. Ввести последовательность символов и вывести ее в обратном порядке.
10. Умножить два многочлена заданной степени, если коэффициенты многочленов хранятся в списках.
11. Определить класс Set на основе множества целых чисел, n = размер. Создать методы для определения пересечения и объединения множеств.
12. Программа получает N параметров вызова (аргументы командной строки). Эти параметры – элементы вектора. Строится массив типа double, а на базе этого массива – объект класса DoubleVector. Далее программа выводит в консоль значения элементов вектора в виде: Вектор: 2.3 5.0 7.3.
13. Списки (стеки) I(1..N) и U(1..N) содержат результаты N измерений тока и напряжения на неизвестном сопротивлении R. Найти приближённое число R методом наименьших квадратов.

**Пример выполнения задания**

Вычислить сколько раз каждая буква встречается в тексте.

Решение:

import java.util.HashMap;

import java.util.\*;

public class Main {

public static void main(String[] args) {

String txt = " лабораторная работа " ;

HashMap<Character, Integer> map = new HashMap<Character, Integer>(40);

for (int i = 0; i < txt.length(); ++i) {

char c = txt.charAt(i);

//проверяем является ли символ буквой

if (Character.isLetter(c)) {

if (map.containsKey(c)) {

map.put(c, map.get(c) + 1);

} else {

map.put(c, 1);

}

}

}

//вывод на экран букв с частотой их появления

for (Entry<Character, Integer> entry : map.entrySet()) {

System.out.println("буква: "+entry.getKey()+" кол - во: "+entry.getValue());

}

}

}