Факультет "Информатика и системы управления"

Методические указания к лабораторной работе

по курсу "Распределенные системы обработки информации"

Основные структуры языка Java

Москва 2004 г.

**Оглавление**

Цель работы 3

Задание для домашней подготовки 3

Задания к лабораторной работе 3

Задание 1 3

Задание 2 5

Задание 3 5

Содержание отчета 6

Контрольные вопросы. 7

Литература. 8

Приложение 1. Встроенные типы данных и основные конструкции языка. 9

Приложение 2. Работа с массивами 26

Приложение 3. Введение в интегрированную среду разработки Eclipse 28

Приложение 4. Листинг программы, подсчитывающей , как долго нужно откладывать деньги, чтобы к выходу на пенсию накопить заданную сумму, при условии, что ежегодно откладывается одна и та же сумма и размер счета возрастает в соответствии с процентной ставкойJava 43

Приложение 5. Листинг программы, иллюстрирующей работу с одномерными массивами (создание и сортировка массива) 45

Приложение 6. Спецификация класса Figure. 47

**Цель работы**

Получить знания о выражениях и операторах языка Java. Научиться писать программы, производящие простые вычисления и поддерживающие диалог с пользователем, разрабатывать программы, осуществляющие обработку одномерных массивов и строк.

# 

# Задание для домашней подготовки

Ознакомиться с материалом, предоставленном в приложении к данным методическим указаниям. Ознакомиться с текстом задания к лабораторной работе в соответствии с вариантом и написать программу.

# 

# Задания к лабораторной работе

# Задание 1

Номер задания соответствует порядковому номеру в журнале (по модулю 14).

1. Написать программу, которая просит ввести два числа, получает эти два числа от пользователя, а затем печатает сумму, разность, произведение и частное этих чисел.
2. Написать программу, запрашивающую у пользователя три числа и печатающую сумму, произведение, среднее значение, меньшее и большее их этих чисел.
3. Написать программу, принимающую от пользователя радиус круга и печатающую диаметр круга, длину окружности и площадь круга.
4. Написать программу, которая считывает два целых числа, определяет и печатает, является ли первое число кратным второму.
5. Написать программу, которая определяет, является введенное число четным или нечетным.
6. Написать программу, которая вычисляет и печатает квадрат и куб чисел от 1 до 10.
7. Написать программу, которая печатает геометрическую фигуру: прямоугольник, ромб, стрелу:

\*\*\*\*\*\*\* \* \*

\* \* \* \* \*\*

\* \* \* \* \*\*\*

\* \* \* \* \*

\*\*\*\*\*\*\* \* \*

\*

1. Написать программу, которая читает размер стороны квадрата и затем печатает звездочками и пробелами пустой квадрат заданного размера.
2. Написать программу, которая читает неотрицательное целое, рассчитывает и печатает его факториал.
3. Написать программу, которая суммирует последовательность целых чисел. Предполагается, что первое прочитанное целое число указывает количество целых чисел, которые будут введены, причем программа должна читать только по одному значению в операторе ввода.
4. Написать программу, которая подсчитывает и печатает среднее значение нескольких целых чисел. Предполагается, что последняя читаемая величина имеет значение 9999.
5. Написать программу, которая находит наименьшее и наибольшее из нескольких чисел. Предполагается, что первое прочитанное число задает количество последующих вводимых чисел.
6. Написать программу, которая считает и печатает произведение нечетных целых чисел от 1 до 15.
7. Написать программу, которая печатает трафареты (А, В, С, D – см. ниже) один под другим. Необходимо использовать цикл for для генерации трафаретов.

\* \*\*\* \*\*\* \*

\*\* \*\* \*\* \*\*

\*\*\* \* \* \*\*\*

А В С D

# Задание 2

Одномерные массивы [Л.2 на с.67-70], [Л.3 на с.136-139]. Номер задания соответствует порядковому номеру в журнале (по модулю 20).

# Задание 3

Написать программу, которая выводит на консоль геометрическую фигуру: квадрат, прямоугольник или стрелу (в соответствии с вариантом задания):

1)

\*\*\*\*\*\*\*

\* \*

\* \*

\* \*

\*\*\*\*\*\*\*

2)

S S S S

S $ $ S

S $ $ S

S S S S

3)

0 0 0 0 0

0 0

0 0

0 0

0 0

Длина стороны квадрата (сторон прямоугольника / длина стрелки) должны задаваться либо в параметрах командной строки, либо с клавиатуры через консоль после запуска программы.

Символы – заполнители (“\*”,“ ”…) также должны либо задаваться в параметрах командной строки, либо назначаться по умолчанию в случае, если программа запущена без параметров. То есть программа при запуске должна определять запущена ли она с параметрами. Если нет – то она должна просить пользователя ввести недостающие размеры с клавиатуры, а символы – заполнители назначаться по умолчанию.

Программа должна быть реализована внутри метода

public static void main(String[] args){

//…

}

с использованием встроенных типов и основных конструкций языка Java.

Также должна быть предусмотрена обработка исключений.

Факультативно (не обязательно): Организовать вывод фигуры в текстовый файл.

В ходе лабораторной работы необходимо отладить написанную дома программу и предъявить работающую программу преподавателю.

# 

# Содержание отчета

Отчет должен содержать:

1. Постановку задачи, решаемой отлаженной программой.
2. Руководство пользователя отлаженной программы, содержащее описание интерфейсов всех функций программы.
3. Листинг программы с необходимыми комментариями.

# 

# Контрольные вопросы

* + - 1. Какие условные операторы есть в языке Java?
      2. Какие операторы цикла существуют в языке Java?
      3. В чем отличие между операторами цикла for, while и do-while? Какой синтаксис у этих операторов?
      4. Как реализовать оператор goto c помощью стандартных операторов Java?
      5. Почему в блоке switch(){} оператор default ставится в конце.
      6. Как уменьшить размер блока оператора switch(){} в данной работе.
      7. Какие операции существуют в языке Java?
      8. Для чего необходимы операторы условного перехода? Какие операторы условного перехода есть в языке Java?

# 

# Литература

1. Арнолд К., Гослинг Дж., Холмс Д. Язык программирования Java:Пер. с англ. – М.: Издательский дом «Вильямс», 2001 г. – 624 с., ил.
2. Павловская Т.А., Щупак Ю.А. С/С++. Структурное программирование: Практикум. -СПб.: Питер, 2002. -240с.
3. Павловская Т.А. С/С++. Программирование на языке высокого уровня -СПб.: Питер, 2002. -464с.

Дополнительная

1. Официальный сайт Java — http://java.sun.com/ (есть раздел на русском языке с учебником).
2. Java™ 2 SDK, Standard Edition Documentation — http://java.sun.com/products/jdk/1.5/index.html.
3. Джеймс Гослинг, Билл Джой, Гай Стил. Спецификация языка Java (The Java Language Specification — http://www.javasoft.com/docs/books/jls/). Перевод на русский язык — http://www.uni-vologda.ac.ru/java/jls/index.html
4. Официальный сайт проекта Eclipse — http://www.eclipse.org/.

А также

1. Дмитрий Рамодин. Начинаем программировать на языке Java.
2. Николай Смирнов. Java 2: Учебное пособие.
3. Картузов А. В. Программирование на языке Java.
4. Вязовик Н.А. Программирование на Java.
5. Алексей Литвинюк. Введение в интегрированную среду разработки Eclipse — http://lib.juga.ru/article/articleview/174/1/69/.

**Приложение 1. Встроенные типы данных и основные конструкции языка**

### 

### Константы

В языке Java можно записывать константы разных типов в разных видах. Перечислим их.

**Целые**

Целые константы можно записывать в трех системах счисления:

* в десятичной форме: +5, -7, 12345678;
* в восьмеричной форме, начиная с нуля: 027, -0326, 0777; в записи таких констант недопустимы цифры 8 и 9;
* в шестнадцатеричной форме, начиная с нуля и латинской буквы х или X: 0xff0a, 0xFC2D, 0x45a8, 0X77FF; здесь строчные и прописные буквы не различаются.

Целые константы хранятся в формате типа int.

В конце целой константы можно записать букву прописную L или строчную l, тогда константа будет сохраняться в длинном формате типа long: +25L, -0371, OxffL, OXDFDF1.

**Действительные**

Действительные константы записываются только в десятичной системе счисления в двух формах:

* c фиксированной точкой: 37.25, -128.678967, +27.035;
* с плавающей точкой: 2.5е34, -0.345е-25, 37.2Е+4; можно писать строчную или прописную латинскую букву Е; пробелы и скобки недопустимы.

В конце действительной константы можно поставить букву F или f, тогда константа будет сохраняться в формате типа float: 3.5f, -45.67F, 4.7e-5f. Можно приписать и букву D (или d): 0.045D, -456.77889d, означающую тип double, но это излишне, поскольку действительные константы и так хранятся в формате типа double.

**Символы**

Для записи одиночных символов используются следующие формы.

* Печатные символы можно записать в апострофах: 'а', 'N', '?'.
* Управляющие символы записываются в апострофах с обратной наклонной чертой:
  + '\n' — символ перевода строки newline с кодом ASCII 10;
  + '\r' — символ возврата каретки CR с кодом 13;
  + '\f' — символ перевода страницы FF с кодом 12;
  + '\b' — символ возврата на шаг BS с кодом 8;
  + '\t' — символ горизонтальной табуляции НТ с кодом 9;
  + '\\' — обратная наклонная черта;
  + '\"' — кавычка;
  + '\'' — апостроф.
* Код любого символа с десятичной кодировкой от 0 до 255 можно задать, записав его не более чем тремя цифрами в восьмеричной системе счисления в апострофах после обратной наклонной черты: '\123' — буква S, '\346' — буква Ж в кодировке СР1251.
* Код любого символа в кодировке Unicode набирается в апострофах после обратной наклонной черты и латинской буквы ц ровно четырьмя шестнад-цатеричными цифрами: '\u0053' — буква S, '\u0416' — буква Ж.

Символы хранятся в формате типа char.

Компилятор и исполняющая система Java работают только с кодировкой Unicode.

### Строки

Строки символов заключаются в кавычки. Управляющие символы и коды записываются в строках точно так же, с обратной наклонной чертой, но без апострофов, и оказывают то же действие. Строки могут располагаться только на одной строке исходного кода, нельзя открывающую кавычку поставить на одной строке, а закрывающую — на следующей. Например:

"Это строка\nс переносом"

"\"Спартак\" — Чемпион!"

Строки символов нельзя начинать на одной строке исходного кода, а заканчивать на другой.

Для строковых констант определена операция сцеплений, обозначаемая плюсом.

"Сцепление " + "строк" дает в результате строку "Сцепление строк".

Чтобы записать длинную строку в виде одной строковой константы, надо после закрывающей кавычки на первой и следующих строках поставить плюс +; тогда компилятор соберет две (или более) строки в одну строковую константу, например:

"Одна строковая константа, записанная "+

"на двух строках исходного текста"

Также нужно учесть, что Windows 95/98/ME вообще не работает с Unicode, a Windows NT/2000 использует для вывода в окно **Command Prompt** шрифт Terminal, в котором русские буквы, расположены в начальных кодах Unicode, почему-то в кодировке СР866, и разбросаны по другим сегментам Unicode.

**Имена**

Имена (names) переменных, классов, методов и других объектов могут быть простыми (общее название — идентификаторы (idenifiers)) и составными (qualified names). Идентификаторы в Java составляются из так называемых букв Java (Java letters) и арабских цифр 0—9, причем первым символом идентификатора не может быть цифра. (Действительно, как понять запись 2е3: как число 2000,0 или как имя переменной?) В число букв Java обязательно входят прописные и строчные латинские буквы, знак доллара $ и знак 'подчеркивания \_, а так же символы национальных алфавитов.

Служебные слова Java, такие как class, void, static, зарезервированы, их нельзя использовать в качестве идентификаторов своих объектов.

Составное имя (qualified name) — это несколько идентификаторов, разделенных точками, без пробелов, например, уже встречавшееся нам имя System.out.println.

### Примитивные типы данных и операции

Все типы исходных данных, встроенные в язык Java, делятся на две группы: примитивные типы (primitive types) и ссылочные типы (reference types).

Ссылочные типы делятся на массивы (arrays), классы (classes) и интерфейсы (interfaces).

Примитивных типов всего восемь. Их можно разделить на логический (иногда говорят булев) тип boolean и числовые (numeric).

К числовым типам относятся целые (integral) и вещественные (floating-point) типы.

Целых типов пять: byte, short, int, long, char.

Символы можно использовать везде, где используется тип int, поэтому JLS причисляет их к целым типам. Например, их можно использовать в арифметических вычислениях, скажем, можно написать 2 + 'ж', к двойке будет прибавляться кодировка Unicode '\u0416' буквы 'ж'. В десятичной форме это число 1046 и в результате сложения получим 1048.

А в записи 2 + "Ж" плюс понимается как сцепление строк, двойка будет преобразована в строку, в результате получится строка "2ж".

Вещественных типов два: float и double.

На рис. 1 показана иерархия типов данных Java.

Поскольку по имени переменной невозможно определить ее тип, все переменные обязательно должны быть описаны перед их использованием. Описание заключается в том, что записывается имя типа, затем, через пробел, список имен переменных, разделенных запятой. Для всех или некоторых переменных можно указать начальные значения после знака равенства, которыми могут служить любые константные выражения того же типа. Описание каждого типа завершается точкой с запятой. В программе может быть сколько угодно описаний каждого типа.

Java — язык со строгой типизацией (strongly typed language).

Разберем каждый тип подробнее.

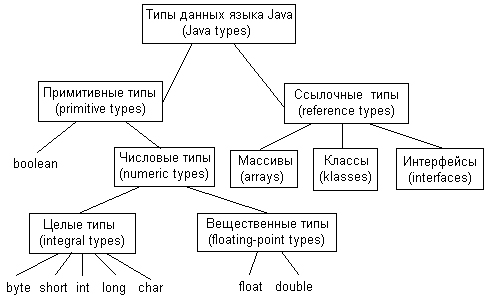


Рис. 1. Типы данных языка Java

**Логический тип**

Значения логического типа boolean возникают в результате различных сравнений и используются, главным образом, в условных операторах и операторах циклов. Логических значении всего два: true (истина) и false (ложь). Это служебные слова Java. Описание переменных этого типа выглядит так:

boolean b = true, bb = false, bool2;

Над логическими данными можно выполнять операции присваивания, например, bool2 = true, в том числе и составные с логическими операциями; сравнение на равенство b == bb и на неравенство b != bb, а также логические операции.

### Логические операции

* отрицание (NOT) ! (обозначается восклицательным знаком);
* конъюнкция (AND) & (амперсанд);
* дизъюнкция (OR) | (вертикальная черта);
* исключающее ИЛИ (XOR) ^ (каре).

Они выполняются над логическими данными, их результатом будет тоже логическое значение true или false. Про них можно ничего не знать, кроме того, что представлено в табл. 1.

**Таблица 1.** Логические операции

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| b1 | b2 | !b1 | b1&b2 | b1|b2 | b1^b2 |
| true | true | false | true | true | false |
| true | false | false | false | true | true |
| false | true | true | false | true | true |
| false | false | true | false | false | false |

Если бы Шекспир был программистом, фразу "То be or not to be" он написал бы так:

2b | ! 2b. (☺)

Кроме перечисленных четырех логических операций есть еще две логические операции сокращенного вычисления:

* сокращенная конъюнкция (conditional-AND) &&;
* сокращенная дизъюнкция (conditional-OR) ||.

Удвоенные знаки амперсанда и вертикальной черты следует записывать без пробелов.

Правый операнд сокращенных операций вычисляется только в том случае, если от него зависит результат операции, т. е. если левый операнд конъюнкции имеет значение true, или левый операнд дизъюнкции имеет значение false.

Это правило очень удобно и ловко используется, например, можно записывать выражения (n != 0) && (m/n > 0.001) или (n == 0) || (m/n > 0.001) не опасаясь деления на нуль.

Практически всегда в Java используются именно сокращенные логические операции.

### Целые типы

Спецификация языка Java, JLS, определяет разрядность (количество байтов, выделяемых для хранения значений типа в оперативной памяти) и диапазон значений каждого типа. Для целых типов они приведены в табл. 2.

**Таблица 2.** Целые типы

|  |  |  |
| --- | --- | --- |
| Тип | Разрядность (байт) | Диапазон |
| byte | 1 | от -128 до 127 |
| short | 2 | от -32768 до 32767 |
| int | 4 | от -2147483648 до 2147483647 |
| long | 8 | от -9223372036854775808 до 9223372036854775807 |
| char | 2 | от '\u0000' до '\uFFFF' , в десятичной форме от 0 до 65535 |

Впрочем, для Java разрядность не столь важна, на некоторых компьютерах она может отличаться от указанной в таблице, а вот диапазон значений должен выдерживаться неукоснительно.

Хотя тип char занимает два байта, в арифметических вычислениях он участвует как тип int, ему выделяется 4 байта, два старших байта заполняются нулями.

Примеры определения переменных целых типов:

byte b1 = 50, b2 = -99, bЗ;

short det = 0, ind = 1;

int i = -100, j = 100, k = 9999;

long big = 50, veryBig = 2147483648L;

char c1 = 'A', c2 = '?', newLine = '\n';

Целые типы хранятся в двоичном виде в дополнительном коде.

Над целыми типами можно производить массу операций. Их набор восходит к языку С, он оказался удобным и кочует из языка в язык почти без изменений.

### Операции над целыми типами

Все операции, которые производятся над целыми числами, можно разделить на следующие группы.

**Арифметические операции**

К арифметическим операциям относятся:

* сложение + (плюс);
* вычитание - (дефис);
* умножение \* (звездочка);
* деление / (наклонная черта — слэш);
* взятие остатка от деления (деление по модулю) % (процент);
* инкремент (увеличение на единицу) ++;
* декремент (уменьшение на единицу) --

Между сдвоенными плюсами и минусами нельзя оставлять пробелы. Сложение, вычитание и умножение целых значений выполняются как обычно, а вот деление целых значений в результате дает опять целое (так называемое "целое деление"), например, 5/2 даст в результате 2, а не 2.5, а 5/(-3) даст -1. Дробная часть попросту отбрасывается, происходит усечение частного. Это поначалу обескураживает, но потом оказывается удобным для усечения чисел.

В Java принято целочисленное деление.

Это странное для математики правило естественно для программирования: если оба операнда имеют один и тот же тип, то и результат имеет тот же тип. Достаточно написать 5/2.0 или 5.0/2 или 5.0/2.0 и получим 2.5 как результат деления вещественных чисел.

Операция деление по модулю определяется так: а % b = а - (а / b) \* b; например, 5%2 даст в результате 1, а 5% (-3) даст, 2, т.к. 5 = (-3) \* (-1) + 2, но (-5)%3 даст -2, поскольку -5 = 3 \* (-1) - 2.

Операции инкремент и декремент означают увеличение или уменьшение значения переменной на единицу и применяются только к переменным, но не к константам или выражениям, нельзя написать 5++ или (а + b)++.

Интересно, что эти операции 'можно записать и перед переменной: ++i, — j. Разница проявится только в выражениях: при первой формe записи (постфиксной) в выражении участвует старое значение переменной и только потом происходит увеличение или уменьшение ее значения. При второй форме записи (префиксной) сначала изменится переменная и ее новое значение будет участвовать в выражении.

### Приведение типов

Результат арифметической операции имеет тип int, кроме того случая, когда один из операндов типа long. В этом случае результат будет типа long.

Перед выполнением арифметической операции всегда происходит повышение типов byte, short, char. Они преобразуются в тип int, а может быть, и в тип long, если другой операнд типа long. Операнд типа int повышается до типа long, если другой операнд типа long. Конечно, числовое значение операнда при этом не меняется.

В таких случаях следует выполнить явное приведение типа. В данном случае это будет сужение типа int до типа short. Оно осуществляется операцией явного приведения, которая записывается перед приводимым значением в виде имени типа в скобках. Определение

short b1=1, b2=2;

short k = (short)(b1 + b2);

будет верным.

Если результат целой операции выходит за диапазон своего типа int или long, то автоматически происходит приведение по модулю, равному длине этого диапазона, и вычисления продолжаются, переполнение никак не отмечается.

В языке Java нет целочисленного переполнения.

### Операции сравнения

В языке Java шесть обычных операций сравнения целых чисел по величине:

* больше >;
* меньше <;
* больше или равно >=;
* меньше или равно <=;
* равно ==;
* не равно !=.

Для записи сложных сравнений следует привлекать логические операции. Например, в вычислениях часто приходится делать проверки вида а < х < b. В данном случае следует написать выражение (а < х) && (х < b), причем здесь скобки можно опустить, написать просто а < х && х < b.

### Вещественные типы

Вещественных типов в Java два: float и double. Они характеризуются разрядностью, диапазоном значений и точностью представления, отвечающим стандарту IEEE 754-1985 с некоторыми изменениями. К обычным вещественным числам добавляются еще три значения.

1. Положительная бесконечность, выражаемая константой POSITIVE\_INFINITY и возникающая при переполнении положительного значения, например, в результате операции умножения 3.0\*6е307.

2. Отрицательная бесконечность NEGATIVE\_INFINITY.

3. "Не число", записываемое константой NaN (Not a Number) и возникающее при делении вещественного числа на нуль или умножении нуля на бесконечность.

Кроме того, стандарт различает положительный и отрицательный нуль, возникающий при делении на бесконечность соответствующего знака, хотя сравнение 0.0 == -0.0 дает true. Операции с бесконечностями выполняются по обычным математическим правилам.

Во всем остальном вещественные типы — это обычные, вещественные значения, к которым применимы все арифметические операции и сравнения, перечисленные для целых типов. Характеристики вещественных типов приведены в табл. 1.4.

В языке Java взятие остатка от деления %, инкремент ++ и декремент — применяются и к вещественным типам.

**Таблица 3.** Вещественные типы

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Разрядность | Диапазон | Точность |
| float | 4 | 3,4е-38 < |х| < 3,4е38 | 7—8 цифр |
| double | 8 | 1,7е-308<|х|<1,7е308 | 17 цифр |

Поскольку к вещественным типам применимы все арифметические операции и сравнения, целые и вещественные значения можно смешивать в операциях. При этом правило приведения типов дополняется такими условиями:

* если в операции один операнд имеет тип double, то и другой приводится к типу double;
* если один операнд имеет тип float, то и другой приводится к типу float;
* в противном случае действует правило приведения целых значений.

### Условная операция

Эта своеобразная операция имеет три операнда. Вначале записывается произвольное логическое выражение, т. е. имеющее в результате true или false, затем знак вопроса, потом два произвольных выражения, разделенных двоеточием, например,

х < 0 ? 0 : х

х > у ? х — у : х + у

Условная операция выполняется так. Сначала вычисляется логическое выражение. Если получилось значение true, то вычисляется первое выражение после вопросительного знака ? и его значение будет результатом всей операции. Последнее выражение при этом не вычисляется. Если же получилось значение false, то вычисляется только последнее выражение, его значение будет результатом операции. Это, например, позволяет написать n == 0 ? да : m / n не опасаясь деления на нуль.

### Приоритет операций

Операции перечислены в порядке убывания приоритета. Операции на одной строке имеют одинаковый приоритет.

1. Постфиксные операции ++ и —.

2. Префиксные операции ++ и —, дополнение ~ и отрицание !.

3. Приведение типа (тип).

4. Умножение \*, деление / и взятие остатка %.

5. Сложение + и вычитание -.

6. Сдвиги <<, >>, >>>.

7. Сравнения >, <, >=, <=.

8. Сравнения ==, !=.

9. Побитовая конъюнкция &.

10. Побитовое исключающее ИЛИ ^.

11. Побитовая дизъюнкция | .

12. Конъюнкция &&.

13. Дизъюнкция | | .

14. Условная операция ?: .

15. Присваивания =, +=, -=, \*=, /=, %=, &=, ^=, |=, <<, >>, >>>.

### Операторы

Как вы знаете, любой алгоритм, предназначенный для выполнения на компьютере, можно разработать, используя только линейные вычисления, разветвления и циклы.

Набор операторов языка Java включает:

* операторы описания переменных и других объектов;
* операторы-выражения;
* операторы присваивания;
* условный оператор if;
* три оператора цикла while, do-while, for;
* оператор варианта switch;
* Операторы перехода break, continue и return;
* блок {};
* пустой оператор — просто точка с запятой.

В языке Java нет оператора goto.

Всякий оператор завершается точкой с запятой. Точка с запятой в Java не разделяет операторы, а является частью оператора (В Pascal’е наоборот).

### Блоки

Блок заключает в себе нуль или несколько операторов с целью использовать их как один оператор в тех местах, где по правилам языка можно записать только один оператор. Блоки операторов часто используются для ограничения области действия переменных и просто для улучшения читаемости текста программы.

### Условный оператор

Условный оператор (if-then-else) в языке Java записывается так:

if (логВыр) оператор1 else оператор2

и действует следующим образом. Сначала вычисляется логическое выражение логвыр. Если результат true, то действует оператор! и на этом действие условного оператора завершается, оператор2 не действует, далее будет выполняться следующий за if оператор. Если результат false, то действует оператор2, при этом оператор1 вообще не выполняется.

Условный оператор может быть сокращенным (if-then statement):

if (логВыр) оператор1

и в случае false не выполняется ничего.

Очень часто одним из операторов является снова условный оператор, например:

if (a >= 10) if (b <= 20) x = 0; else x = 1;

Правило таково: ветвь else относится к ближайшему слева условию if, не имеющему своей ветви else. Изменить этот порядок можно с помощью блока:

if (a > 10) {if (b < 20) x = 0; else x = 1;}

### Операторы цикла

Основной оператор цикла — оператор while — выглядит так:

while (логВыр) оператор

Вначале вычисляется логическое выражение логВыр; если его значение true, то выполняется оператор, образующий цикл. Затеем снова вычисляется логВыр и действует оператор, и так до тех пор, пока не получится значение false. Если логВыр изначально равняется false, то оператор не будет выполнен ни разу. Предварительная проверка обеспечивает безопасность выполнения цикла, позволяет избежать переполнения, деления на нуль и других неприятностей. Поэтому оператор while является основным, а в некоторых языках и единственным оператором цикла.

Можно организовать и бесконечный цикл:

while (true) оператор

Второй оператор цикла — оператор do-while — имеет вид

do оператор while (логВыр)

Здесь сначала выполняется оператор, а потом происходит вычисление логического выражения логвыр. Цикл выполняется, пока логвыр остается равным true.

Существенное различие между этими двумя операторами цикла только в том, что в цикле do-while оператор обязательно выполнится хотя бы один раз.

Третий оператор цикла — оператор for — выглядит так:

for (списокВыр; логВыр; списокВыр2) оператор

Работа цикла полностью аналогична языкам C/C++

Вместо списокВыр1 может стоять одно определение переменных обязательно с начальным значением. Такие переменные известны только в пределах этого цикла.

Любая часть оператора for может отсутствовать: цикл может быть пустым, выражения в заголовке тоже, при этом точки с запятой сохраняются. Можно задать бесконечный цикл:

for (;;) оператор

### Оператор continue и метки

Оператор continue используется только в операторах цикла. Он имеет две формы. Первая форма состоит только из слова continue и осуществляет немедленный переход к следующей итерации цикла. В очередном фрагменте кода оператор continue позволяет обойти деление на нуль:

for (int i = 0; i < N; i++){

if (i '== j) continue;

s += 1.0 / (i - j);

}

Вторая форма содержит метку:

continue метка

метка записывается, как все идентификаторы, из букв Java, цифр и знака подчеркивания, но не требует никакого описания. Метка ставится перед оператором или открывающей фигурной скобкой и отделяется от них двоеточием. Так получается помеченный оператор или помеченный блок.

Метка не требует описания и не может начинаться с цифры.

Вторая форма используется только в случае нескольких вложенных циклов для немедленного перехода к очередной итерации одного из объемлющих циклов, а именно, помеченного цикла.

### Оператор break

Оператор break используется в операторах цикла и операторе варианта для немедленного выхода из этих конструкций.

Оператор break метка

применяется внутри помеченных операторов цикла, оператора варианта или помеченного блока для немедленного выхода за эти операторы. Следующая схема поясняет эту конструкцию.

Ml: { // Внешний блок

М2: { // Вложенный блок — второй уровень

М3: { // Третий уровень вложенности...

if (что-то случилось) break M2;

// Если true, то здесь ничего не выполняется

}

// Здесь тоже ничего не выполняется

}

// Сюда передается управление

}

Обратите внимание, что метка ставится перед блоком или оператором, а управление передается за этот блок или оператор.

### Оператор варианта

Оператор варианта switch организует разветвление по нескольким направлениям. Каждая ветвь отмечается константой или константным выражением какого-либо целого типа (кроме long) и выбирается, если значение определенного выражения совпадет с этой константой. Вся конструкция выглядит так.

switch (целВыр){

case констВыр1: оператор1

case констВыр2: оператор2

. . . . .

case констВырN: операторN

default: операторDef

}

Стоящее в скобках выражение целвыр может быть типа byte, short, int, char, но не long. Целые числа или целочисленные выражения, составленные из констант, констВыр тоже не должны иметь тип long.

Оператор варианта выполняется так. Все константные выражения вычисляются заранее, на этапе компиляции, и должны иметь отличные друг от друга значения. Сначала вычисляется целочисленное выражение целВыр. Если оно совпадает с одной из констант, то выполняется оператор, отмеченный этой константой. Затем выполняются все следующие операторы, включая и операторDef, и работа оператора варианта заканчивается.

Если же ни одна константа не равна значению выражения, то выполняется операторDef и все следующие за ним операторы. Поэтому ветвь default должна записываться последней. Ветвь default может отсутствовать, тогда в этой ситуации оператор варианта вообще ничего не делает.

Таким образом, константы в вариантах case играют роль только меток, точек входа в оператор варианта, а далее выполняются все оставшиеся операторы в порядке их записи.

В отличие от Pascal’я после выполнения одного варианта оператор switch продолжает выполнять все оставшиеся варианты.

Чаще всего необходимо "пройти" только одну ветвь операторов. В таком случае используется оператор break, сразу же прекращающий выполнение оператора switch. Может понадобиться выполнить один и тот же оператор в разных ветвях case. В этом случае ставим несколько меток case подряд. Вот простой пример.

switch(dayOfWeek){

case 1: case 2: case 3: case 4:case 5:

System.out.println("Week-day");, break;

case 6: case 7:

System.out.println("Week-end"); break;

default:

System.out.printlnt"Unknown day");

}

He забывайте завершать варианты оператором break.

**Приложение 2. Работа с массивами**

В классе Arrays из пакета java.utii собрано множество методов для работы с массивами. Их можно разделить на четыре группы.

Восемнадцать статических методов сортируют массивы с разными типами числовых элементов в порядке возрастания чисел или просто объекты в их естественном порядке.

Восемь из них имеют простой вид

static void sort(type[] a)

где type может быть один из семи примитивных типов byte, short, int, long, char, float, double или тип Object.

Восемь методов с теми же типами сортируют часть массива от индекса from включительно до индекса to исключительно:

static void sort(type[] a, int from, int to)

Оставшиеся два метода сортировки упорядочивают массив или его часть с элементами типа Оbject по правилу, заданному объектом с, реализующим интерфейсComparator:

static void sort(Object[] a, Comparator c)

static void sort(Object[] a, int from, int to, Comparator c)

После сортировки можно организовать бинарный поиск в массиве одним из девяти статических методов поиска. Восемь методов имеют вид

static int binarySearch(type[] a, type element)

где type — один из тех же восьми типов. Девятый метод поиска имеет вид

static int binarySearch(Object[] a, Object element, Comparator c).

Он отыскивает элемент element в массиве, отсортированном в порядке, заданном объектом с.

Методы поиска возвращают индекс найденного элемента массива. Если элемент не найден, то возвращается отрицательное число, означающее индекс, с которым элемент был бы вставлен в массив в заданном порядке, с обратным знаком.

Восемнадцать статических методов заполняют массив или часть массива указанным значением value:

static void fill(type[], type value)

static void fill(type[], int from, int to, type value)

где type — один из восьми примитивных типов или тип Оbject. Наконец, девять статических логических методов сравнивают массивы:

static boolean equals(type[] al, type[] a2)

где type — один из восьми примитивных типов или тип Object.

Массивы считаются равными, и возвращается true, если они имеют одинаковую длину и равны элементы массивов с одинаковыми индексами.

В листинге1 приведен простой пример работы с этими методами.

**Листинг 1.** Применение методов класса Arrays

import java.utii.\*;

class ArraysTest{

public static void main(String[] args){

int[] a = {34, -45, 12, 67, -24, 45, 36, -56};

Arrays.sort(a) ;

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

System.out.print (a[i]. + " ");

System.out.println();

Arrays.fill(a, Arrays.binarySearch(a, 12), a.length, 0);

for (int i = 6; i < a.length; i++)

System.out.print(a[i] + " ");

System.out.println();

}

}

# Приложение 3. Введение в интегрированную среду разработки Eclipse

Eclipse – это расширяемая, open-source интегрированная среда разработки (IDE, Integrated Development Environment). Этот проект был запущен в ноябре 2001 года, когда IBM выделили исходный код из Websphere Studio Workbench, ценой в 40 миллионов долларов, и сформировали Eclipse Consortium для управления разработкой этого инструмента.

Первоначальная цель заключалась в том, чтобы "разработать стабильную, полнофункциональную, промышленную платформу, качества коммерческого продукта, для разработки интегрируемых инструментов". На настоящий момент Eclipse Consortium фокусируется на следующих проектах:

1. The Eclipse Project (http://www.eclipse.org/eclipse/index.html) – ответственен за разработку непосредственно Eclipse IDE (платформу для сборки прочих инструментов Eclipse), Java Development Tools (JDT) и Plug-In Development Environment (PDE), используемую для предоставления возможности расширения самой платформы.
2. The Eclipse Tools Project (http://www.eclipse.org/tools/index.html) – занимается созданием оптимальных инструментальных средств для платформы Eclipse. В текущие подпроекты входят: Cobol IDE, C/C++ IDE, а также инструмент моделирования EMF.
3. The Eclipse Technology Project (http://www.eclipse.org/technology/index.html) – занимается технологическими исследованиями, инкубацией и образованием по части использования платформы Eclipse.

Совместно с JDT, платформа Eclipse предоставляют множество различных возможностей, которые вы могли наблюдать в коммерческих IDE: подсветка синтаксиса в редакторе, компиляция кода, отладчик уровня исходного кода с поддержкой "нитей" (threads), навигатор по классам, файловый менеджер и менеджер проектов, интерфейсы для стандартных контролирующих систем исходного кода, таких как, например, CVS и ClearCase.

Помимо этого Eclipse содержит ряд уникальных возможностей, например, рефакторинг кода (http://www.refactoring.com/), автоматическое обновление и сборка кода (посредством Update Manager), список задач, поддержка возможности тестирования модулей с помощью JUnit (http://www.junit.org/), а также интеграция с инструментом сборки приложений Jakarta Ant (http://jakarta.apache.org/ant/index.html).

Несмотря на большое количество стандартного набора возможностей, Eclipse отличается от традиционных IDE по нескольким фундаментальным особенностям. Может быть, самая интересная возможность Eclipse – это абсолютная нейтральность относительно платформы и языка программирования. Вдобавок к эклектичному набору языков программирования, которые поддерживаются Eclipse Consortium (Java, C/C++, Cobol), существует множество сторонних проектов, с помощью которых вы можете обеспечить поддержку интересующего вас языка программирования в Eclipse. На сегодняшний день существуют реализации следующих популярных языков программирования: Python, Eiffel, PHP, Ruby, и C#.

Платформа Eclipse предоставляется, благодаря Eclipse Consortium, в виде заранее скомпилированных исполняемых файлов для Windows, Linux, Solaris, HP-UX, AIX, QNX, и Mac OS X. Очень много внимания концентрируется вокруг архитектурной системы plug-in'ов этой платформы, а также "богатых" API (Application Programming Interface), поставляемых с Plug-in Development Environment для расширения Eclipse. Добавить поддержку нового типа редактора, просмотрщика (панели) или языка программирования до безобразия просто, благодаря хорошо спроектированным API и строительным блокам, которые предоставляет Eclipse.

Имея приблизительно сотню проектов, по разработке plug-in'ов, имея ресурсы, предоставляемые такими промышленными гигантами, как IBM, HP и Rational (которая недавно была куплена IBM), и дизайн Erich Gamma, который помогает быстро разобраться в процессе разработки – будущему Eclipse предстоит быть весьма ярким, интересным и перспективным.

### Установка

Бьюсь об заклад, что вы сейчас только и думаете о том, где бы достать копию этого самого Eclipse для вашей платформы и как бы это его установить? :) Первое, о чем вам для начала нужно позаботиться – это убедиться в наличии подходящей, рабочей JRE (Java Runtime Evironment). Хотя Eclipse и создавался в расчете на возможность компиляции кода как для JVM (Java Virtual Machine) версии 1.3, так и для версии 1.4, однако текущие версии Eclipse были спроектированы таким образом, чтобы запускаться на JVM версии 1.3. Если вы не уверены, где можно достать "виртуальную" Java-машину (JVM) для вашей платформы, вы можете найти подробные инструкции и дополнительную информацию об этом на сайте http://www.eclipse.org/.

Как только вы убедились в наличии JVM или установили подходящую ее версию, вы готовы к установке Eclipse. Для этого посетите download-раздел сайта проекта Eclipse (http://www.eclipse.org/downloads/) и скачайте оттуда последний релиз, собранный специально под используемую вами платформу. Все дистрибутивы представляют собой .zip-архивы. Распакуйте скачанный архив в подходящую директорию и потратьте немного времени на изучение всех файлов, содержащихся в директории readme.

Хотите - верьте, хотите – нет, но на этом инсталляция завершена. Если вы правильно установили JVM и корректно распаковали архив с Eclipse – вы можете приступать к процессу первого запуска этой IDE. Все дистрибутивы, содержащие скомпилированную версию платформы, предоставляют программы для запуска, которые располагаются в основной директории: eclipse. Имя этой программы запуска изменяется в зависимости от платформы, под которую вы скачали дистрибутив: eclipse.exe под Windows, eclipse под Solaris, и так далее. При первом запуске приложение Eclipse выполнит некоторые оставшиеся задачи по установке (например, создание workspace-директории для хранения файлов проектов) до того, как приложение окончательно будет готово к работе.

### Интерфейс

Итак, когда у вас уже есть установленный и запущенный Eclipse, самое время начать им пользоваться. После того как вы запустили Eclipse, вы должны увидеть приблизительно следующее:

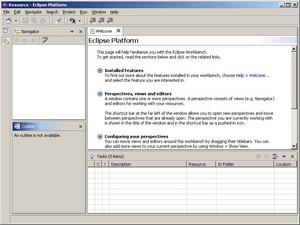


Рисунок 1

Как вы можете заметить, Eclipse IDE имеет вполне стандартное содержание меню:



Рисунок 2

Помимо меню, здесь ярко видно сходство панели инструментов и системы закладок с такими же в других средах разработки приложений.

### Пример приложения

Панели **File Navigator**, **Code Outline** и **Task List** отображены, но на данный момент не содержат никаких данных. Чтобы продолжить знакомство с Eclipse мы создадим простое Swing-приложение программу-калькулятор. Распишем действия по шагам:

1. Выберите пункт меню **File->New->Project...,** чтобы начать создание вашего первого Java-проекта. В окне "мастера" выберите **Java** среди предложенных типов проектов из левого списка и Java Project в правом списке, после чего нажмите кнопку **Next**.

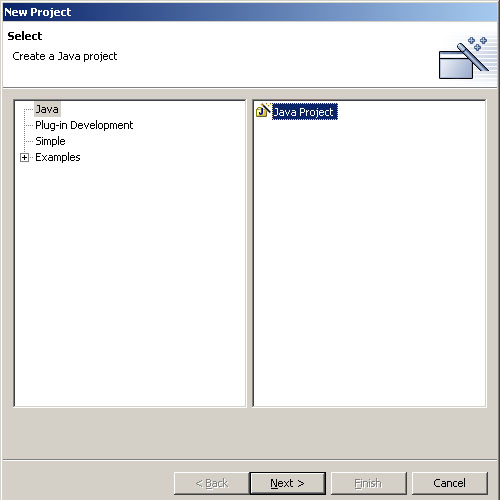


Рисунок 3

2. В качестве имени проекта введите Calculator и продолжите создание проекта, нажав кнопку **Next**.

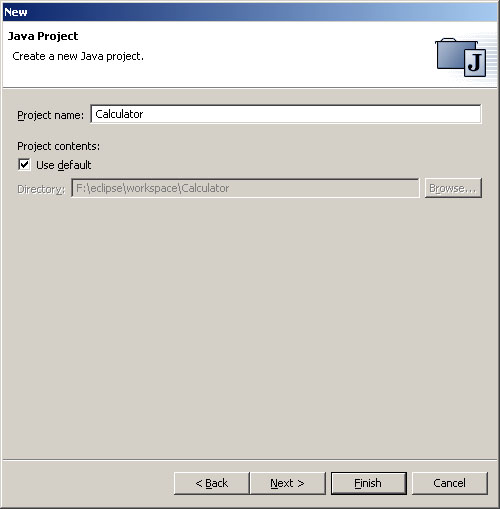


Рисунок 4

3. И, наконец, на завершающем этапе "мастер" создания нового проекта дает вам возможность определить директории, где будут храниться файлы исходного кода и class-файлы, определить подпроекты, которые могут быть использованы создаваемым нами, текущим проектом, а также любые библиотеки, которые мы предполагаем использовать в своем приложении. Установите переключатель на значение **Use source folder contained in the project**, после чего нажмите кнопку **Create New Folder...**. В качестве имени новой директории для хранения исходного кода введите src. Нажмите **Yes**, когда Eclipse спросит вас о том, изменить ли директорию с собираемым приложением на Calculator/bin.

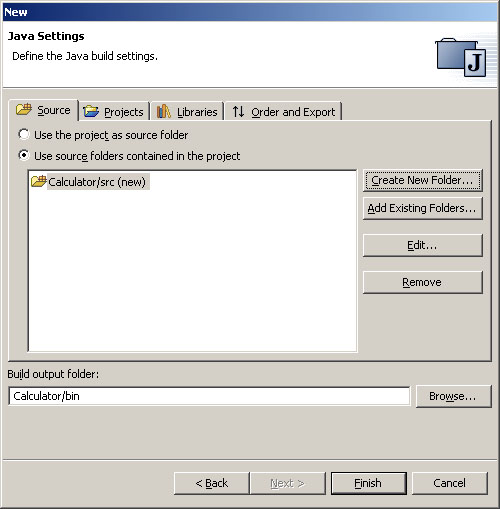


Рисунок 5

4. Нажмите кнопку **Finish**, чтобы дать возможность Eclipse завершить создание нового проекта.

Как только вы создали новый проект, вы можете заметить, что внешний вид Eclipse несколько изменился (этого и следовало ожидать): панель **Outline** переместила в другую часть окна, панель **Navigator** была заменена панелью **Package Explorer** и т.д.

Вот какой вид приняло окно Eclipse после того, как мы создали новый проект:



Рисунок 6

Такая планировка называется **Java Perspective**. "Ракурс", в понятии Eclipse, определяет какой-либо сохраненный порядок расположения любого количества различных панелей редактора и просмотрщиков. Eclipse поставляется с рядом "ракурсов" определенных по умолчанию (**Resource**, **Java**, **Debug** и другие), которые могут быть изменены в соответствии с вашими нуждами, или же вы можете создавать свои собственные ракурсы. Ракурсы управляются при помощью элементов меню **Window** или специальной панели инструментов, которая обычно располагается вдоль левой границы окна Eclipse.



Рисунок 7

Следующим шагом в создании нашего Java-проекта нужно создать директории, которые будут содержать весь наш исходный код. Для этого переключитесь в ракурс **Resource**, используя пункт меню **Window->Open Perspective->Resource**. В панели **Navigator** разверните узлы дерева структуры папок таким образом, чтобы вы могли видеть узел-папку src. Выделите этот узел, и после выполните пункт меню **File->New->Folder**. В появившемся диалоговом окне, убедитесь, что директория src выбрана и затем введите в качестве имени новой, в поле **Folder Name**, имя com.

Создавать большое количество папок с помощью диалога **New Folder** было бы не очень удобно и утомительно. К счастью, Eclipse может предоставить вам возможность самим создавать нужные директории с помощью вашего любимого файлового менеджера: командной строки, Windows Explorer и прочих. Используя один из предложенных методов, создайте в папке com папку devious, а в ней папку calculator. Как только вы создали эти папки, выберите любую папку-узел в панели Navigator и, с помощью элемента меню **File->Refresh**, обновите содержимое этой панели. Таким образом, Eclipse произведет просмотр файловой системы и обновит панель проекта в соответствии с теми изменениями, что вы внесли (создали новые директории). На этом этапе ваше окно Eclipse должно иметь примерно следующий вид:

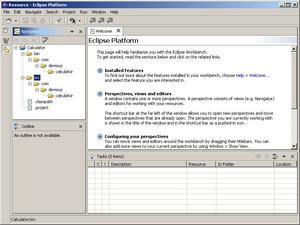


Рисунок 8

И еще одно замечание по поводу создания файлов: на ранних этапах создания проекта, вы, скорее всего, достаточно часто будете запускать свое приложений, создавать новые файлы и папки, добавлять новый код, компилировать, и тестировать свой код снова. Важно помнить, что расположение меню в Eclipse зависит от используемого ракурса. Так, если вы переключаетесь в ракурс **Resource**, чтобы создать новый файл или папку в панели **Navigator**, вы обнаружите, что меню **Run**, например, в корне отличается от меню **Run** в ракурсе **Java**. Одно из решений этой часто встречающейся проблемы – это использование меню **Windows->Show View** с тем, чтобы отобразить панель **Navigator** в ракурсе **Java**. Если вас полностью устроит полученный ракурс то, воспользовавшись элементом меню **Window->Save Perspective As...**, вы сможете сохранить ваши изменения в новом ракурсе, либо переписать существующий ракурс **Java**.

**Добавление кода**

Теперь, когда у нас есть созданная структура директорий, мы можем начинать добавление исходного кода в наш проект. Мы разделим нашу программу-калькулятор на три класса (файла): CalcModel.java, CalcPanel.java, и Calculator.java. Скачайте .zip-архив с исходным кодом этой программы отсюда http://www.onjava.com/onjava/2002/12/11/examples/calculator.zip. После этого распакуйте этот архив и поместите три упомянутых выше файла в директорию com/devious/calculator. Снова воспользуемся пунктом меню **File->Refresh**, чтобы обновить содержимое панели Project Manager. Вот, что вы должны увидеть после выполнения этих действий:

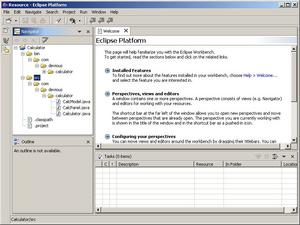
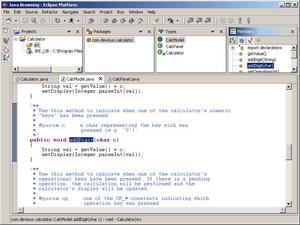


Рисунок 9

CalcPanel.java представляет основу интерфейса пользователя нашего калькулятора. Calculator.java выполняет подготовительные инициализационные функции и запускает GUI (графический интерфейс пользователя). Самая же важная часть нашей программы находится в файле CalcModel.java, который занимается непосредственно реализацией фактической математики нашего калькулятора: реагирование на события, выполнение математических вычислений, обновление отображаемой информации, и т.д. Самый простой способ просмотреть исходный код – это переключиться из ракурса **Java** в **Java Browsing**. Для этого используйте **Open Perspetive** из меню **Window** или кликните на кнопке **Open**, которая находится на панели инструментов для работы с ракурсами.

В то время как вы перемещаетесь по исходному коду на панели **Package Explorer**, вы можете заметить, что панель **Outline** становится ненужной вам на данном этапе. Ее можно на время убрать с экрана, кликнув по крестику в правом углу этой панели. Когда она вам понадобится, вы можете воспользоваться пунктом меню **Window->Show View->Outline**, чтобы отобразить ее вновь.



### Рисунок 10

### Запуск и отладка кода

Итак, вы скачали и установили Eclipse, создали новый Java-проект и добавили необходимые файлы и папки для своего простого приложения. Настало время проверить это приложение в деле, запустив его. Для этого вам нужно установить ракурс **Java**, чтобы выполнить шаги, которые необходимы для запуска приложения:

1. Из основного меню **Run** выберите элемент **Run...**

2. В появившемся диалоге "мастера" запуска, выберите "Java Application" из списка **Launch Configurations** и нажмите кнопку **New**

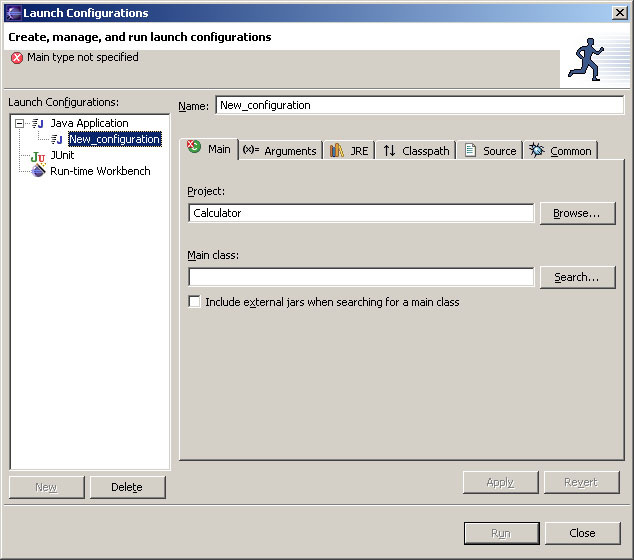


Рисунок 11

3. В качестве имени новой конфигурации введите Calculator

4. Нажмите кнопку **Search...** и выберите основным классом класс Calculator в поле **Main class**, или просто введите в поле ввода полное имя класса: com.devious.calculator.Calculator.

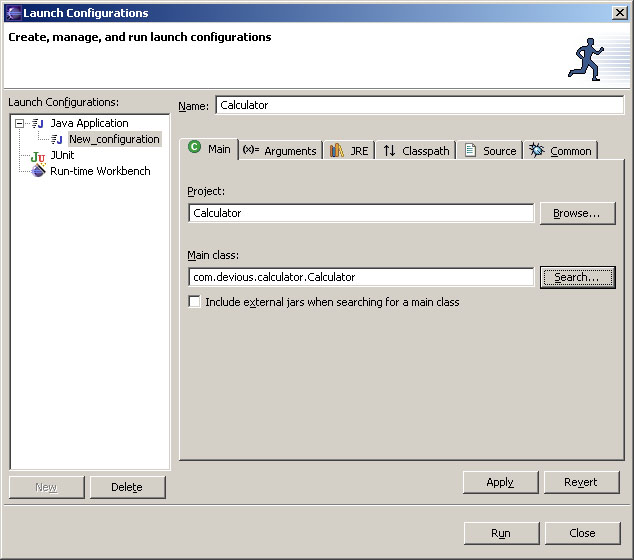


Рисунок 12

5. Нажмите кнопку **Run**, чтобы сохранить установленную конфигурацию и запустить приложение.

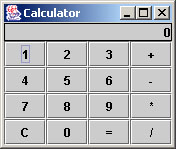


Рисунок 13

Создав однажды конфигурацию запуска своего приложения, вы можете запускать его вновь с помощью меню **Run->Run History** или кнопки **Run** на панели инструментов. Если вы потратите достаточно времени, "играя" с этим калькулятором, вы обнаружите, что он работает в основном правильно, но только с одном "но": он не хочет правильно выполнять операцию вычитания! Время пустить в ход отладчик исходного кода Eclipse и обнаружить эту злосчастную ошибку. Если вы следовали всем инструкциям, приведенным выше в этой статье, то вы можете запустить это приложение с отладчиком с той же конфигурацией, которую мы определили для обычного запуска. Для этого выберите в меню пункт **Run->Debug History->Calculator**.

Когда вы запускаете свое приложение вместе с отладчиком, Eclipse автоматически переключается в ракурс **Debug**:

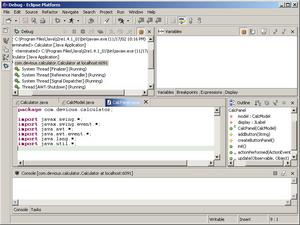


Рисунок 14

Поскольку, обнаруженная нами, проблема сконцентрирована вокруг функции вычитания, мы установим точку прерывания (breakpoint) в обработчик события для кнопок "-" и "=". Для начала, нам нужно переключиться в закладку редактора класса CalcPanel. (Если у вас нет закладки с исходным кодом этого класса, вам нужно открыть файл с именем CalcPanel.java, предварительно переключившись в ракурс **Java** или **Resource** и потом вернуться обратно в ракурс **Debug**).

Воспользуйтесь панелью **Outline**, чтобы определить местонахождение метода actionPerformed. Переместитесь немного ниже, пока не обнаружите выражение case для знака "-" (строка 126). Если вы выполните двойной клик на левой границе этой строки, Eclipse автоматически установит точку прерывания в ней. Переместитесь немного ниже и установите еще одну точку прерывания на вызове метода model.calculate() в обработчике события кнопки "=" (строка 138).

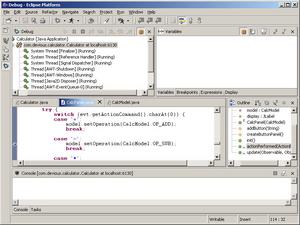


Рисунок 15

Теперь переключитесь в окно калькулятора и нажмите несколько кнопок. Например, "2", "4", "6" и "-". Как только отладчик натыкается на точку прерывания, вы можете видеть появившиеся кнопки на панели инструментов **Debug**, которые служат для пошагового прохождения по исходному коду, причем можно делать шаги по коду и, не заходя внутрь методов, выполнять их автоматически, или же, заходя внутрь, проходиться и по их исходному коду.



Рисунок 16

Если мы зайдем внутрь метода setOperation(), станет понятно, что он выполняют достаточно тривиальную функцию: сохраняет код операции в поле pendingOp. Теперь нажмите кнопку **Resume** на панели инструментов **Debug**, чтобы продолжить выполнение программы.

Замечание: Основная ошибка новых пользователей Eclipse в том, что они иногда нажимают кнопки **Debug** или **Run** на основной панели инструментов (что приводит к запуску еще одной копии приложения) вместо того, чтобы нажать **Resume**.

Теперь нажмите еще пару кнопок (например, "1", "2", "3") и после этого кнопку "=". Таким образом, мы оказываемся внутри метода calculate(), проблема оказалась очевидна: кто-то скопировал код из варианта OP\_ADD, но забыл заменить знак + на -. Проблема решена! :)

**Исправление ошибки методом HotSwap**

Если для запуска приложения вы использовали JVM версии 1.4, то нам остался всего один завершающий шаг на пути к исправлению этой ошибки в нашем примере. Вместо того, чтобы выходить из программы-калькулятора, вносить изменения в исходный код, компилировать заново весь проект, и только потом запускать новую сессию отладчика, мы можем изменить все "налету". В то время как отладчик все еще запущен, измените в исходном коде строку с:

setDisplay(previousValue + currentValue);

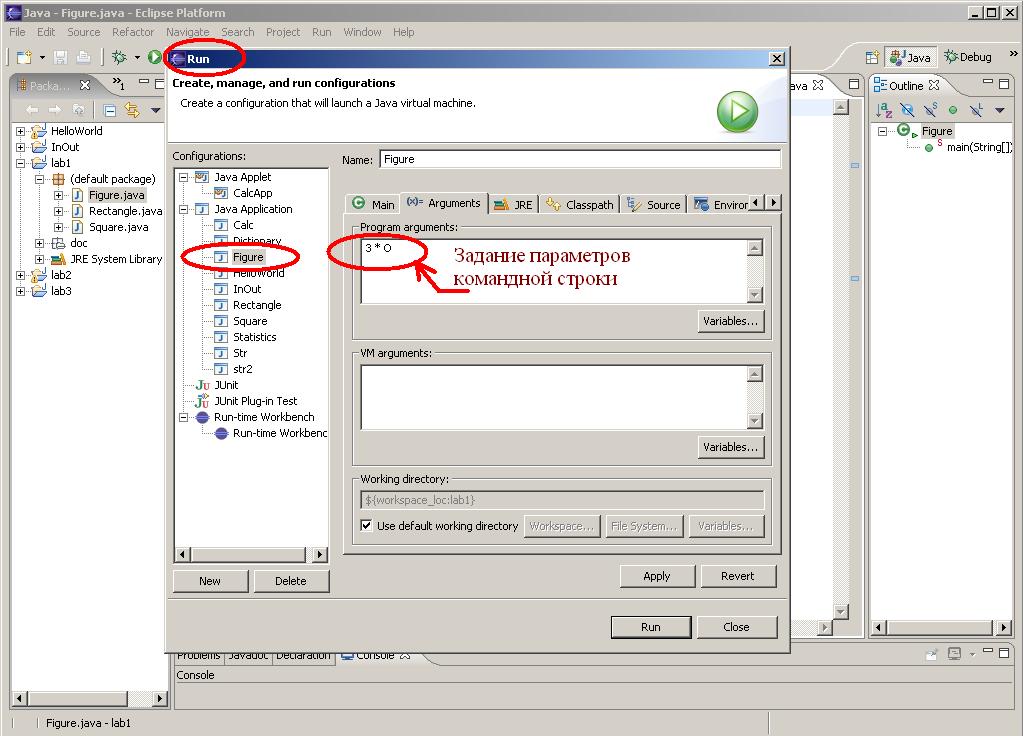
// на:setDisplay(previousValue - currentValue);

Сохраните этот файл, после чего нажмите кнопку **Resume**, чтобы "отпустить" приложение и продолжить работу с ним. Используя кнопку **C**, чтобы сбросить его, и повторите действия, которые мы проделали ранее. Теперь все работает, как положено!

То, что мы сейчас продемонстрировали – это поддержка Eclipse'ом новой возможности (которая носит название HotSwap) JVM версии 1.4. Теперь Java Platform Debugger Architecture (JPDA) поддерживает возможность замены исходного кода у выполняющегося приложения. Это особенно полезно, когда запуск приложения или нахождение места, где оно "рушится", отнимает много времени.

**Установка параметров командной строки в IDE Eclipse.**

Чтобы установить параметры командной строки, надо выбрать меню Run->Run…, затем ввести данные так, как показано на рисунке.



### Ресурсы

* eclipse.org (http://www.eclipse.org/)
* Eclipse Project FAQ (http://www.eclipse.org/eclipse/faq/eclipse-faq.html)
* Eclipse Platform Technical Overview (http://www.eclipse.org/whitepapers/eclipse-overview.pdf)

**Приложение 4. Листинг программы, подсчитывающей , как долго нужно откладывать деньги, чтобы к выходу на пенсию накопить заданную сумму, при условии, что ежегодно откладывается одна и та же сумма и размер счета возрастает в соответствии с процентной ставкойJava**

import javax.swing.\*;

public class Retirement {

public static void main(String[] args) {

// чтение входных данных

String input = JOptionPane.showInputDialog

("Сколько денег вам нужно для пенсии?");

double goal = Double.parseDouble(input);

input = JOptionPane.showInputDialog

("Каков объем вашего ежегодного взноса?");

double payment = Double.parseDouble(input);

input = JOptionPane.showInputDialog

("Процентная ставка, %:?");

double interestRate = Double.parseDouble(input);

// Обновлять баланс счета, пока не накопится нужная сумма

double balance = 0;

int years = 0;

while (balance < goal)

{

// Добавить ежегодный взнос и процент

balance += payment;

double interest = balance\*interestRate/100;

balance += interest;

years++;

}

System.out.println

("Можете идти на пенсию через "+ years + "лет");

System.exit(0);

}

}

# Приложение 5. Листинг программы, иллюстрирующей работу с одномерными массивами (создание и сортировка массива)

import java.util.\*;

import javax.swing.\*;

public class Lottery {

public static void main(String[] args){

String input = JOptionPane.showInputDialog

("Сколько номеров нужно угадать?");

int k = Integer.parseInt(input);

input = JOptionPane.showInputDialog

("Чему равен наибольший из возможных номеров?");

int n = Integer.parseInt(input);

// Заполняем массив числами 1..n

int[] numbers = new int[n];

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

numbers[i] = i+1;

// Генерируем k чисел и помещаем их во второй массив

int[] result = new int[k];

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

{

// Генерируем случайный индекс от 0 до n -1

int r = (int)(Math.random()\*n);

// Помещаем элемент в случайную ячейку

result[i] = numbers[r];

// Перемещаем последний элемент в случайную ячейку

numbers[r] = numbers[n-1];

n--;

}

// Выводим на печать упорядоченный массив

Arrays.sort(result);

System.out.println

("Поставте на следующую комбинацию - не пожалеете!");

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

System.out.println(result[i]);

System.exit(0);

}

}

# Приложение 6. Спецификация класса Figure

### 

### Обязательная часть.

import java.io.\*;//подключение пакета с классами ввода/вывода

public class Figure {

public static void main(String[] args) {

int n;//Например, длина стороны.

char outline='\*', fill=' '; //Символы, реализующие контур и заливку.

//…

try{

switch (args.length){ //Размер массива args[] – число слов-аргументов командной строки

case 1: {n=Integer.parseInt(args[0]);break;} //Преобразвание 1-го слова в int

case 2: {n=Integer.parseInt(args[0]); outline=args[1].charAt(0);break;}

//outline-это первый символ 2-го слова командной строки

//…

case 0: {

System.out.println("Введите длину стороны квадрата (размер стрелки и т.д.)");

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

//Чтение строки из входного потока:

n=Integer.parseInt(br.readLine()); //и преобразование в int

break;

}

} //окончание try{}

catch(Exception ex){

System.out.println(ex.getMessage());

System.exit(0); //выход из программы

}

for (int i=1;i<=n;i++){

//…

for (int j=1;j<=n;j++){

//…

}

System.out.println();

}

} //Конец public static void main(…)

} //Конец программы (класса Figure())

### Необязательная часть (Вывод в файл).

try{

File f = new File("d:\\Figure.txt"); //Создаем объект “Файл”. Физически это файл d:\Figure.txt

if (!f.exists()) {f.createNewFile();} //Если файл не существует, создаем его.

FileWriter fr = new FileWriter(f); //fr-объект класса FileWriter, у которого есть набор методов //для записи в файл. Например,

// fr.write(outline) эквивалентно System.out.print(outline)

for (int i=1;i<=n;i++){

//…

for (int j=1;j<=n;j++){

//…

}

fr.write ("\r\n"); //Переход на новую строку

}

fr.close(); //Поток обязательно надо закрыть!

}

catch(Exception e){

System.out.println(e.getMessage());

System.exit(0); }